

Constructing Security Functions for Resource-Limited Devices With Memory Fingerprints

by

Yang Su

B. Eng. (Electrical & Electronic Engineering),
with First Class Honours,
The University of Adelaide, Australia, 2015

Thesis submitted for the degree of

Doctor of Philosophy

in

School of Computer & Mathematical Science
Faculty of Sciences, Engineering, & Technology
The University of Adelaide

January 2023

Supervisors:

A/Prof. Damith Chinthana Ranasinghe,
School of Computer Science,
The University of Adelaide

Dr. Omid Kavehei,
School of Biomedical Engineering,
The University of Sydney

Contents

Contents	iii
Abstract	xiii
HDR Thesis Declaration	xv
Acknowledgments	xvii
Thesis Conventions	xix
List of Abbreviations	xxi
Awards and Scholarships	xxv
Publications	xxvii
List of Figures	xxix
List of Tables	xxxv
Chapter 1. Introduction	1
1.1 Introduction	2
1.2 Challenges and Opportunities	4
1.2.1 Challenges	5
1.2.2 Opportunity 1: Secure Wireless Firmware Update	8
1.2.3 Opportunity 2: Security Functions from Device Memory Fingerprints	9
1.3 Research Questions	11
1.4 Summary of Original Contributions	11
1.5 Thesis Structure	16
Chapter 2. Fundamentals of CRFID and Memory PUFs	21
2.1 Notations and Conventions	22

- 2.2 Battery-less Edge Computing: CRFID Devices 22
 - 2.2.1 RFID Preliminaries 22
 - 2.2.2 RFID Protocol Overview 23
 - 2.2.3 Computational RFID Devices 25
 - 2.2.4 RF Energy Harvesting 26
- 2.3 Memory PUFs or Memory Fingerprinting 27
 - 2.3.1 Memory PUFs 27
 - 2.3.2 Memory PUF Evaluation Measures 33
 - 2.3.3 Reliable Key Generation with PUFs 34
 - 2.3.4 Memory Fingerprinting and Memory PUF Datasets 38
- 2.4 Chapter Summary 40

Chapter 3. Securing Wireless Code Updates to Intermittently Powered Devices 41

- 3.1 Motivation and Contribution 42
 - 3.1.1 Chapter Overview 44
 - 3.1.2 Notations and Concepts 44
- 3.2 SecuCode: Protocol Design 44
 - 3.2.1 Adversary Model 45
 - 3.2.2 SecuCode Protocol 46
- 3.3 Implementation 49
 - 3.3.1 Protocol Control Flow on a Transponder 49
 - 3.3.2 Random Number Generator 51
 - 3.3.3 Lightweight Physically Obfuscated Key Derivation 53
 - 3.3.4 Message Authentication Code 58
 - 3.3.5 Intermittent Execution Model 60
 - 3.3.6 End-to-end Implementation 62
- 3.4 Experimental Results and Analysis 64
 - 3.4.1 Physically Obfuscated Key Derivation 65
 - 3.4.2 SecuCode Implementation Footprint 68
 - 3.4.3 SecuCode Overhead and IEM Settings 68
 - 3.4.4 Summary 71

3.4.5	Security Analysis	72
3.5	Related Work and Discussion	74
3.5.1	In-application Behaviour Modification	75
3.5.2	Wireless Code Dissemination	75
3.5.3	End-to-end Comparison	76
3.6	Acknowledgement	78
3.7	Chapter Summary	78
Chapter 4. Secure Simultaneous Code Updates to Multiple CRFID Devices		81
4.1	Motivation and Contribution	82
4.1.1	Chapter Overview	85
4.1.2	Notations and Concepts	85
4.2	Wisecr Design	85
4.2.1	Threat Model	85
4.2.2	Wisecr Update Scheme	86
4.2.3	Token Security Requirements and Functional Blocks	91
4.3	On-device Security Function Engineering	92
4.3.1	Immutable Bootloader and Secure Storage	92
4.3.2	Uninterruptible Bootloader Execution	92
4.3.3	Efficient Security Primitives	101
4.3.4	Bootloader Control Flow	102
4.4	End-to-end Implementation and Experiments	105
4.4.1	End-to-end Implementation	105
4.4.2	Bootloader and Memory Management	105
4.4.3	Wisecr Implementation Overheads	108
4.4.4	Experiment Designs and Evaluations	109
4.4.5	Performance Evaluation and Results	109
4.4.6	Case Study	111
4.4.7	Security Analysis	112
4.5	Related Work and Discussion	114
4.5.1	Comparison with Stork (Insecure Method)	116
4.6	Acknowledgement	118
4.7	Chapter Summary	118

Chapter 5. Lightweight Key Generation with MRR Strategies	121
5.1 Motivation and Contribution	122
5.1.1 Chapter Overview	123
5.1.2 Notations and Concepts	124
5.2 Multiple Reference Response-based Reverse Fuzzy Extractor (MR ³ FE)	124
5.2.1 Our Observation	125
5.2.2 RFE-based Key Derivation with MRR Strategy	126
5.2.3 Key Failure Rate	129
5.3 Experimental Validations	130
5.3.1 SRAM PUF Dataset	131
5.3.2 Overhead Evaluations	131
5.3.3 Comparisons	132
5.4 Multiple Reference Response-based Fuzzy Extractor (MR ² FE)	136
5.4.1 MR ² FE	136
5.4.2 BCH Code Decoding	137
5.4.3 Comparison	138
5.5 Related Work and Discussion	139
5.5.1 RFE-based Key Derivation	139
5.5.2 Helper Data Manipulation Attack	140
5.5.3 Entropy Leakage	142
5.5.4 Brute Force Attack Complexity under MRR	143
5.5.5 Generalisabilty of MRR	144
5.5.6 MRR Enrolment	145
5.5.7 Hash and Encoding/Decoding Overhead	146
5.6 Chapter Summary	146
Chapter 6. Noise-Tolerant Secret Keys from Device Memory	149
6.1 Motivation and Contribution	150
6.1.1 Chapter Overview	153
6.1.2 Notations and Concepts	153
6.2 NoisFre Transformation	153

6.2.1	Our Pragmatic Approach	153
6.2.2	Single ℓ_1 -Norm Transformation (S-Norm)	155
6.2.3	Differential ℓ_1 -Norm Transformation (D-Norm)	158
6.3	Formalising Performance Measures	161
6.3.1	Reliability	161
6.3.2	Extraction Efficiency	162
6.3.3	Summary	164
6.4	Experimental Validations	164
6.4.1	Evaluation Approaches	164
6.4.2	Validating Extraction Efficiency and Bit Error Rate	166
6.4.3	Evaluating Uniformity and Uniqueness	168
6.5	Deriving Cryptographic Keys for Security Functions	170
6.5.1	A Method for Realising a NoisFre Key Generator	171
6.5.2	Evaluations	174
6.6	Security Function Implementation for Comparison	177
6.6.1	An Overview	178
6.6.2	Overhead Comparisons	180
6.7	Discussion	183
6.7.1	Generality of NoisFre	183
6.7.2	Provisioning Fingerprints at Run-time	183
6.7.3	Security Analysis	184
6.8	Related Work	186
6.9	Chapter Summary	187

Chapter 7. Noise-Tolerant Key Generators for Highly Limited On-device Memory **189**

7.1	Motivation and Contribution	190
7.1.1	Chapter Overview	191
7.1.2	Notations and Concepts	193
7.2	NoisFre-Lite Concept	193
7.2.1	An overview of the NoisFre-Lite Scheme	194

- 7.3 Transformation and Selection Algorithms 197
 - 7.3.1 Sorted D-Norm Selection with Fixed Distance (Fixed-d) 200
 - 7.3.2 Sorted D-Norm Selection with Variable Distance (Variable-d) 204
 - 7.3.3 Summary 204
- 7.4 Tolerating Errors with Trial-and-error 205
 - 7.4.1 Design of the TRE algorithm 205
- 7.5 Formalising Performance Measures 208
 - 7.5.1 Key Failure Rate 209
 - 7.5.2 Extraction Efficiency 210
 - 7.5.3 Trial-and-error Complexity 212
- 7.6 Experimental Evaluations 213
 - 7.6.1 Evaluation Approaches 214
 - 7.6.2 Key Failure Rate 214
 - 7.6.3 Extraction Efficiency 215
 - 7.6.4 Uniformity (Bias) Evaluation 216
 - 7.6.5 Uniqueness Evaluation 217
 - 7.6.6 Bit-aliasing Evaluation 217
 - 7.6.7 TRE Complexity 218
- 7.7 Key Derivation for Security Functions 221
 - 7.7.1 A Method for Realising a NoisFre-Lite-Based Key Generator 222
 - 7.7.2 On-device Implementation Overhead and Comparisons 224
 - 7.7.3 Security Analysis 227
- 7.8 Chapter Summary 228

Chapter 8. Conclusion and Outlook 231

- 8.1 Summary Preamble 232
- 8.2 Summary of Original Contributions 232
- 8.3 Future Research Opportunities 234
 - 8.3.1 Fingerprinting Emerging COTS Device Memory Types 234
 - 8.3.2 Securing Firmware Broadcasting for Fast Moving Passively Powered Devices 235

8.3.3	Investigating Task Scheduling Mechanisms for Executing Security Functions	236
8.3.4	Investigating The Multiple Reference Concept Beyond Silicon-Based PUFs	236
8.3.5	Developing Theoretical Bounds for Transformation and Selection . . .	237
8.3.6	Investigating TRE Algorithms with Lower Computational Complexity .	237

Appendices 239

Appendix A. Read-Rate Based Dynamic Execution Scheduling 241

A.1	State of Problem	241
A.2	Related work	242
A.3	Background	244
A.3.1	Typical Energy-harvesting Device Architecture	244
A.3.2	Execution Model in HwFixed Scheme	245
A.4	Read-Rate Based Dynamic Execution Scheduling Method	246
A.4.1	Theory and Proof	246
A.4.2	<i>ReaDmE</i> Design	248
A.5	<i>ReaDmE</i> Implementation	249
A.6	Experimental Validation	251
A.6.1	Validating Our Model	252
A.6.2	The Accuracy of Read-rate-based Charging Time Predictions	254
A.6.3	Application Case Study with a Cryptographic Algorithm	255

Appendix B. Memory Bus Snooping and Open Debug Interface Exploits 257

B.1	State of Problem	257
B.2	Related Attacks Against IoT Devices	258
B.3	Threat Model	259
B.3.1	Victim Devices	259
B.3.2	Attacker Capabilities	259
B.3.3	Resources and Costs	260
B.3.4	Attacker Tools	260

- B.4 Exploitable Debug Interfaces and Exposed Memory Buses 261
 - B.4.1 Open Debug Interfaces 261
 - B.4.2 Exposed Memory Buses 263
- B.5 Open JTAG Interface Exploit Case study 263
- B.6 Memory Bus Snooping Case study 268
- B.7 Conclusion and Discussion 271

- Appendix C. Chapter 3 Appendix 273**
 - C.1 EPC Global C1G2v2 Command Encoding 273
 - C.1.1 Experimental Methods 273

- Appendix D. Chapter 4 Appendix 277**
 - D.1 Memory Management Comparison 277
 - D.2 Detailed Wisecr Update Scheme 278
 - D.3 Low Overhead Execution Scheduling 280
 - D.4 Powering Channel State Measurement and Power Aware Execution Model (PAM)280
 - D.5 Transponder Modes & Broadcast Channel 283
 - D.6 Pilot Election Experiments 284
 - D.7 Execution Overhead for Receiving Broadcast Packets 286
 - D.8 Experiment Setup to Access Token’s Internal State 287
 - D.9 Firmware Update to Mobile Tokens 289

- Appendix E. Chapter 5 Appendix 291**

- Appendix F. Chapter 6 Appendix 293**
 - F.1 Derivation of Performance Metric Models 293
 - F.1.1 Synthetic Chip Model 293
 - F.1.2 Unreliability Formalisation of S-Norm Transformation 293
 - F.1.3 Unreliability Formalisation of D-Norm Transformation 297
 - F.1.4 Extraction Efficiency of S-Norm Transformation 298
 - F.1.5 Extraction Efficiency of D-Norm Transformation 300
 - F.2 Remote Attestation 302

Appendix G. Chapter 7 Appendix	305
G.1 Bit-aliasing Pattern	305
Bibliography	307
Index	331
Biography	333

Abstract

The exponential rate of hardware miniaturisation, the emergence of low cost and low power sensing modalities coupled with rapid developments in communication technologies are driving the world towards a future where *tiny scale computing* will be more pervasive and seamlessly integrated with everyday life. This sea of change is driven by the increasing ability of tiny computing platforms to connect people and things to the Internet—the Internet-of-Things (IoT)—enabling transformative applications ranging from connected healthcare to smart cities.

More recently, we have seen the emergence of such tiny scale computing platforms in the form of highly resource-constrained and intermittently powered batteryless devices that rely only on harvested RF (radio frequency) energy for operations, best exemplified by Computational Radio Frequency Identification (CRFID) devices. Despite the simplicity of deployment, reduction in cost and perpetual operational life offered by such devices, provision of security services, offered to typical computing platforms, is significantly more challenging for CRFID devices due to: intermittent powering, unavailability of hardware security support, constrained air interface protocols, lack of secure storage space, limited computational capabilities, and the absence of a supervisory operating system. This dissertation focuses on addressing the challenging problems associated with the provision of security services to resource-limited and intermittently powered devices exemplified by CRFID technologies.

Given the need to update the firmware of such devices, the thesis investigates *how a secure and wireless code update mechanism that is compliant with current communication protocols can be realised under resource constraints and intermittent powering without additional hardware components*. The thesis presents a rigorous design, development and implementation of the first secure wireless firmware update scheme for CRFID devices based on entangling a volatile and hardware instance specific secret key from the on-chip SRAM to the firmware update mechanism. The method, called *SecuCode*, only allows an authorised party to perform a wireless firmware update and does not require any hardware modifications whilst being standards-compliant. The update methods are further developed for simultaneous and secure wireless firmware update of multiple CRFID devices to prevent security threats such as malicious code injection, IP theft, and incomplete code installation whilst complying with standard hardware and protocols. The proposed method, called *Wisecr*, facilitate a secure and scalable method of code update for battery-free passively powered CRFID devices.

Given the lack of secure storage and the prohibitive cost of providing such storage under cost, power and resource constraints, the thesis investigates *exploiting ubiquitously available memory fingerprints for security functions on resource-limited devices*. Device memory fingerprints generated at different time instances are susceptible to unpredictable noise. The state-of-the-art reverse fuzzy extractor (RFE)-based method has been demonstrated to derive usable keys from the inherently unreliable device fingerprints for security functions. However, the computationally-intensive nature of the on-device RFE encoder renders it challenging to employ RFE-based methods on resource-constrained devices. The thesis first proposes a multiple referenced responses (MRR) strategy for device fingerprint enrolment. The proposed approach significantly reduces the on-device implementation overheads for the RFE encoder. The thesis then investigates the transformation of raw memory fingerprints into a noise-tolerant space where the generated device fingerprints are intrinsically highly reliable. The proposed method, *NoisFre*, fundamentally removes the need for an RFE encoder from reliable device fingerprint key derivation methods. The thesis investigates and proposes *NoisFre-Lite* to further improve the extraction efficiency of the noise-tolerant fingerprints to enable mounting *NoisFre* on devices with limited memory sizes. To this end, a highly reliable yet lightweight key generation from ubiquitously available memory fingerprints is achieved for devices with computation and resource limitations to realise the practical use of memory fingerprints for security functions.

HDR Thesis Declaration

I certify that this work contains no material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint award of this degree.

The author acknowledges that copyright of published works contained within the thesis resides with the copyright holder(s) of those works.

I give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library Search and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

I acknowledge the support I have received for my research through the provision of an Australian Government Research Training Program Scholarship.

Signed

February 27, 2023

Date

Acknowledgments

First of all, I would like to express my sincere gratitude to Associate Professor Damith Chinthana Ranasinghe and Doctor Omid Kavehei. This thesis would not be made possible without the support, guidance, encouragement and inspiration from them. In particular, I would like to thank my principal supervisor A/Prof. Ranasinghe. Since my first year in Australia, we have known each other, that was in 2011, 10 years up to now! At every key step I have taken here in Adelaide, I have been in his accompany: I entered the university, I published my first research paper, I got my first paid employment, and I started pursuing a higher degree in education.

I would like to express my appreciation to my colleagues Yansong Gao for fruitful collaborations in the research field and timely support.

I would like to thank my colleagues in the Auto-ID Lab Adelaide: Mr Michael Chesser and Mr Fei Chen for their help on software and hardware support when I was preparing my publications and this thesis.

I would like to thank Associate Professor Alanson Sample for sharing his WISPs and providing insightful feedback to support my research.

My special thanks go to my dear parents, Shiyong Su and Chunyan Wang. I thank them for giving me life, support, encouragement and understanding. Sadly, my dear father Shiyong Su passed away just few days after my thesis examination is announced. I hope in afterlife we will still be a family.

Last but not least, to my beloved fiancée. Although we had quarrels and misunderstandings, you still stayed with me firmly and keep encouraging me to this finishing line. I believe we will have a brighter and happier future.

Thesis Conventions

The following conventions have been adopted in this Thesis:

Typesetting

This document was compiled using pdfLatex ver. 2019. overleaf.com was used as web interfaced editor. Multiple software, including Adobe Illustrator CS2, Foxit PhantomPDF ver.10, Inkscape ver. 0.92 and draw.io were used to produce schematic diagrams and other drawings. 3D modeling was created with vectary.com and Autodesk Inventor ver. 2019.0.2.

Spelling

Australian English spelling conventions have been used, as defined in the Macquarie English Dictionary [1]. Grammary.com is also used check spelling and syntax.

Referencing

The Institute of Electrical and Electronics Engineers (IEEE) reference style [2] is used for referencing and citation in this thesis.

System of Units

The units comply with the international system of units recommended in an Australian Standard: AS ISO 1000-1998 (Standards Australia Committee ME/71, Quantities, Units and Conversions 1998).

Binary prefix in this document follows ISO/IEC 80000-13:2009 [3], e.g., one kilobytes (1 KiB) = $2^{10} = 1024$ bytes.

List of Abbreviations

ACK	Acknowledgement
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
APUF	Arbiter Physical Unclonable Function
ARM	Advanced RISC Machines
BCH	Bose–Chaudhuri–Hocquenghem codes
BER	Bit Error Rate
CCS	Code Composer Studio
COTS	Commercial Off-the-Shelf
CPS	Cyber Physical Systems
CRFID	Computational Radio Frequency Identification
CMAC	Cipher-based Message Authentication Code
CRP	Challenge Response Pair
CVN	Classic Von Neumann (de-biasing)
DB	Database
DC	Direct current
DES	Data Encryption System
DoS	Denial of Service
DRAM	Dynamic Random Access Memory
DRV	Data Retention Voltage
DSO	Digital Storage Oscilloscope
ECC	Error Correction Code
ECDH	Elliptic Curve Diffie-Hellman
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPC	Electronic Product Code

List of Abbreviations

FAA	Federal Aviation Administration
FAR	False Authentication Rate
FDA	(US) Food and Drug Administration
FE	Fuzzy Extractor
FHD	Fractional Hamming Distance
FPGA	Field Programmable Gate Array
FRAM	ferroelectric random-access memory
GCC	GNU Compiler Collection
GNU	Gnu's Not Unix
GMAC	Galois Message Authentication Code
HWAES-CMAC	hardware-accelerated AES CMAC
HD	Hamming Distance
HDM	Helper data manipulation
IC	Integrated Circuit
ID	Identification
IEM	Intermittent execution model
i.i.d.	independently and identically distributed
IoT	Internet of Things
IP	Intellectual Property
ISO	International Organisation for Standardisation
JTAG	Joint Test Action Group
LPM	Low power mode
MAC	Message Authentication Code
MAC	Media Access Control
MCU	Microcontroller Unit
MRR	Multiple Response Reference
NIST	National Institute of Standards and Technology
NVM	Non-Volatile Memory
PAM	Power-aware Execution Model
PUF	Physical Unclonable Function

RF	Radio Frequency
RFE	Reverse fuzzy extractor
RFID	Radio Frequency Identification
RISC	Reduced Instruction Set Computer
RNG	Random Number Generator
RO	Ring Oscillator
RQ	Research Question
SEA	Secure and Efficient Architecture
SoC	System on chip
SPI	Serial Peripheral Interface
SRAM	Static random-access memory
SRR	Single Response Reference
SS	Secure sketch
TI	Texas Instrument
TRE	Trial-and-error
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
UHF	Ultra high frequency
ULP	Ultra low power
US	United States
WISP	Wireless Sensing and Identification Platform
WORM	Write-Once-Read-Many (memory)

Awards and Scholarships

2018

- AMSI Travel funding Authentication for the Future Internet of Things Workshop

2017

- Australian Postgraduate Award (APA) Scholarship for doctoral studies
- IEEE RFID Special Recognition Award for Mega-Challenge Finalist

Publications

Journal Articles

[1] Y. Gao*, Y. Su*, L. Xu, D. C. Ranasinghe “Lightweight (reverse) fuzzy extractor with multiple reference PUF responses” *IEEE Transactions on Information Forensics and Security*, vol. 14, issue. 7, pp. 1887 - 1901, 2018.

[2] Y. Su, Y. Gao, M. Chesser, O. Kavehei, A. P. Sample and D. C. Ranasinghe “Secuocode: Intrinsic PUF entangled secure wireless code dissemination for computational RFID devices” *IEEE Transactions on Dependable and Secure Computing*, vol. 18, issue. 4, pp. 1699 - 1717, 2019.

[3] Y. Su, M. Chesser, Y. Gao, A. P. Sample, D. C. Ranasinghe, “Wisecr: Secure simultaneous code dissemination to many batteryless computational RFID devices” *IEEE Transactions on Dependable and Secure Computing*, Early Access, 2022, <https://dx.doi.org/10.1109/TDSC.2022.3175313> .

[4] Y. Gao*, Y. Su*, S. Nepal, D. C. Ranasinghe. “NoisFre: Noise-tolerant memory fingerprints from commodity devices for security functions.” *IEEE Transactions on Dependable and Secure Computing*, Early Access, 2022, <https://doi.org/10.1109/TDSC.2022.3221928> .

* First Co-Authors in alphabetical order.

Conference Articles

[1] Y. Su, Y. Gao, O. Kavehei, D. C. Ranasinghe, “Hash functions and benchmarks for resource constrained passive devices: A preliminary study” in *Proc. IEEE Int. Conf. Pervasive Computing and Communication Workshops (Percom Workshops)*, Kyoto, Japan, 11–15 March 2019, <http://dx.doi.org/10.1109/PERCOMW.2019.8730835> .

Publications

[2] Y. Gao, **Y. Su**, W. Yang, S. Chen, S. Nepal, D. C. Ranasinghe, “Building secure SRAM PUF key generators on resource constrained devices” in *Proc. IEEE Int. Conf. Pervasive Computing and Communication Workshops (Percom Workshops)*, Kyoto, Japan 11–15 March 2019, <http://dx.doi.org/10.1109/PERCOMW.2019.8730781> .

[3] **Y. Su**, D. C. Ranasinghe, “ReaDmE: Read-rate based dynamic execution scheduling for intermittent RF-powered devices” in *Proc. IEEE Int. Conf. on RFID (IEEE RFID)*, Atlanta, GA, USA, April 27–29, 2021, <https://doi.org/10.1109/RFID52461.2021.9444321> .

[4] **Y. Su**, D. C. Ranasinghe. “Leaving your things unattended is no joke! Memory bus snooping and open debug interface exploits.” in *Proc. IEEE Int. Conf. Pervasive Computing and Communication Workshops (Percom Workshops)*, Pisa, Italy, 21–25 March 2022, <https://doi.org/10.1109/PerComWorkshops53856.2022.9767390> .

Patents

[1] **Y. Su**, Y. Gao, D. C. Ranasinghe, “Device Fingerprinting” Patent No. PCT/AU2021/051024, September 03, 2021.

List of Figures

1.1	CRFID example and typical architecture	3
1.2	IoT market revenue percentage in the US in 2020	4
1.3	Sudden power loss due to brownout	5
1.4	Distribution of available clock cycles and corresponding time before <i>brownout</i> .	6
1.5	The generalised physical unclonable function (PUF) concept.	9
1.6	Thesis structure.	17
<hr/>		
2.1	<i>EPC Gen2</i> system overview	24
2.2	Illustrating the developments in computational RFID technology	25
2.3	RF energy-harvesting architecture	26
2.4	The basic architecture of SRAM.	28
2.5	The SRAM PUF principle	28
2.6	The basic architecture of DRAM	30
2.7	The basic architecture of NAND Flash	31
2.8	(Reverse) fuzzy extractor	35
2.9	Fuzzy extractor concept.	36
2.10	Reverse fuzzy extractor concept.	36
2.11	Evaluated memories	38
<hr/>		
3.1	SecuCode control flow and protocol functional blocks	50
3.2	SecuCode protocol.	52
3.3	SecuCode enrolment process and mask data structure.	56
3.4	CRP-block map.	58
3.5	Intermittent Execution Model (IEM) concept and validation.	60
3.6	SecuCode deployment, in-field operation and memory arrangement.	62
3.7	Bit Error Rate and bias measurement.	66

List of Figures

3.8	PUF key derivation latency and success rate evaluation	69
3.9	MAC latency and success rate evaluation.	70
3.10	SecuCode application case study.	71
3.11	Comparing latency and success rate of SecuCode and non-secure method Wisent.	77
3.12	Comparing the throughput of SecuCode and non-secure method Wisent.	77
3.13	Upcoming chapter sneak peek.	79
<hr/>		
4.1	Wisecr code dissemination scheme to multiple devices.	87
4.2	Impact of four key device tasks on power-loss.	94
4.3	PAM sequence diagram.	95
4.4	Experimental evaluation of the latency and success rate of PAM.	97
4.5	Proposed Pilot-Observer method and pilot-election strategy.	99
4.6	Wisecr experiment setup.	101
4.7	Wisecr bootloader control flow.	103
4.8	MPU segmentation and memory arrangement.	106
4.9	Remote attestation data flow and operating modes.	107
4.10	Comparing Wisecr against SecuCode, with experiment setup.	110
4.11	Attestation execution overhead with respect to different firmware sizes.	111
4.12	Application scenario: CRFID sensor devices embedded in an UAV rotor prototype.	112
4.13	Comparing Wisecr against non-secure method Stock.	117
4.14	Upcoming chapter sneak peek.	119
<hr/>		
5.1	Referenced response under three operating conditions	125
5.2	RFE-based key derivation with MRR enrolment strategy.	127
5.3	Experiment setup. The CRFID transponder is enlarged.	131
5.4	BER when single readout response enrolment strategy is utilised.	133
5.5	BER when majority voting response enrolment strategy is utilised.	133
5.6	BER when pre-selection response enrolment strategy is utilised.	134
5.7	RFE-based key derivation scheme.	139

5.8	Bias of 23 tested SRAM PUFs under five different temperatures.	143
5.9	Upcoming chapter sneak peek.	148

6.1	NoisFre overview	151
6.2	NoisFre transformation concept	155
6.3	NoisFre transformation via S-Norm	156
6.4	The role of the noise tolerance parameter θ in S-Norm-based selection	158
6.5	D-Norm-based NoisFre transform	159
6.6	The role of the noise tolerance parameter θ in D-Norm-based selection	161
6.7	S-Norm BER_F and η_{SNorm} validation using the synthetic chip model	166
6.8	D-Norm BER_F and η_{DNorm} validation on the synthetic chip model	167
6.9	S-Norm and D-Norm validation on the physical memory datasets	168
6.10	Uniqueness evaluation using physical nRF52832 chips	169
6.11	Uniformity evaluation using physical nRF52832 chips	170
6.12	NoisFre secure key enrolment process	172
6.13	On-device NoisFre key generation	173
6.14	Validation of D-Norm-based key failure rate	174
6.15	Extraction efficiency comparison between D-Norm and S-Norm	175
6.16	NoisFre system overview	178
6.17	NoisFre-based remote attestation protocol without WORM	179
6.18	NoisFre-based remote attestation protocol with WORM	181
6.19	Implementation overhead of NoisFre and traditional (R)FE-based key derivation	182
6.20	Upcoming chapter sneak peek.	188

7.1	NoisFre-Lite overview	195
7.2	The problem with the Baseline D-Norm-based transformation and selection procedure.	198
7.3	The basis for the two proposed methods.	199
7.4	Sorted D-Norm with fixed distance and variable distance	201

List of Figures

7.5	Example demonstrating the TRE approach	207
7.6	Evaluating the accuracy of the theoretical distribution of ℓ_1 -Norm	209
7.7	Fixed-d extraction efficiency	210
7.8	Evaluating D-Norm selection under odd/even combinations of n and θ	212
7.9	Comparing the key failure rates of the Fixed-d and Variable-d methods.	215
7.10	Comparing the extraction efficiency of the Baseline, Fixed-d and Variable-d methods.	216
7.11	Uniformity (bias) evaluation	217
7.12	Uniqueness of the Baseline, Fixed-d and Variable-d methods	217
7.13	Evaluation of the TRE mechanism with the Fixed-d method.	220
7.14	Evaluation of the TRE with Variable-d method.	221
7.15	Key derivation protocol based on NoisFre-Lite	222
7.16	The RFE-based lightweight POK derivation in Chapter 3	224
7.17	Comparing the implementation overhead of NoisFre-Lite and SecuCode	225
<hr/>		
A.1	RF energy propagation, harvest and brownout	242
A.2	CRFID operational power cycle and burst operational mode	245
A.3	Comparing CEM, IEM and <i>ReaDmE</i>	249
A.4	The implementation of <i>ReaDmE</i> over the <i>EPC Gen2</i> protocol.	250
A.5	<i>ReaDmE</i> experimental setup	251
A.6	Reader's read-rate and CRFID's internal timing variable	253
A.7	Accuracy of charge time predicted based on read-rate	254
A.8	Evaluating the effectiveness of CEM, IEM and <i>ReaDmE</i>	256
<hr/>		
B.1	JTAG debug interface found in IoT devices	262
B.2	Two-wire debug interfaces found in IoT devices.	262
B.3	Different NVM chips in IoT devices	263
B.4	Tear-down of a Schlage electronic lock FE575 electronic lock	264
B.5	The main board and embedded MSP430 chip of the electronic lock.	265

B.6	JTAG debugger attached to the electronic lock main board.	267
B.7	The memory content read from the electronic lock.	267
B.8	The tear-down of the TP-link Tapo C100 IP camera	268
B.9	Accessing Wi-Fi camera's Flash memory using CH341A programmer.	269
<hr/>		
C.1	SecureComm command encoding	273
C.2	Authenticate command encoding	274
C.3	Experiment setup for SecuCode latency and success rate measurement	275
<hr/>		
D.1	Wisecr protocol implemented over the <i>EPC Gen2</i> protocol.	279
D.2	CRFID charge and discharge time versus operational distance.	282
D.3	Evaluation of the pilot token selection strategies we propose.	285
D.4	Clock cycle measurement of receiving and acknowledging RFID package.	286
D.5	Experiment setup and instrumented CRFID device.	288
D.6	Attempt to update firmware while the UAV rotor is rotating.	289
<hr/>		
<hr/>		
F.1	Unreliability Formalisation of S-Norm Transformation	294
F.2	Unreliability Formalisation of D-Norm Transformation	295
F.3	Validation on D-Norm extraction efficiency	299
F.4	Showing the relationship between P_{block} and Q terms.	301
F.5	Remote attestation memory management and data flow	303
<hr/>		
G.1	Bit-aliasing evaluation for Baseline method	306
G.2	Bit-aliasing evaluation for Fixed-d method	306
G.3	Bit-aliasing evaluation for Variable-d method	306

List of Tables

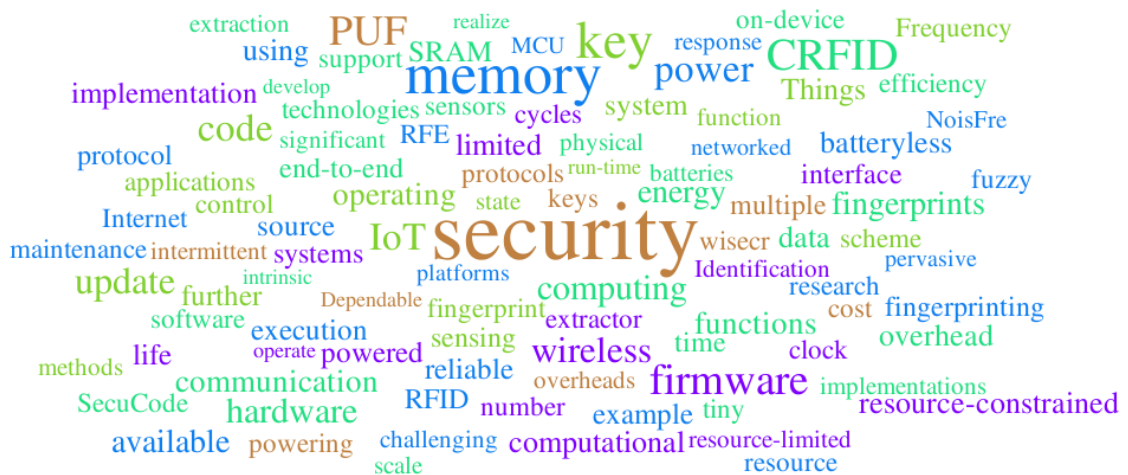
2.1	Overview of memory fingerprinting (or memory PUF) datasets	39
3.1	Table of notations in this chapter.	45
3.2	Random Number Generator Comparison.	53
3.3	MAC evaluation with 153-byte firmware	59
3.4	MAC evaluation with 419-byte firmware	59
3.5	CRP-block efficiency.	66
3.6	Memory & execution load of SecuCode functional blocks.	67
3.7	SecuCode case study performance measures.	72
3.8	Comparison between related works.	74
4.1	Table of Notations in This Chapter	83
4.2	Execution overhead of pilot and observer tokens when receiving broadcast packets.	98
4.3	Wisecr Implementation and Execution Overheads	107
4.4	Comparison with Related Studies.	116
5.1	Table of notations in this chapter.	124
5.2	Key failure rate using single readout, majority voting and pre-selection	135
5.3	Overhead of RFE and FE when SRR, 2MRR and 3MRR are used.	135
5.4	BER evaluated for PUFs over different operating conditions	145
5.5	Comparing the implementation overhead of MRR, SRR and RFE-based methods	147
6.1	Table of notations in this chapter.	154
6.2	Memory Datasets.	163
6.3	The lowest key failure rate P_F^{fail} achievable	177
6.4	Implementation overhead of R(FE) employing BCH codes.	183
7.1	Table of notations used in this chapter.	192
7.2	Bit-aliasing evaluations	218

List of Tables

A.1	Comparing with related works.	244
A.2	Constants value measured for <i>Readme</i>	253
B.1	Basic JTAG signals.	262
B.2	JTAG pins voltage drop	266
B.3	The estimated JT1 pins definitions.	266
D.1	Comparing the settings of tokens.	284
E.1	The implementation overhead of different Hash functions	291
E.2	BCH code encoding overhead.	291
E.3	BCH code decoding overhead.	292

Chapter 1

Introduction



This chapter introduces an emerging and unique class of Internet of Things devices—battery-free, energy-harvesting platforms with computational and sensory capabilities. This thesis focuses on developing and providing security services for such devices. In this context, this thesis investigates two challenging problems: i) how to securely and wirelessly update firmware for resource-constrained devices deployed in insecure environments and ii) how to exploit ubiquitously available memory fingerprints to provide security functions for these low-cost and resource-limited devices. This chapter summarises the ways that the thesis contributes to addressing these two challenging problems before outlining the whole work to conclude the chapter.

‘All we have to decide is what to do with
the time that is given to us.’

*The Lord of the Rings: The Fellowship of
the Ring*
JRR Tolkien

1.1 Introduction

The term ‘Internet of Things’ (IoT) refers to technologies that connect physical ‘things’, such as buildings, vehicles, sensors, actuators and computational nodes, via the Internet to enable person-to-thing and thing-to-thing interactions [4]. Over the last decade, the number of IoT technologies has increased, and the technology has expanded its reach into every corner of our lives by enabling a multitude of applications. Smart card payments and ticketing systems deliver convenience to modern life; networked smoke alarms call for firefighters to save lives; industrial Internet of Things technologies safeguard the operation of machines and processes; injectable blood-alcohol sensors help deter drink-driving to prevent motor vehicle accidents and reduce road fatalities [5].

According to a 2020 International Data Corporation intelligence report, the number of Internet-connected entities, or ‘Things’, will reach 55.7 billion by 2025, with the amount of data generated by these Things expected to reach 73.1 zettabytes [6]. The most promising markets are consumer electronics, transportation, health care, manufacturing and the retail, logistics, agriculture and animal husbandry sectors are also poised for further growth [7]. By 2030, IoT will connect up an estimated 195.7 billion Things globally [8], more than 23 devices per capita.

The maturing of energy-harvesting technology and ultra-low-power computing systems has led to the realisation of battery-less devices, a new class of extremely resource-limited computing platforms that has extended the reach of the IoT. These devices operate entirely on energy extracted from the ambient environment [12]. Recent developments of *Computational Radio Frequency Identification* (CRFID) devices—such as the Wireless Sensing and Identification Platform (WISP) [13], Moo [14] and Farsens Pyros [15]—are examples of resource-limited, intermittently powered and battery-less devices that operate on harvested radio frequency (RF) energy. Such battery-less devices offer several benefits. First, they remove the need for risky maintenance in scenarios that are challenging for traditional battery-powered sensors, as in the case of pacemaker control and implanted blood glucose monitoring [16] and in applications

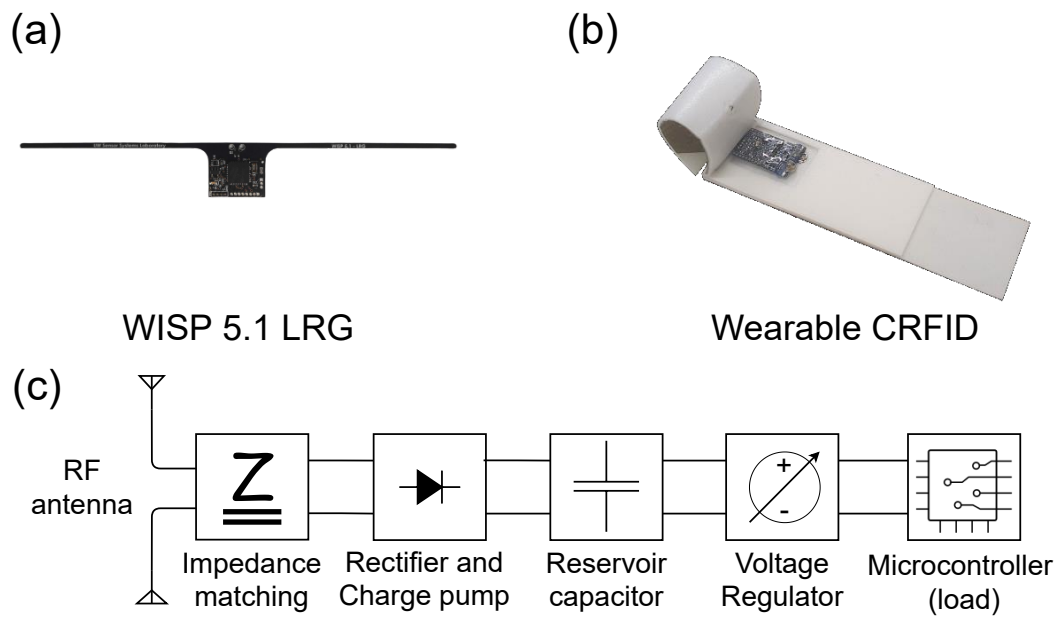


Figure 1.1: Computational Radio Frequency Identification (CRFID) devices, including: (a) the Wireless Sensing and Identification Platform (WISP) 5.1 LRG [9]; (b) a wearable CRFID device with a flexible textile planar inverted-F antenna [10], [11] and (c) the architecture of a typical CRFID device operating on harvested radio frequency (RF) energy.

involving expensive maintenance of batteries, or in which batteries are undesirable, such as wearable devices in healthcare applications [10], [17]–[21]. Second, they reduce the cost of devices. Third, they offer potentially indefinite operational life, attractive for monitoring and maintenance activities in the aviation sector [22]–[25]. The battery-less, low-cost simplicity and perpetual, maintenance-free operational life of these *small-scale* computing platforms represent a compelling proposition for their use as devices in the IoT and cyber-physical systems.

Figure 1.1(a) illustrates a CRFID device exemplified by the popular WISP, first developed by Intel Research, Seattle, United States (US) [26]; (b) shows a CRFID wearable developed for on-body movement monitoring applications [10], [11], [27]. Despite the various CRFID device embodiments, the typical architecture of a device is depicted in (c). The RF antenna is excited by incident electromagnetic waves, starting from the left-hand side, and excites charges into the impedance matching network. The impedance matching technique maximises energy-harvesting efficiency at the operating frequency bands. The rectifier ensures that charges flow in only one direction, producing a direct current (DC). The device can employ an optional charge pump to collect charges, and the generated charges are buffered into an energy-storage component or reservoir—normally a capacitor. The buffered electricity is regulated to the desired voltage before being delivered to the load, such as the microcontroller unit (MCU) of a CRFID device.

1.2 Challenges and Opportunities

The MCU can perform computing tasks including functions that use peripheral sensors and actuators to support applications.

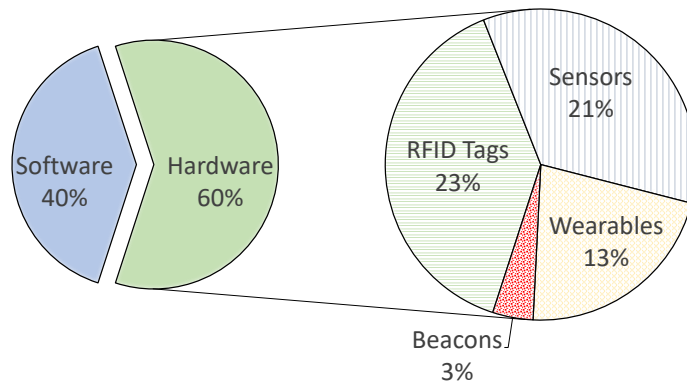


Figure 1.2: Internet of Things market revenue percentage in the US in 2020. The plot was recreated based on the data provided by Grand View Research [28].

Interestingly, according to a 2021 Grand View Research report (summarised in Figure 1.2), traditional RFID technologies, battery-less and operating on harvested RF energy capable of automatic and unique identification, are currently the largest revenue source in the world's biggest IoT market, the United States (US) [28]. More concretely, in the US in 2020, IoT hardware sales produced 60% of overall revenue [28]. Further delineating hardware into RFID tags, sensors, wearables and beacons, RFID tags and sensors represent 23% and 21% of the total revenue. Thus, it can be expected that merging sensing and computation capabilities using Computational RFID devices will conquer a significant portion of the hardware and software IoT market.

1.2 Challenges and Opportunities

'A wise man always seizes the
opportunity to turn it into a better future.'

Armored Warfare

JFC Fuller

Despite the significant potential of low-cost, battery-less devices—and the many attractive properties of these devices—it is critical to avoid deploying systems built on such technologies without considering security. Accordingly, this dissertation concentrates on a broad class of problems related to the provision of security services for the kinds of resource-limited and

intermittently powered devices exemplified by CRFID. This section outlines the challenges and opportunities associated with provision of security services to such devices.

1.2.1 Challenges

Working with low-cost and resource-limited devices is non-trivial, and providing security services presents challenges. This section elaborates on those challenges with reference to CRFID devices, as illustrated in Figure 1.1.

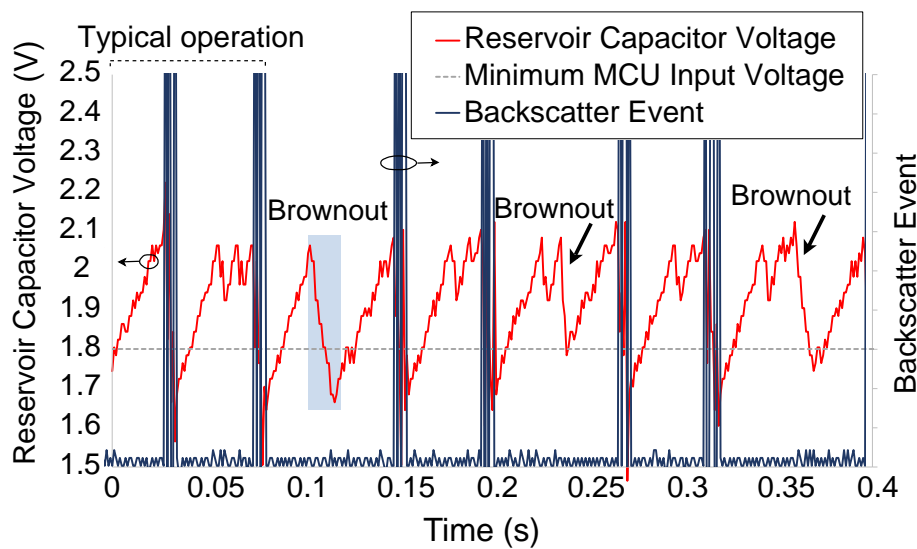


Figure 1.3: An example of sudden state loss when the reservoir capacitor voltage falls below 1.8 V on a CRFID device due to a brownout event. The energy stored in the reservoir capacitor enables the device to complete its operation. A backscatter event indicates a communication event originating with the device where incident radio waves are reflected back with modulated data encoding.

Limited and Indeterminate Powering.

Energy-harvesting systems operate intermittently and only when energy is available from the environment. During operation, a device gradually buffers energy into a storage element (such as a reservoir capacitor, as shown in Figure 1.1). Once sufficient energy has been accumulated, the device begins operation. However, energy depletes more rapidly (i.e., in milliseconds) during operation than during accumulation and charging (i.e., in seconds). Furthermore, energy accumulation during RF energy harvesting is sacrificed by backscatter communication links, with a portion of the incident energy reflected back during communications. Therefore, power failures are common and occur at a millisecond timescale [22], [29], [30], as experimentally validated in Figure 1.3.

1.2 Challenges and Opportunities

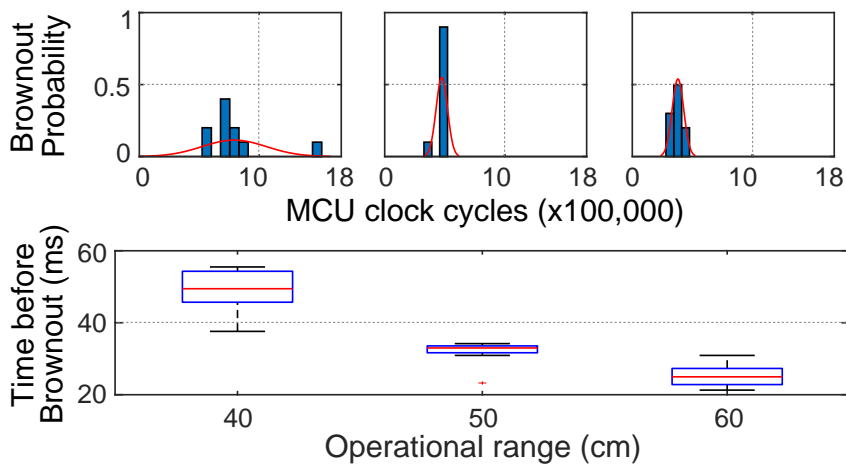


Figure 1.4: Distribution of available clock cycles (top panel) and corresponding time (bottom panel) before *brownout*—exhaustion of available energy under computational load—for the open hardware and open software CRFID WISP 5.1 LRG [9] built with an MSP430 microcontroller. A message authentication code was executed as the computational load, and the device location was increased from 40 to 60 cm above a powering source (an RFID reader antenna). We observed that *limited available MCU clock cycles and operational time periods before state loss for each charging–brownout cycle*.

The timing of power-loss events across devices is ad-hoc, as illustrated in Figure 1.4. For example, because harvested energy varies between different distances, a computation task may be only partially executed before power failure, and it is difficult to predict when this will occur. Furthermore, saving and subsequently retrieving states at code execution checkpoints for a long-running application in an intermittent execution mode [31] is not only costly in terms of computations and energy [32] (saving a state in non-volatile memory [NVM], such as flash or electrically erasable programmable read-only memory [EEPROM], consumes more energy than static random-access memory [SRAM], and reading a state from ferroelectric random-access memory [FRAM] consumes more energy than writing it, as demonstrated in Figure 4.2) but also leaves devices vulnerable to attacks [33] when the checkpoint state is stored on an off-chip memory (due to the lack of internal MCU memory), where non-invasive contact probes can be used to readout values when the memory bus is easily accessible [34], [35]. For example, common memory readout ports, such as the I²C (Inter-Integrated Circuit) bus used by a microcontroller to connect to external devices, are often exposed on the top layer of a printed circuit board (PCB); or debug interfaces are often left open after product rollout to facilitate the extraction of secret keys from non-volatile memory. Appendix. B uses two case studies to demonstrate how such attacks can be easily committed by a person with inexpensive equipment and limited knowledge. These issues make the execution of long-running security algorithms, such as the Elliptic Curve Diffie-Hellman (ECDH) key exchange, difficult to deploy securely¹.

¹We are aware of optimised public key exchange implementations, such as ECDH and Ring-LWE [36], [37]; however, these implementations are impractical for passively powered, resource-constrained devices. For example,

Consequently, power must be managed carefully to avoid power loss and ensure the device is not left in a potentially vulnerable state. Furthermore, computationally efficient security schemes must be sought.

Unavailability of Hardware Security Support.

Because highly resource-constrained devices lack hardware security support, many security features are unavailable, including a trusted execution environment (e.g., Advanced RISC Machines [ARM] TrustZone [39] and memory dedicated to explicitly maintaining the secrecy of long-term secret keys, as in [40]).

Constrained Air Interface Protocols.

In the context of CRFID devices that communicate over the ultra-high frequency (UHF) industrial, scientific and medical band, the widely used wireless protocol for long-range RFID communications only provides *insecure unicast communication links* [41]. For example, the ISO-18000-6C protocol supports maximum data rates of 640 kbps for RFID tag to reader and 128 kbps for reader to RFID tag, given equiprobable ones and zeros in the data stream. Importantly, the air interface does not support broadcast features or device-to-device communication in mesh networking, which is typical of wireless sensor networks [22].

Unavailability of Supervisory Control from an Operating System.

Unlike *wireless sensor network nodes*, severely resource-limited systems, such as CRFID devices, do not operate under the supervisory control of an operating system to provide support for secure executions or security services. Firmware for such devices has been described as *monolithic*.

Lack of Secure Storage Space.

Security mechanisms rely on the secure storage of keys. Traditional key-storage methods in NVM are vulnerable to various invasive and non-invasive attacks. Non-invasive attacks on the readout of keys from exposed ports between off-chip Flash memory or EEPROM memory units and MCUs have been demonstrated in both [34] and this thesis's study, Appendix B. It is

ECDH [36] with Curve25519 requires 4.88 million clock cycles on a MSP430 MCU. In contrast, an extremely limited number of clock cycles are available from harvested energy before energy depletion [29], [38] (e.g., 400,000 clock cycles are expected even at a proximity of 50 cm from an energy source; see Figure 1.4). There is limited time available for computations in which a CRFID must reply before the breach of strict air interface protocol time-outs.

1.2 Challenges and Opportunities

difficult to protect keys from invasive attacks without using costly methods, such as protective coatings [42] and active tamper-sensing circuitry requiring batteries. Low-end devices usually lack a secure NVM to maintain key security. Therefore, providing a secure key-storage method without an additional implementation overhead, hardware modifications or additional costs to the device represents a challenge.

1.2.2 Opportunity 1: Secure Wireless Firmware Update

A shocking 2016 Cable News Network report broadcasted a market-research company's claim that products made by implantable cardiac devices manufacturer St. Jude Medical were hackable [43]. Implantable cardiac devices, such as pacemakers and defibrillators, are important life-saving medical devices that help heart disease patients live a normal life. These devices are connected to the Internet via a base station installed in the user's home. Physicians can remotely monitor patient health and make adjustments to the therapeutic device accordingly. Although St. Jude Medical denied the allegations that its products were hackable, a report issued by the US Food and Drug Administration (FDA) confirmed the claim, holding that the vulnerabilities could allow a hacker to wirelessly log in to devices, allowing them to conduct malicious attacks, such as depleting the device's batteries or introducing incorrect pacing or shocks. Fortunately, no patient fitted with a St. Jude Medical device was harmed, with a software patch soon installed via a remote system update to resolve the vulnerabilities. The FDA subsequently advised that patients could continue using the devices.

Despite the wide range of possibilities and various manifestations of the resource-limited-device concept, including the St. Jude Medical pacemakers and examples of CRFID iterations, existing realisations feature a relatively similar basic architecture: MCUs, transceivers, sensors, batteries, power harvesters and/or actuators are combined with important components of the *application-specific software* or, more simply, the 'code' that enables the Things to communicate with other parties and fulfil interactive tasks [44]–[46]. Consequently, as exemplified by the St. Jude Medical story, updating and patching firmware is necessary and inevitable. Without standard protocols or system-level support, firmware is typically updated using a wired programming interface [9], [14]. In practical applications, wired interfaces are often inaccessible without significant effort, as in the case of the pacemaker, presenting a potential attack vector for tampering with the device behaviour [47]. Furthermore, compromised devices can be hijacked to attack other networked entities [48]. Disabling a wired interface after the initial programming phase, when the device is still at the manufacturer, can prevent further access. However, doing so makes the wireless update option the only approach to altering

the firmware or repurposing the device post-manufacture. Therefore, this thesis considers the development of mechanisms for securely and wirelessly updating code for CRFID devices that can address the challenges faced by resource-limited and intermittently powered devices discussed in Section 1.2.1 while complying with current communication protocols, ultimately enabling their practical implementation.

1.2.3 Opportunity 2: Security Functions from Device Memory Fingerprints

Fingerprinting pervasively available embedded memory (including SRAM [49]–[51], dynamic random-access memory (DRAM) [52], Flash memory [53], [54] and EEPROM) in commercial off-the-shelf (COTS) devices is highly desirable for providing security functions, particularly in the absence of cryptographic modules for key security. This is because: i) memory cells are intrinsic to computing platforms and available in large volumes to obtain many independent fingerprints or secret keys; ii) memory biometrics are a physical source of true randomness; iii) using memory biometrics remove the need for a protected NVM for secret keys (root keys can be generated on-demand and ‘forgotten’ after usage); and iv) implementing memory biometrics imparts no extra hardware cost to *existing COTS* devices.

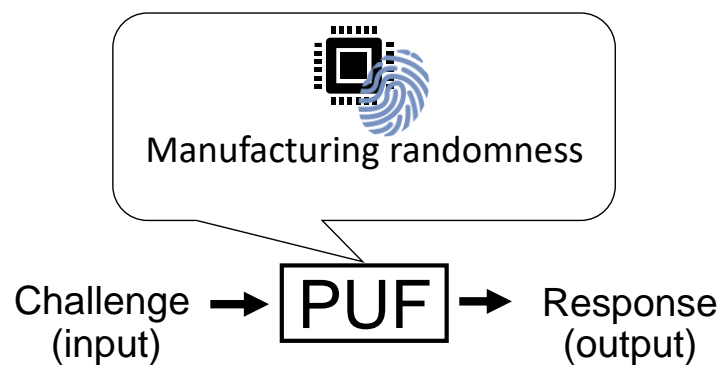


Figure 1.5: The generalised physical unclonable function (PUF) concept. A PUF takes a challenge (input) and uniquely maps it to a response (output), depending on the manufacturing randomness of the hardware in which the PUF resides.

Hardware fingerprinting is closely related to the notion of a physical unclonable function (PUF) [55]–[57]. As Figure 1.5 shows, a PUF reacts with an instance-specific response (output) by exploiting manufacturing randomness when queried by a challenge (input) [58], [59]. Commodity memory fingerprinting techniques are not new, including SRAM PUF [60]–[62], Flash PUF [53] and the so-called memory PUFs [63]–[65]. However, despite the many attractive features of using intrinsic memory to incorporate security functions built upon memory

1.2 Challenges and Opportunities

fingerprints into low-end resource-constrained devices, in practice, deriving usable keys for security functions remains a challenge [38], [62], [66].

A fingerprint (or PUF response) generated from a given device should always be exact for security functions. However, fingerprints generated at different time instances are susceptible to unpredictable noise, including thermal noise, supply-voltage fluctuations and device ageing. Consequently, fingerprints differ in various ways [64], [67]–[69]. For example, the positions of flipped raw bits vary, and the exact number of flipped raw bits varies over time. Thus, it is a challenge to reliably determine raw bits, and existing memory fingerprinting schemes may not naturally tolerate noise in the raw noisy fingerprint space. Hence, it is typically infeasible to directly employ on-demand device-generated fingerprints (or PUF responses) for security functions.

Until now, using approximate and noisy renditions of biometric fingerprint templates in security functions has been demonstrated using fuzzy extractors [70], [71], a method that relies on post-error correction using generated helper data from a reference fingerprint to reconcile bits errors². However, employing a fuzzy extractor on-device engenders two fundamental problems. First, the fuzzy extractor logic responsible for mitigating errors introduces high on-device computational overhead [72]. Second, the associated helper data, assumed to be publicly known, can be actively manipulated in helper data manipulation (HDM) attacks to weaken or compromise the security of the derived fingerprint or secret key [73], [74]. Creating a generic countermeasure to HDM attacks remains an ongoing challenge [74].

Therefore, this thesis investigates the significant leap from the desire to repurpose ubiquitously available memory on devices for security functions and the practicability of exploiting memory fingerprints for security, especially given this approach removes the need for a protected NVM for secret keys.

²This concept is elaborated upon in Section 2.3.3 in Chapter 2.

1.3 Research Questions

‘If you are not moving closer to what you want, you probably aren’t doing enough asking.’

Chicken Soup for the Soul

Jack Canfield

Capitalising on the opportunities discussed, this thesis considers the following two research questions:

RQ 1. *How can we realise a secure and wireless code update mechanism that is compliant with current communication protocols under resource constraints and intermittent powering without additional hardware components?*

RQ 2. *How can we address the problems associated with exploiting ubiquitously available memory fingerprints for security functions on resource constrained devices?*

1.4 Summary of Original Contributions

‘Once you make a decision, the universe conspires to make it happen.’

Ralph W. Emerson

In addressing the proposed research questions, this dissertation makes several original contributions to the broader field of computer security and the specific area of developing security functions for resource-limited computing platforms and memory fingerprinting:

1. To address the challenges posed and meet the demands of a secure wireless firmware update method for resource-constrained and intermittently powered CRFID devices, we present SecuCode. It is evident that building hardware security primitives with on-device memory fingerprints or memory-based PUFs is a compelling proposition given the ubiquity of memory in electronic devices, especially for the types of low-end devices for which cryptographic modules are often unavailable. Hence, we first address the challenges associated with using on-chip SRAM fingerprints to realise lightweight, reliable, secure key generation for a resource-constrained device. Second, we demonstrate an end-to-end

1.4 Summary of Original Contributions

design that extends from firmware compilation to a successful firmware update process supported by an intermittent execution model for task scheduling on a CRFID device, managing the transient nature of power availability. Third, we develop a tool (the SecuCode App) to update firmware and conduct the complete end-to-end implementation and evaluation on resource-constrained and intermittently powered CRFID devices. SecuCode is fully compatible with the existing communication protocols employed by CRFID devices, particularly the ISO-18000-6C protocol, and it is built on a standard and industry-compliant firmware compilation and update method realised by extending a recent Texas Instruments framework for firmware updates. SecuCode is the first *secure* firmware dissemination scheme for a battery-less CRFID devices. This work addresses **RQ 1** and **RQ 2** and was published in the journal *IEEE Transactions on Dependable and Secure Computing (TDSC)* under the title ‘SecuCode: Intrinsic PUF Entangled Secure Wireless Code Dissemination for Computational RFID Devices’ [38]. A demonstration video of the end-to-end implementation in an example application scenario can be viewed here:

<https://www.youtube.com/watch?v=nWcwGLsjJK0>



scan to watch

2. We introduce Wisecr as a means of solving the limited efficiency associated with the one-to-one firmware update scheme and the lack of intellectual property (IP) protection provided by SecuCode’s wireless channel. Wisecr performs three security functions for secure and fast updates, preventing malicious code injection attacks, preventing IP theft and attesting to code installation. Wisecr achieves *rapid* updates by supporting simultaneous updates to multiple CRFID devices via the secure broadcasting of firmware over a *standard non-secure unicast air interface protocol*. Wisecr represents the first *secure, simultaneous (fast)* firmware dissemination scheme for multiple battery-less CRFID devices. In the first step in the development process, we build an efficient broadcast session key exchange method. Second, to avoid power loss, and therefore, achieve uninterrupted execution of a firmware update session, we propose the power-aware execution model (PAM). The task scheduling method embodied by PAM provides adaptive control of the execution model on devices using RF powering channel state information collected and reported by field-deployed devices while also reducing disruptions to broadcast data synchronisation across multiple devices via the introduction of the concept of a *pilot* tag selection from participating devices in the update scheme to drive the protocol. These methods negate the need for costly, secure checkpointing

methods and avoid scenarios in which a device is left in a vulnerable state during power loss. Third, we extensively evaluate Wisecr by comparing it with current non-secure methods and SecuCode as well as validating our scheme via an end-to-end implementation and demonstration. This research addresses **RQ 1** and was published in the journal *IEEE Transactions on Dependable and Secure Computing (TDSC)* under the title ‘Wisecr: Secure Simultaneous Code Dissemination to Many Batteryless Computational RFID Devices’ [75]. A demonstration video of the end-to-end implementation in an example application scenario can be viewed here:

<https://www.youtube.com/watch?v=GgDHPJi3A5U>



scan to watch

3. Although it was possible to employ memory fingerprinting methods or a PUF-based key generator in SecuCode, doing so in the Wisecr broadcast presented a considerable challenge due to the overhead associated with even the lightweight key generation method developed in SecuCode. Interestingly, all extant PUF key generators and memory-fingerprinting-based key generation methods enrol only one response evaluated in the nominal operating corner (e.g., room temperature) at a server. In proposing and exploring the multiple reference responses (MRR) concept, we show that enrolling multiple responses at a server in discrete operating corners engenders an exponential reduction in the overhead of the reverse fuzzy extractor (RFE) implementation at the device, helping to further reduce the overhead of SecuCode’s lightweight key generator. This involves first leveraging MRR enrolled under discrete operating conditions for key generation. As an immediate application, we propose a lightweight mutual-authentication protocol based on such an RFE called M^3RFE . We subsequently analyse the key failure rate of M^3RFE . Second, we demonstrate a significantly reduced implementation overhead using M^3RFE via experiments employing software implementations that target a resource-constrained CRFID device with an embedded SRAM PUF. Third, to examine the generalisability of MRR, we experimentally showcase its applicability to a fuzzy extractor and validate the substantially reduced implementation overheads. This work addresses **RQ 2** and was published in the journal *IEEE Transactions on Information Forensics and Security (TIFS)* under the title ‘Lightweight (Reverse) Fuzzy Extractor with Multiple Referenced PUF Responses’ [76].
4. Using fingerprints in security functions is challenged by small-but-unpredictable variations in fingerprint reproductions from the same device due to measurement noise. SecuCode’s lightweight key generation method and the proposed MRR concept

1.4 Summary of Original Contributions

represent a significant step towards addressing the challenges in the context of reverse-fuzzy-extractor-based key generation methods for resource-constrained devices. However, we challenge current thinking by considering alternative methods to potentially avoid the significant overhead attached to a reverse-fuzzy-extractor-based method on a resource-constrained device and the security issues associated with such extractors whilst seeking a *pragmatic* approach for obtaining highly reliable fingerprints from device memories. We investigate the transfer of raw fingerprints to a noise-tolerant space where fingerprint generation is intrinsically highly reliable and propose two methods for doing so, S-Norm and D-Norm. Subsequently, we derive formal performance bounds for both methods to facilitate the implementation of the methods by practitioners. We demonstrate the expressive power of our formalisation by using it to investigate the practicability of extracting noise-tolerant fingerprints from commodity devices. Alongside extensive simulations, we employ 119 chips from five different manufacturers for extensive experimental validations. Our results, including an end-to-end implementation demonstration with a low-cost wearable Bluetooth inertial sensor capable of *on-demand and run-time key generation*, show that key using noise-tolerant fingerprint based generators with failure rates below 10^{-6} can be obtained with significantly more efficiency than RFE methods together with ease-of-enrolment supported by the use of a single fingerprint snapshot. This work addresses **RQ 2** and was published in the journal *IEEE Transactions on Dependable and Secure Computing (TDSC)* under the title ‘NoisFre: Noise-Tolerant Memory Fingerprints from Commodity Devices for Security Functions’ [77]. A demonstration video of the end-to-end implementation in the example application scenario can be viewed here:

<https://www.youtube.com/watch?v=O5NWZw-swpw>



scan to watch

5. NoisFre assumes that memory is freely available and abundant. Because this may not be the case for highly resource-constrained devices, such as CRFID devices with limited SRAM memory, we propose NoisFre-Lite, which can provide a root key on commodity electronic devices even when computational and memory resources are highly-constrained. More concretely, given the critical challenge associated with employing a NoisFre key generator on devices with limited memory space is the undesirably low extraction efficiency (the number of noise-tolerant bits extracted versus available intrinsic memory size), NoisFre-Lite addresses the need to derive more key bits from SRAM memory with *highly constrained resources (in terms of both memory*

capacity and computational power). Similar to NoisFre, NoisFre-Lite transforms raw SRAM fingerprints into noise-tolerant fingerprints. However, it develops two new algorithms for transformation and noise-tolerant bit selection that are built around a trade-off between reliability and extraction efficiency. Consequently, some noise-tolerant bits can remain below the level of reliability needed for a highly reliable key generator. Therefore, we devise a trial-and-error (TRE) method to optimise key reliability by using the server’s computational resources without imposing any overhead on the highly resource-constrained device. We formulate analytical models for the proposed key extraction method to evaluate their performance to help practitioners accurately estimate the best key size and reliability given a specific memory chip, and the complexity of the on-server TRE function. We extensively evaluate NoisFre-Lite in terms of key reliability, extraction efficiency, bias, uniqueness and bit-aliasing. The experiments affirm the validity of our formulations and the efficacy of our approach. We implement a NoisFre-Lite key generator on a battery-less CRFID with a highly constrained 2 KiB SRAM memory. A 128-bit key with a key failure rate of $< 10^{-6}$ is provided by the NoisFre-Lite method using as few as 20,238 clock cycles for key generation; this is a 92.8% reduction in computational overhead compared to the state-of-the-art error correction-based key derivation using a reverse fuzzy extractor (needing 280,063 clock cycles). Furthermore, the approach negates the security concerns associated with key generation using a reverse fuzzy extractor. This contribution addresses **RQ 2**.

6. Based on the extensive systems research conducted, this thesis contributes three open-source code releases:

- The source code for the complete end-to-end system for SecuCode [38] in Chapter 3: A secure wireless firmware update to a single device is available at <https://github.com/AdelaideAuto-IDLab/SecuCode>

Acknowledgement. Michael Chesser contributed to the development of the SecuCode App (the desktop application) (SecuCode/src/SecuCodeApp) and the immutable on-device SecuCode bootloader (SecuCode/src/wisp5-bootloader).

- The source code for the complete end-to-end system for Wisecr [75] in Chapter 4: A secure wireless firmware update for multiple devices with IP protection over the insecure channel is available at <https://github.com/AdelaideAuto-IDLab/Wisecr>

Acknowledgement. Michael Chesser contributed to the development of the Wisecr App (the desktop application) (Wisecr/src/SecuCodeApp), the immutable

1.5 Thesis Structure

on-device SecuCode bootloader (Wisecr/src/wisp5-bootloader) and implementing the broadcast method over the RFID air interface through the Wisecr App (Wisecr/src/wisp5-base/CCS/wisp-base/RFID).

- The source code for the complete end-to-end system demonstration for NoisFre [77] in Chapter 6: A reliable memory fingerprinting-based remote attestation method is available at <https://github.com/AdelaideAuto-IDLab/NoisFre>

This thesis also contributes two extensive datasets . These have been publicly released to support research on memory fingerprinting and PUFs:

- A dataset of 2 KiB internal SRAM from 20 MSP430 chips measured at 5 temperature corners (-15 °C, 0 °C, 25 °C, 40 °C and 80 °C) is available at <https://dx.doi.org/10.21227/H27T0S>
- A dataset of 64 KiB internal SRAM from 12 nRF52 chips measured at 3 temperature corners (-15 °C , 25 °C and 80 °C) is available at <https://dx.doi.org/10.21227/ktc9-x515>

1.5 Thesis Structure

This thesis is organised into eight chapters. The structure is summarised in Figure 1.6 and described below.

Chapter 1 introduces resource-constrained devices (with an emphasis on CRFID devices) and device fingerprinting (or PUFs). It also discusses the challenges and opportunities associated with providing security services to these devices and facilitating their implementation and provides an overview of the thesis’s contributions before concluding with an outline of the structure of the work.

Chapter 2 defines notations and conventions before outlining the fundamentals of CRFID devices and memory fingerprinting (or PUFs), and state-of-the-art key generation methods based on memory fingerprints before detailing the datasets used.

Chapter 3 studies the problem of developing mechanisms for securely and wirelessly updating code that comply with current communication protocols under resource constraints and intermittent powering. We propose SecuCode, a first attempt at securing wireless firmware updates for a single CRFID device, disseminating code to a single device at a time. A lightweight and secure RFE-based device key-generation method is developed that uses on-chip

Introduction	<i>Chapter 1</i>	<ul style="list-style-type: none"> • Introduction to resource constrained devices • Challenges and opportunities • Summary of original contributions • Dissertation structure
	<i>Chapter 2</i>	<ul style="list-style-type: none"> • Notations and conventions • Fundamentals of CRFID technology • Fundamentals of memory fingerprints or memory PUF-based key generators • Introduction to the datasets contributed and used in the thesis
Securing Wireless CRFID Firmware Updates (RQ 1)	<i>Chapter 3</i>	<ul style="list-style-type: none"> • A secure wireless firmware dissemination scheme to a single CRFID device • A lightweight physically obfuscated key derivation method using on-chip SRAM and reverse fuzzy extractor (RFE) • Intermittent execution model (IEM), programmatically encoded at device provisioning time to support the long-run execution of tasks under intermittent powering conditions • Demonstration of an end-to-end design from firmware compilation to a successful firmware update process
	<i>Chapter 4</i>	<ul style="list-style-type: none"> • A secure and simultaneous (fast) wireless firmware dissemination scheme to multiple CRFID devices • A power-aware execution model (PAM) capable of dynamically adjusting the execution model at run-time based on available power • Reducing disruptions to broadcast data synchronisation across multiple devices via the introduction of the concept of a Pilot tag selection from participating devices in the update scheme to drive the protocol • Remote attestation methods to enable the verification of code installations • A holistic design trajectory, from a formal secure scheme design to an end-to-end implementation requiring only limited on-device resources
Reliable Device Key Generation (RQ 2)	<i>Chapter 5</i>	<ul style="list-style-type: none"> • A multiple referenced response (MRR) enrolment strategy for reducing the implementation overhead of the RFE • M³RFE, a lightweight physically obfuscated key derivation protocol based on the combination of MRR and RFE
	<i>Chapter 6</i>	<ul style="list-style-type: none"> • NoisFre approach for achieving highly reliable fingerprints from resource-constrained device's memory without the need for a RFE • Two methods are proposed and investigated for extracting highly reliable fingerprints: S-Norm and D-Norm • Analytical models are developed to predict bounds for unreliability and highly reliable fingerprint extraction efficiency • Develop, implement, evaluate and compare NoisFre-based key generators • Demonstrate an end-to-end design and implementation of on-demand (run-time) generation of a highly reliable key for a remote attestation security function
	<i>Chapter 7</i>	<ul style="list-style-type: none"> • NoisFre-Lite approach for achieving highly reliable fingerprints from highly resource-constrained devices with limited memory capacity and computational power • Two methods that trade reliability for extraction efficiency are proposed and investigated for extracting highly reliable fingerprints • A trial-and-error (TRE) method is proposed for resolving any potential errors in the extracted highly reliable fingerprint at the server (this method benefits from the server's computational power without introducing additional overheads on the highly resource-constrained device) • Analytical models are developed to predict extraction efficiency and key failure rate of highly reliable fingerprints from the proposed methods, and the complexity of the on-server TRE function
Conclusion	<i>Chapter 8</i>	<ul style="list-style-type: none"> • Conclusions • Future research directions

Figure 1.6: Thesis structure.

SRAM fingerprints (or an SRAM PUF). We propose the intermittent execution model (IEM), encoded at the device provisioning time, to support the long-running execution of tasks under intermittent power. Additionally, we demonstrate an end-to-end design that extends from firmware compilation to a successful firmware update process to illustrate the practicability of implementing SecuCode for an extremely resource-limited CRFID device.

Chapter 4 considers four unresolved issues associated with SecuCode. First, SecuCode only disseminates code to a single device at a time, largely limiting the efficiency of firmware updates in cases involving a large number of devices. Second, SecuCode does not address privacy or IP protection goals. Firmware is sent as plaintext over the wireless channel, making eavesdropping possible for any attacker equipped with a simple RF sniffer. Third, in SecuCode, the server is notified by a simple acknowledgement (ACK) response; however, an attacker could easily spoof this command to cause incomplete firmware installation. It is important to provide a mechanism for the server to verify that the firmware update has been performed correctly. Fourth, the IEM developed in Chapter 3 uses programmatically encoded intermittent operating settings for the fuzzy extractor (helper data generator) and message authentication code functions. Because these settings are determined when the bootloader is provisioned, there is no flexibility to dynamically adjust the execution model based on available power at run-time. Wisecr, the first secure and simultaneous (fast) firmware dissemination scheme addresses these problems. Notably, Wisecr includes PAM, a task scheduling model that provides the flexibility to dynamically adjust the execution model based on available power at run-time, and an attestation function that facilitates the verification of firmware installation. The chapter describes a holistic design trajectory that builds from a formal secure scheme design to an end-to-end implementation for resource-limited devices. Wisecr no longer uses an SRAM PUF (or memory fingerprints) due to the outstanding key reliability issues, the substantial overheads imposed by even the lightweight RFE-based key generator and the cumbersome PUF-enrolment procedures. Although, these omissions challenge the feasibility of scaling the implementation of PUF-based device fingerprinting, they are addressed in subsequent chapters.

Chapter 5 examines the problem of the computationally expensive and challenging nature of implementing an RFE-based key-generation method using memory fingerprints in the context of resource-limited and intermittently powered devices. This chapter proposes the adoption of MRR. The MRR methodology of enrolling responses at the server reduces the burden on the on-device RFE function by allowing the server to reconcile a larger number of errors with minimal compromise on the level of security afforded by the key generator. However, the MRR-based method still relies on the underlying RFE to generate reliable keys, and the cost of implementing the RFE generator for resource-limited devices inhibits practical applications (such as Wisecr), especially if the aim is to achieve a widely accepted industry standard operating temperature range (over $-15\text{ }^{\circ}\text{C}$ to $80\text{ }^{\circ}\text{C}$) and industrial-standard key reliability (failure rate less than 10^{-6}).

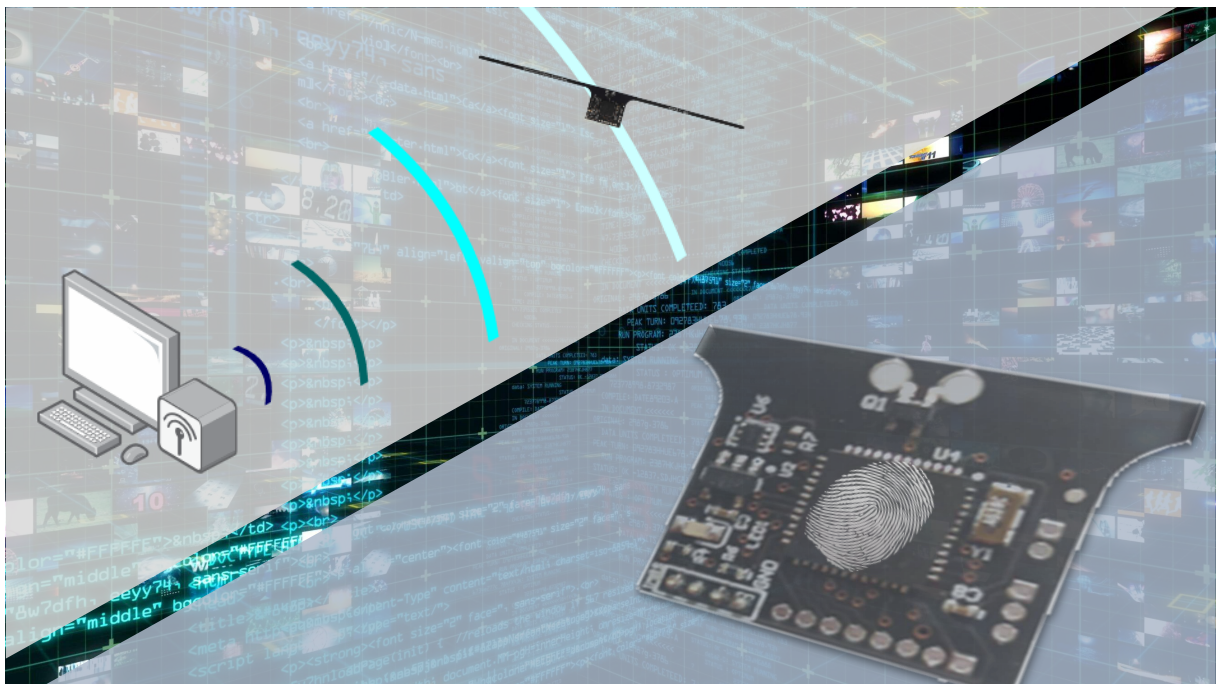
Chapter 6 investigates an alternative to the traditional RFE-based method by contemplating avoiding employing an RFE completely. The chapter proposes NoisFre, a method for achieving highly reliable fingerprints from commodity device memory. NoisFre demonstrably obviates the need for an RFE, circumventing the vulnerability of helper data manipulation attacks (a known and open problem with using fuzzy-extractor-based methods) and significantly reduces implementation and execution overheads. NoisFre requires a single readout of memory fingerprints during the enrolment phase and can achieve industrial standard key reliability, making implementation feasible. However, NoisFre's extraction efficiency (how many noise-tolerant bits can be extracted from a given memory space) is too low for use by extremely resource-constrained devices with very limited intrinsic memory space, such as a chip-embedded system with only a few KiB of memory.

Chapter 7 addresses NoisFre's extraction efficiency problem, proposing two fingerprint transformation and selection algorithms to trade reliability for improved extraction efficiency. A trial-and-error method is developed to resolve the remaining errors in the generated fingerprint key at the server rather than the device, exploiting the server's computational power without introducing additional overheads.

Chapter 8 summarises the research conducted and discusses future research avenues.

Chapter 2

Fundamentals of CRFID and Memory PUFs



This chapter defines the notations and conventions used in this thesis (Section 2.1) before introducing the fundamentals of CRFID (Section 2.2) and memory PUFs, or memory fingerprints (Section 2.3). Section 2.2 provides a brief summary of Electronic Product Code (EPC)-enabled UHF RFID technology, the development of CRFID devices and RF energy harvesting. Section 2.3 outlines common memory PUFs, evaluation measures for memory PUFs, state-of-the-art methods used for reliable key generation with memory PUFs, and the datasets used in this work, including those produced by this research and those publicly available.

2.1 Notations and Conventions

This section defines the notations and symbols used in this thesis, following the approach used in [62].

Scalar. An italic lowercase name is used for a scalar, for example, the index i .

Vector. A bold lowercase name is used for a vector, for example, the PUF response \mathbf{r} .

Collection. A bold all-uppercase name is used for a collection, for example, the database **DB**.

Set. A blackboard uppercase symbol is used for a set, for example, the whole number set \mathbb{Z} .

Cardinality. Two vertical bars enclosing a vector variable represent the cardinality (dimension) of the variable, for example, the size of memory $|\mathbf{mem}|$.

Quantity. An italic upper case name is used for a physics- or mathematics-based quantity, for example, capacitance C and entropy H .

Function. A sans-serif font function name followed by a bracket is used for a function, for example, `PUF()`.

EPC Gen2 commands. A typewriter font is used to represent an *EPC Gen2* command, for example, the `Query` command.

2.2 Battery-less Edge Computing: CRFID Devices

We mainly use CRFID devices to exemplify ultra-low-power energy-harvesting computing devices. As explained in Section 1.1, CRFID devices are lightweight and require less maintenance than traditional battery-powered platforms, anticipating more flexibility in some special applications, particularly in deeply embedded cases. This section outlines the basic concepts of Radio Frequency Identification (RFID) technology before discussing developments in CFRID technology. Interested readers can find more information about other RFID technologies, networked RFID systems and applications in [78].

2.2.1 RFID Preliminaries

Passive or *battery-less* RFID technology combines wireless transmission and reception of RF energy with automatically and uniquely identifying objects [79]. RFID has been widely adopted in commercial and industrial applications for logistics and inventory control, as well as to complement and even replace barcodes, which have several limitations, including line-of-sight access, rigorous alignment requirements (for readability) and propensity for

physical damage [80]. This ubiquity is partly attributed to the establishment of standards such as the ISO/IEC 18000-63:2015 air interface protocol and the Electronic Product Code (EPC) identification number format.

The widespread adoption of the *EPC Gen2* standards and diffusion of *EPC Gen2*-related RFID technology have made RFID infrastructure available globally [81]. The Ultra High Frequency (UHF) band facilitates longer-range communication than other RFID technologies [82], benefiting battery-less sensing applications [17], [83]–[88]. Therefore, developing solutions based on *EPC Gen2*-compliant RFID technologies represents a natural path forward [29]. Hence, this thesis mainly considers *EPC Gen2*-compliant CRFID devices that operate on the license-free UHF Industrial Scientific and Medical bands located between 860 MHz and 930 MHz [80]. Although considering the characteristics of different RFID technologies is beyond the scope of this dissertation, such discussion, including comparisons between different technologies, can be found in [80].

As shown in Figure 2.1(a), a typical *EPC Gen2* RFID system comprises four components: a Host computer, an RFID reader, RFID reader antenna(s) and RFID tag(s). The Host computer communicates to the RFID reader via a local area network (LAN) using the Low Level Reader Protocol (LLRP). A coaxial RF cable connects the RFID reader to the RFID reader antenna. The reader antenna wirelessly transmits power and downlink data modulated in amplitude shift keying (ASK) to the RFID tag(s), and the RFID tag backscatters³ uplink data modulated in ASK or phase-shift keying (PSK), as defined in the *EPC Gen2* air-interface protocol [90].

2.2.2 RFID Protocol Overview

A typical RFID system uses the following steps to communicate with an RFID tag (or, in general, any transponder compliant with the *EPC Gen2* protocol, including CRFID devices):

- **Host to Reader.** An application on the Host machine constructs LLRP commands to build an `ROSpec` and `AccessSpec` to control the reader and transmits these specifications to a reader. In some documents, the Host is a client, and the RFID reader is a server, such as the LLRP specification [91]. This thesis treats the Host as the coordinator of the entire system and the RFID Reader as a wireless gateway, analogous to a typical wireless sensor network.

³‘Backscatter’ describes a technique that allows the RFID tag to change the impedance of its own antenna over time, enabling the tag to reflect back more or less of the incoming signal in a pattern that encodes the uplink data, such as the tag’s ID.

2.2 Battery-less Edge Computing: CRFID Devices

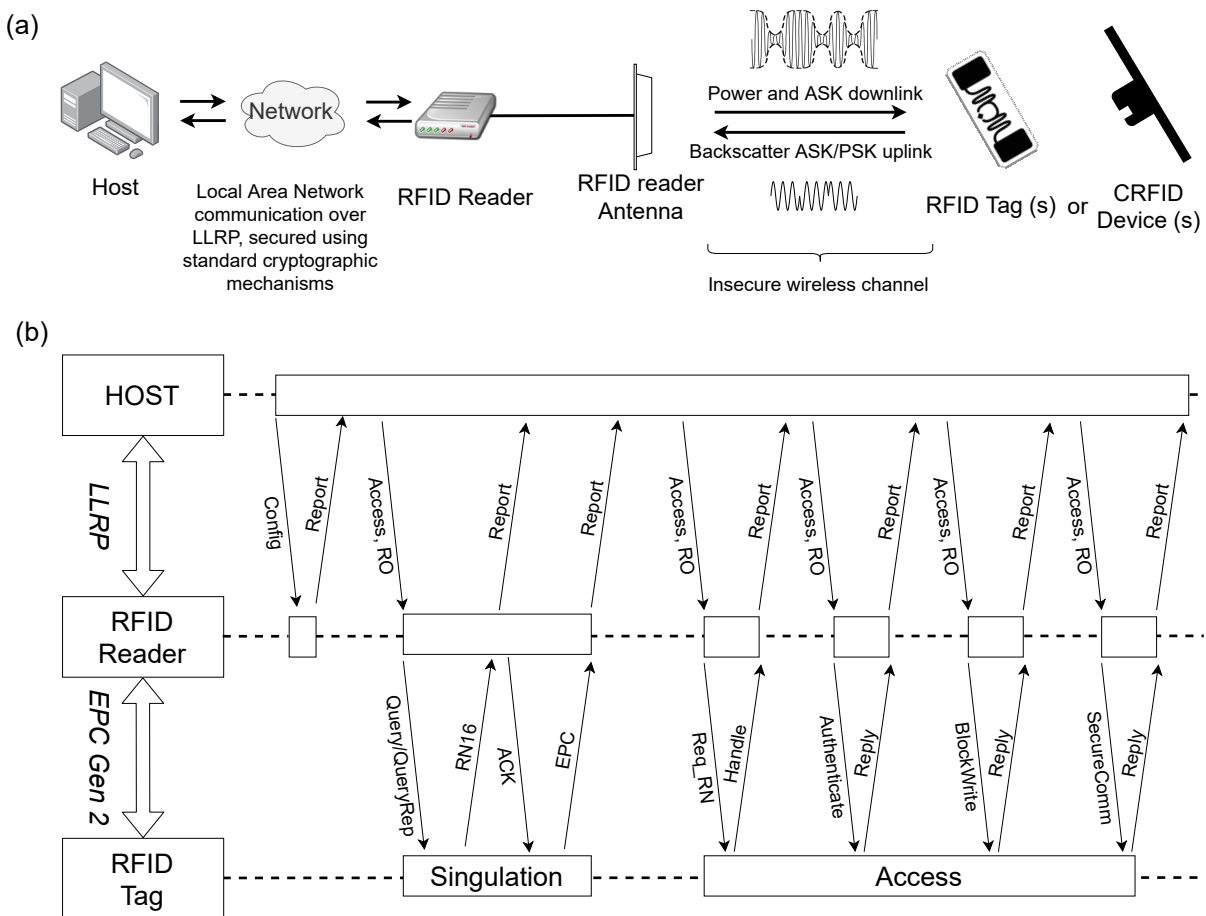


Figure 2.1: (a) Overview of a typical RFID system and (b) Control commands and data flow over the LLRP and the *EPC Gen2* air interface. Our focus is on the insecure communication channel between the RFID reader connected antenna and the RFID tags or CRFID devices. Hence, we assume that the communication between a host and a reader is secured using standard cryptographic mechanisms [89].

- Reader to tag.** As part of the anti-collision algorithm in the media access control (MAC) layer, the reader must first singulate an RFID tag and obtain a handle, RN16. Singulation is achieved as part of the inventory operation to discover RFID devices in a reader's powering and reading range. Inventory is performed using a combination of *Query*, *QueryRep*, and *QueryAdjust* commands. At the start of the inventory cycle, the reader transmits a *Query* command that notifies any tag within range of the beginning of a new inventory session. Each tag then selects a random slot counter between zero and an upper value, Q , defined by the *Query* command. If the selected slot counter is zero, the tag backscatters its handle, RN16, to the reader. If the slot counter is greater than zero, the tag waits for a *QueryRep* or *QueryAdjust* command. Upon receiving a *QueryRep*, the tag decrements its slot counter, and if the resulting counter value is zero, it backscatters the previously described response. Upon receiving a *QueryAdjust*, the tag adjusts the Q value and re-generates its slot counter.

The protocol flow is illustrated in Figure 2.1 (b). After the singulation of a tag, the Host can use *EPC Gen2 access command specifications* such as `BlockWrite` (specified in the *EPC Gen2*) to instruct the device to perform further operations, such as writing to device memory. For more information, please refer to the *EPC Gen2* protocol [92].

2.2.3 Computational RFID Devices

As discussed, CRFID devices represent an emerging class of battery-less, computational, sensor-enabled devices that operate on harvested energy. More importantly, CRFID devices retrieve sensor readings and exchange computational data via RFID-compatible channels. CRFID devices can be seamlessly integrated with existing passive RFID systems and extend RFID to sensing and ubiquitous computing applications. The development of UHF-based CRFID technology is illustrated in Figure 2.2. A conventional UHF RFID tag appears on the left-hand side, providing support for unique identification and a small amount of data storage (up to several KiB) [78]. The α -WISP next to it allows sensor sampling via the RFID channel by toggling mercury switches to enable one of two RFID chips with different identification (ID) numbers. By remotely reading the ID of the α -WISP, it is possible to read the orientation of the tagged object using an unmodified COTS RFID reader [82], [93], [94]. The Farsens Spider is a commercial RFID sensor tag that passively supports sensor sampling and data retrieval through the standard *EPC Gen2* channel [15]. However, the Farsens Spider is not programmable, largely limiting its flexibility. However, advances in low-power microelectronics mean that it is now possible to power programmable logic circuits and sensor sub-systems purely using harvested RF energy, as exemplified by the WISP—Intel WISP [13] and the WISP 5.1 LRG [9] in Figure 2.2 and introduced in Section 1.1. A WISP-equipped accelerometer and barometer can sample a plural number of sensors, process data and backscatter the result via the standard *EPC Gen2* channel [95]. Furthermore, combining sensor-enabled CRFID devices with flexible textile antennas, as in the case of the Wearable WISP [10], [11], make it possible to use battery-less devices as wearables in healthcare applications[17].



Figure 2.2: The development of computational RFID technology.

2.2.4 RF Energy Harvesting

CRFID devices operate on harvested RF energy. As illustrated in Figure 2.3 (a), even in an ideal setting, the harvested power is inversely proportional to the squared distance between the device and the radiating RFID reader antenna according to the Friis transmission equation [78], [96]–[98].

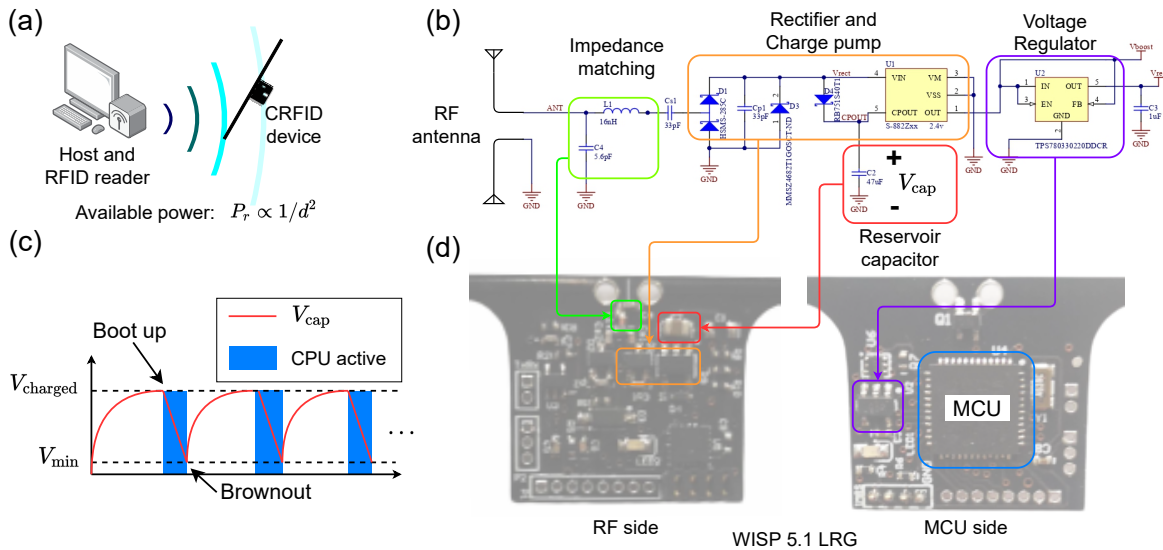


Figure 2.3: (a) Due to the free space path loss, ideally, the available power at the CRFID device is inversely proportional to the square of the operational distance; (b) The schematic (modified from the WISP 5 schematic [9]) of the RF energy-harvesting circuit of a typical passively powered device–WISP 5.1 LRG; (c) Charge burst operational mode and (d) A close-up view showing each component of the energy harvesting network of the WISP 5.1 LRG.

While a typical RF energy-harvesting architecture appeared in Figure 1.1 in Chapter 1, Figure 2.3 (b) uses WISP 5.1 LRG as an example to illustrate the circuit-based realisation of RF energy harvesting. This circuit features five major functional blocks: 1) a dipole antenna, which interacts with the incident electromagnetic waves and transfers the RF energy to the flow of electric charges; 2) an L-network impedance matching circuit, which provides maximum power transfer from the antenna to the rectifier; 3) the combination of rectifier diodes and an S-882Z charge pump, which converts the small and alternating charge flow into a usable DC voltage; 4) a 47 uF reservoir capacitor, which buffers and stabilises the harvested energy; and 5) a TPS7803 Low-Dropout Regulator (LDO), which ensures the output voltage is maintained at the desired level ready to be delivered to the load (in this example, the MCU and sensors).

Once the voltage at the reservoir capacitor, V_{cap} , reaches a threshold, $V_{charged}$ —see Figure 2.3 (c)—the MCU is booted up. The CPU inside the MCU can execute code, sample sensors, prepare the up-link packets, and control the backscattering of data to the RFID reader. If V_{cap} drops below the minimum supply voltage required by the MCU, a brownout occurs, and

the MCU ceases to work. The CRFID device must then start recharging its reservoir capacitor. This *boot-up and brownout cycle* characterises the intermittent powering conditions experienced by CRFID devices.

Figure 2.3 (d) shows the corresponding RF energy-harvesting components on the WISP 5.1 LRG studied in this thesis as an example.

2.3 Memory PUFs or Memory Fingerprinting

We use the terms ‘memory fingerprints’ and ‘memory PUFs’ interchangeably. As discussed Section 1.2.3 in Chapter 1, memory is ubiquitously available in electronic devices and computing platforms such as microcontrollers, with memory PUFs (or memory fingerprints) simply re-purposing ubiquitously available memory to build security functions.

2.3.1 Memory PUFs

In this section, we review the construction of memory PUFs using different memory technologies. Although some researchers [99], [100] classify PUFs built upon memory-like synthesised circuits as memory PUFs (e.g., Latch PUF [101], Flip-flop PUF [102], Butterfly PUF [103], and Buskeeper PUF [104]), these require either expensive configurable logic (such as FPGA or CPLB) or fabrication on dedicated hardware. This makes it challenging to implement such PUFs in low-cost commodity devices. This section reviews memory PUFs built on popular memory technologies commonly available to commodity electronic devices. The PUF variants we discuss employ various methods for extracting fingerprints from device memories, including SRAM, DRAM and Flash. Although PUF construction based on emerging memory technologies are popular in the literature (e.g., SHIC [105], STT-MRAM PUF [106] and memristive PUFs [107]–[109]), these memory types as yet unavailable to low-cost computational modules, such as microcontrollers. Therefore, this thesis does not discuss these. Interested readers can obtain further information on these emerging memory-based PUFs from [99], [110].

SRAM PUF

Static random-access memory (SRAM) is probably the most common digital memory type. Inside an SRAM chip—depicted in Figure 2.5—an essential component is an SRAM array. Besides the SRAM array, some auxiliary circuits, such as the control logic, coordinate

2.3 Memory PUFs or Memory Fingerprinting

communication with other devices, such as a CPU. There are also address latch and decoders, I/O latch and bus drivers to perform address and data read/write operations.

In addition to standalone chips, SRAMs are also commonly integrated with other logic circuits, such as the cache inside a CPU, directly addressable via a CPU's internal high-speed bus [111].

Regardless of how they are constructed, all SRAM types share the same principle: the SRAM cells store information, and each SRAM cell holds one binary bit. As Figure 2.4 illustrates, one SRAM cell comprises six transistors (M_1 - M_6). PMOS M_1 , M_3 and NMOS M_3 , M_4 form two cross-coupled digital inverters, and two additional transistors M_5 and M_6 act as bit selectors to perform read/write operations on one specific SRAM cell. The cross-coupled inverter-pair creates a bi-stable state. In normal conditions, SRAM tends to maintain its existing state. During write operations, an external driving signal applied to a single SRAM cell changes its state. However, because SRAM's bi-stable state requires uninterrupted powering to avoid losing its state upon losing power, SRAM is a volatile memory type.

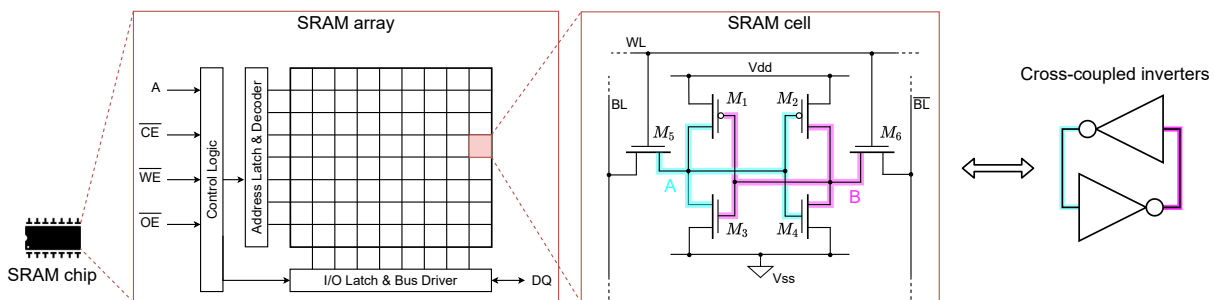


Figure 2.4: The basic architecture of SRAM.

In 2007, Guajardo [49] and Holcomb [60] each independently proposed an SRAM PUF. Their approaches consider the SRAM memory address a PUF challenge and the start-up state the response, as shown in Figure 2.5 (a). Notably, the SRAM PUF is an *intrinsic* PUF [100], [112], meaning that realisation of the PUF does not require a custom design, additional chip area overhead or hardware modification [49], [113].

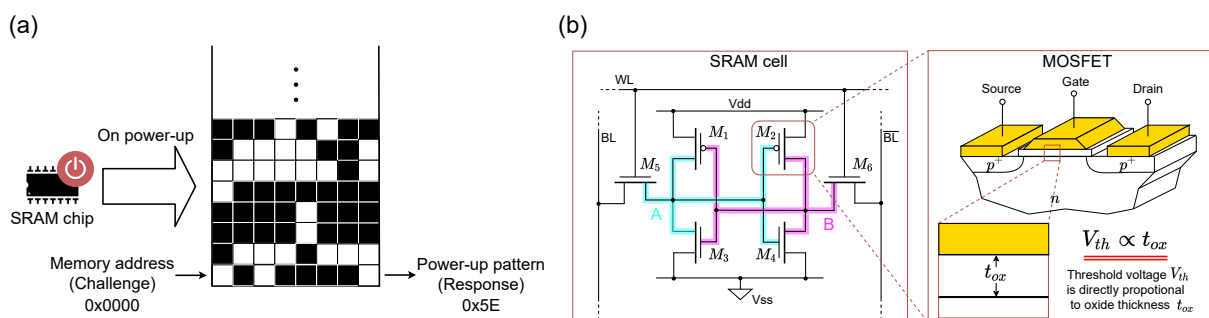


Figure 2.5: (a) an SRAM PUF (start-up state) and (b) mismatched MOSFET threshold voltage V_{th} .

Upon power-up, the two digital inverters in an SRAM cell are trying to overcome each other, akin to two tug-of-war combatants. Eventually, the SRAM cell will fall into one of the two stable states. Digital inverters have different ‘strengths’, and as in a game of tug of war, the stronger participant often wins. As shown in Figure 2.5 (b), the thickness of gate oxide t_{ox} in a MOSFET determines its threshold voltage V_{th} . A MOSFET with lower V_{th} may switch faster than others, meaning the corresponding digital inverter has a ‘stronger’ driving force, enabling it to (usually) dominate the SRAM start-up state. Notably, the t_{ox} is randomly assigned during the manufacturing process.

Because the initial power-up state of each SRAM cell is random but reproducible [60], the power-up pattern of bits generated from an SRAM memory can be considered a unique identifier. In such an SRAM PUF, the address of the SRAM cell acts as the challenge, and the initial power-up state—‘1’/‘0’—acts as the response.

The analytical model constructed by Roel Maes et al. [114] to describe the SRAM start-up state (Equation 2.1) sees two random components contribute (response $r_i^{(j)}$). For the i^{th} SRAM cell, the stochastic variable m_i models the mismatched inverter’s inherent property, and another stochastic variable $n_i^{(j)}$ describes the electrical noise at the moment of j^{th} evaluation (i.e., the j^{th} start-up for SRAM PUF). Roel Maes et al. assume that both m_i and $n_i^{(j)}$ are subject to normal distributions, with T a threshold parameter for a specific SRAM technology.

$$r_i^{(j)} = \begin{cases} 0 & , \text{ if } m_i + n_i^{(j)} > T \\ 1 & , \text{ if } m_i + n_i^{(j)} \leq T \end{cases} \quad (2.1)$$

Meanwhile, Xu et al. [115] investigated SRAM data retention voltage (DRV) in 2015. As previously discussed, SRAM requires uninterrupted power to maintain its state. If the supply voltage is gradually reduced, some SRAM cells may lose their state, and some may maintain their state. The minimum voltage required for an SRAM cell to maintain its state is the DRV. Each cell’s DRV is subject to uncontrollable random variation during the manufacturing process, and measuring the DRV requires that the SRAM bank be separately powered and the supply voltage be finely adjusted, which most electronic systems do not support. This impracticability hinders implementation and further development of the SRAM PUF.

DRAM PUF

DRAM is another memory type frequently used in modern computing systems, with examples including the RAM module in PCs and mobile phones. DRAM also plays a vital role in IoT scenarios, such as providing a web camera’s first-in-first-out buffer [116]. Because DRAM has

2.3 Memory PUFs or Memory Fingerprinting

only one transistor in each cell instead of the six that SRAM features, DRAM typically has a much higher capacity than even SRAM in identical manufacturing technology contexts.

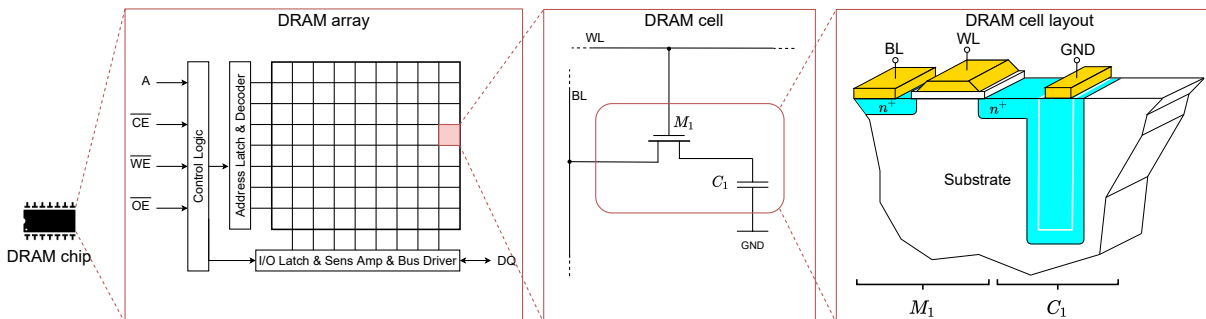


Figure 2.6: The basic architecture of DRAM, showing the array, cell and physical layout of a single DRAM cell. Modern DRAM uses deep trench capacitors to maximise component density [117].

DRAM's basic architecture is depicted in Figure 2.6. The top-level architecture of DRAM is very similar to that of SRAM. However, there is only one access transistor M_1 and one capacitor C_1 in a DRAM cell. DRAM stores information by charging and discharging the C_1 . To write logic '1' into a DRAM cell, the corresponding word-line (WL) is pulled up, and the bit-line (BL) is fed with V_{charge} , charging the C_1 . In contrast, the BL is grounded to discharge the C_1 while writing logic '0'. The DRAM read-out is somewhat complicated. The controller first pre-charges the BL to $V_{charge}/2$ and then energises the WL. If the C_1 is charged, charges will flow out of the DRAM cell, with a positive current captured by the underlying sense amplifier. Otherwise, charges flow into the discharged DRAM cell and induce an opposing current. The direction of the current flow is subsequently translated to the logic '0' and '1'. It must be emphasised that the DRAM reading operation is destructive, meaning the information is destroyed after reading it out, making a subsequent write-back necessary.

Factors including junction leakage, gate-induced drain leakage, off-leakage, field transistor leakage and capacitor dielectric leakage cause the charge in C_1 to escape over time [118], demanding periodically refreshing DRAM to compensate for this loss. In modern computers, the DRAM controller automatically performs regular read-out and write-back. Charge leakage means DRAM eventually loses its state upon losing power, which makes DRAM a volatile memory.

In 2015, Tehranipoor et al. [119] observed that the discharge rate of each DRAM cell is determined by the random hardware variation of M_1 and C_1 and could be used to instantiate a PUF concept known as retention-based DRAM PUF.

The leakage charge from one DRAM cell may also affect the state of an adjacent DRAM cell. Again, the strength of this correlation is subject to the random fabrication variation. In 2017, Schaller et al. exploited this phenomenon to propose DRAM row hammer PUF [120].

Flash PUF

Flash memory is currently probably the most well-known non-volatile memory. It is used in the solid-state drives of PCs, mobiles and servers and the firmware storage of computer components, routers, modems, and switches. Many MCU, transceivers and sensors store executable code and calibration settings in Flash memory in IoT scenarios.

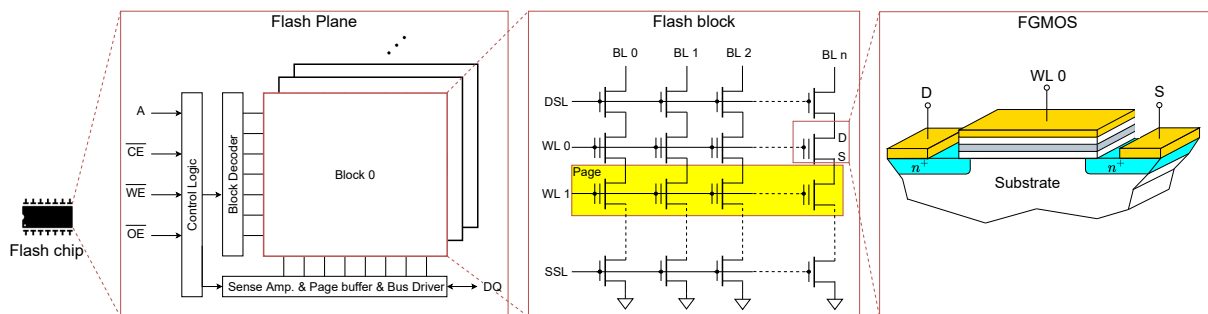


Figure 2.7: The basic architecture of NAND Flash, showing the Flash Plane, block, page and the physical structure of one floating-gate MOSFET.

The basic structure of Flash memory is illustrated in Figure 2.7 using the NAND Flash architecture as an example. NOR is a less common Flash-type storage due to its lower density and slower speed. One Flash chip die contains one or more Flash Planes, and each Plane features auxiliary circuits, including control logic, decoder and sense amplifier circuits. There are many Flash blocks in a single Flash Plane, and a block is the smallest unit for erase operation. The Flash block is further divided into pages. Flash cells in a row sharing the same WL form a Flash page. A page is the smallest unit to perform read and write operations. Manufacturers define the size of pages and blocks, with page size of 4 KiB and planes size of 256 KiB being common practice. Each Flash cell features only a single floating-gate MOSFET (FGMOS) and no access transistor, making it infeasible to access a single specific Flash address in the matter of RAM types. However, its single-transistor organisation means Flash can be manufactured to be incredibly high dense.

Digital bits are represented by charges trapped in the floating gate of FGMOS, which can be injected and removed by applying a higher program voltage. If charges are present in the floating gate, they prevent a channel forming between the drain (D) and source (S) of the FGMOS, meaning no current flows through, read as logic '0'. Otherwise, if no charges are trapped in

2.3 Memory PUFs or Memory Fingerprinting

the floating gate, a current, I_{DS} , will flow from D to S via the channel and be captured by the underlying sense amplifier, read as logic '1'. Flash memories are non-volatile memory. Because insulators contain the floating gate, it is difficult for trapped charges to escape. Modern NAND Flash memories feature typical data retention of about ten years.

Wang et al. [53] first proposed a Flash-based PUF in 2012, applying a technique called partial programming, which builds on the understanding that injecting charges into the floating gate of Flash memory does not happen in no time. Instead, this time is measured in hundreds of microseconds, and the process can be interrupted by sending a reset command (RST) during the Flash programming process, producing Flash cells in a partially programmed state. Random hardware fabrication variations mean that different Flash cells may be programmed at distinct speeds. Furthermore, following a particular partial program time, some cells may be programmed while others not. The address to a specific Flash page is considered a challenge, and the partial program bit pattern acts as the response.

In 2015, Jia et al., [121] proposed two new methods for extracting a Flash PUF's randomness: partial erasure and program disturbance. As previously noted, Flash erasing is performed block by block, and Flash programming is performed page by page. Partial erasure is 64 times faster than partial programming. The program disturbance utilise the interference between two physically adjacent Flash pages, which is conceptually much similar to the DRAM row hammer we have discussed in Section 2.3.1. In addition to the randomness extraction methods, this study also proposes a post-processing scheme to acquire reliable responses based on the differential programming time between two adjacent Flash addresses.

Summary

This thesis focuses on SRAM PUFs (or SRAM fingerprints) because SRAM instances are widely available in various devices, including battery-less and intermittently powered CRFID devices. As an intrinsic PUF, realising SRAM PUFs does not require a custom design, additional chip area overhead or hardware modifications [100]. Additionally, in some embedded systems [122], each memory bank can be individually powered off by exploiting a particular power control register, facilitating the *run-time fingerprinting of the integrated device memory*.

2.3.2 Memory PUF Evaluation Measures

This section outlines the common PUF evaluation measures applicable to memory PUFs, including reliability, uniformity and uniqueness. The derivation and formalisation of these measures are available in [112], [123].

Reliability

Reliability [73], [112], [124]–[126] (also referred to as robustness, steadiness, stability or reproducibility [53], [127], [128]) indicates how stable the PUF responses are, that is, the extent of the PUF’s ability to reproduce the same response to a corresponding challenge. Ideally, reliability should be 100% (i.e., without any error across repeated PUF evaluations). In practice, some factors may inhibit a PUF’s ability to reproduce responses reliably, most prominently environment temperature variations, supply voltage offsets and ageing.

A reference response, \mathbf{r}_i , is recorded from hardware instance i under normal operating conditions (such as room temperature and rated voltage supply) before another response, \mathbf{r}'_i , is requested under different operating conditions for the same challenge. We calculate the Hamming distance (HD) between \mathbf{r}_i and \mathbf{r}'_i , calling it the intra-class HD (intra-HD) because we are evaluating the same PUF instance with the same challenge. The ideal intra-HD value is 0.

Generally, the expected bit error rate (BER) is used to describe intra-HD, estimating the average ratio of the number of bit errors in the total number of evaluated response bits over the n -bit binary response vector. If the n -bit response \mathbf{r}_i is re-evaluated from the same hardware instance i over m times, under a different operating condition with value $\mathbf{r}'_{i,t}$ at the t^{th} time, the average intra-class HD or the expected BER is defined as:

$$\text{BER} = \frac{1}{m} \sum_{t=1}^m \frac{\text{HD}(\mathbf{r}_i, \mathbf{r}'_{i,t})}{n} \times 100\% \quad (2.2)$$

Then, the reliability of a PUF is defined as:

$$\text{reliability} = 1 - \text{BER} \quad (2.3)$$

We follow the definition in [112] and refer to the **worst-case BER** as the largest intra-HD which can be expected within a given range of operating conditions with respect to a particular reference condition. To find the worst-case BER in a given range of operating conditions, we can selectively evaluate the intra-HD around the boundaries of this operating range. This technique is

2.3 Memory PUFs or Memory Fingerprinting

reasoned based on the assumption that intra-HD with respect to the reference condition increases, if the evaluation is conducted under an operating condition further away from the reference.

Some of the literature [53] also uses the correlation coefficient between two responses repeatedly evaluated on the same challenge from the same device to describe reliability.

Uniqueness

Uniqueness describes how easily we can distinguish one PUF device from another. When applying the same challenge, \mathbf{c} , to different PUFs, the response vectors from different PUFs are expected to differ due to the intrinsic variations of each PUF. Uniqueness is a highly desirable characteristic that measures the amount of unique information extracted from a PUF. Uniqueness is measured by inter-class HD (inter-HD). If two distinct PUFs, i and j , have n bit responses, they produce \mathbf{r}_i and \mathbf{r}_j response vectors, which correspond to the same challenge \mathbf{c} . Ideally, uniqueness should equal 50%, with the average inter-HD among k PUF instances defined as:

$$\text{Uniqueness} = \frac{1}{\binom{k}{2}} \sum_{i=1}^{k-1} \sum_{j=i+1}^k \frac{\text{HD}(\mathbf{r}_i, \mathbf{r}_j)}{n} \times 100\% \quad (2.4)$$

Some of the literature [53] also uses the correlation coefficient between responses from two PUFs to describe uniqueness.

Uniformity

Uniformity (also referred to as randomness or bias [129]) is an indicator of the balance between ‘0’ and ‘1’ in a response vector. Ideally, a PUF should have equal probability to output ‘0’ or ‘1’ in its response vector, that is, uniformity should equal 50%. Uniformity is analytically defined as:

$$\text{Uniformity} = \frac{1}{n} \sum_{l=1}^n r_{i,l} \times 100\% \quad (2.5)$$

Where $r_{i,l}$ is the l^{th} binary bit of an n -bit response from a hardware instance i .

2.3.3 Reliable Key Generation with PUFs

PUF response re-generation is susceptible to fluctuations in environmental conditions, including thermal noise and power supply and temperature variations (as discussed in Section 1.2.3 in Chapter 1). Thus, PUF responses cannot be used directly as keys in security functions and

require error correction to reconcile flipped response bits (errors) in relation to a reference response. This section details the state-of-the-art techniques used to generate reliable and secure keys from PUFs or memory fingerprints: 1) (reverse) fuzzy extractor to perform noise compensation or error correction; and 2) fingerprint pre-processing to enhance the reliability of the memory fingerprints.

(Reverse) Fuzzy Extractor

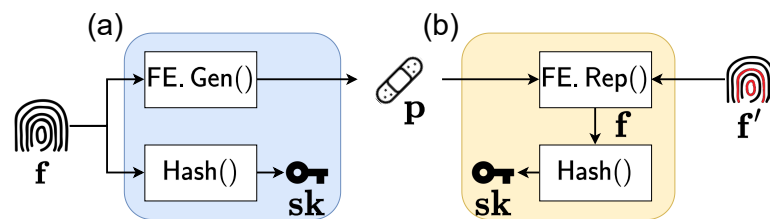


Figure 2.8: (Reverse) fuzzy extractor. Based on f , the first phase concurrently generates helper data p via the encoding function (also referred to as the helper data generation function $FE.Gen()$) and secret sk via an entropy extractor (typically a hash function $Hash()$). The re-generation phase uses p and reevaluates f' to restore f by reconciling errors in f' using a decoding function (also referred to as a reproduction function $FE.Rep()$) to reproduce sk . In a reverse fuzzy extractor, (a) is embedded in a device and (b) is offloaded to the server.

A fuzzy-extractor-based PUF key generator can turn a fingerprint, f , into a cryptographic key, sk , with full bit entropy, as shown in Figure 2.8. A fuzzy-extractor (FE) comprises of a secure sketch and an entropy extractor [73], [124], [130]. The secure sketch computes helper data based on the provisioned fingerprint and subsequently utilises helper data to correct errors in the re-generated fingerprint. Two prevalent secure sketch schemes are used to realise an FE: code-offset construction and syndrome construction [73]. This thesis considers syndrome-based construction because syndrome (helper data) has been demonstrated to not leak information about the original memory fingerprints [129], [131]. We shall briefly describe this construction here.

The secure sketch construction has two functions: $FE.Gen()$ and $FE.Rep()$. During the key enrolment phase, helper data, p , are computed using $FE.Gen(f)$, where $p = f \times H^T$, and H is a parity check matrix of a linear error correction code. The key reconstruction described by $FE.Rep(f', p)$, where f' is the reproduced fingerprint, which may slightly differ from the enrolled fingerprint, f , first constructs a syndrome, $s = (f' \times H^T) \oplus p = e \times H^T$, where e is an error vector. Next, an error location algorithm determines e . Subsequently, the fingerprint f is recovered via $f = e \oplus f'$. Because the recovered PUF fingerprint f may be non-uniformly distributed, an entropy extraction operation—for example, via a universal hash function—compresses the PUF fingerprint into a cryptographic key with full bit entropy.

2.3 Memory PUFs or Memory Fingerprinting

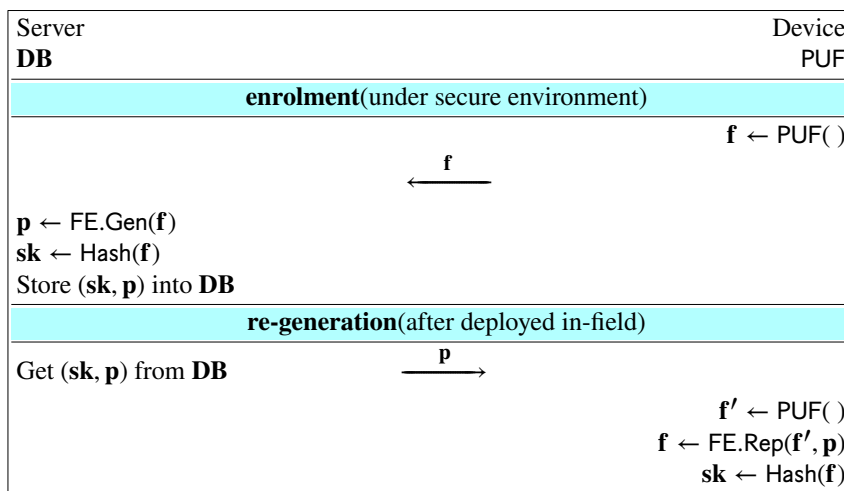


Figure 2.9: Fuzzy extractor concept.

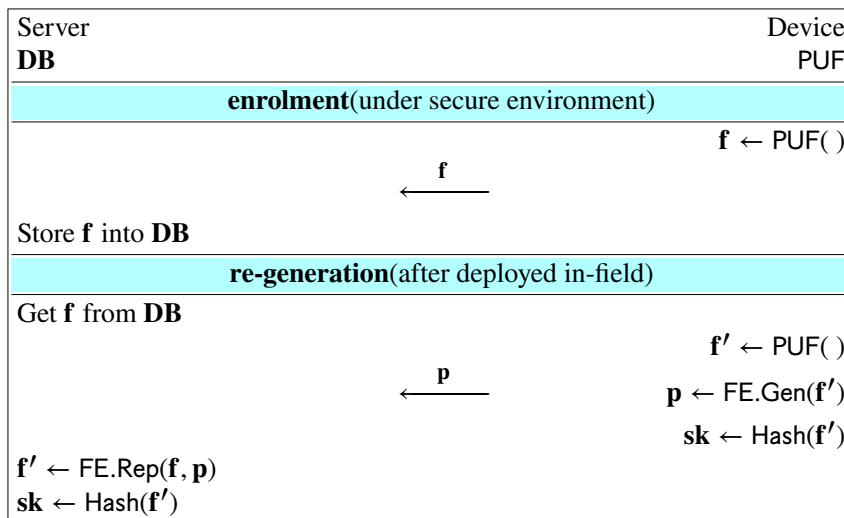


Figure 2.10: Reverse fuzzy extractor concept.

Normally, in an FE setting, as shown in Figure 2.9, the $\text{FE.Gen}()$ function is executed by the server during the provisioning phase to compute helper data. In the field, the $\text{FE.Rep}()$ function is implemented on a token. By recognising that the computational burden of the $\text{FE.Rep}()$ function significantly exceeds the $\text{FE.Gen}()$ function, Van Herrewege *et al.* [66] place the $\text{FE.Gen}()$ on the resource-constraint token and leave the computationally intensive $\text{FE.Gen}()$ function execution to the resource-rich server. This method, known as the reverse fuzzy extractor (RFE) method, is depicted in Figure 2.10.

However, as discussed in Section 1.2.3 in Chapter 1, the high implementation overhead of the encoder in its use in the RFE setting is not desirable for resource-limited devices. More importantly, the vulnerability of the associated helper data to helper data manipulation (HDM)

attacks remains unsolved. Solutions considered to mitigate HDM, including hashing the helper data with the generated PUF key, can be found in [73].

Fingerprint Pre-Processing Techniques

Differing from RFE, fingerprint pre-processes, including pre-selection and majority voting, can be adopted during one-off enrolment. These methods incur no overheads during fingerprint re-generations but introduce more complex enrolment steps.

Pre-Selection. According to Maes [114], each SRAM cell may have a different start-up value across repeated evaluations. Building on this, Böhm *et al.* [132] developed a technique called pre-selection to exclusively select those SRAM cells that always produce reliable start-up values across repeated evaluations. The selected stable SRAM cells are used for secure key generation.

However, this [114], [132] requires numerous repeated evaluations, which is tedious and time-consuming. Additionally, the reliability measurement is empirical and coarse-grained. For example, when a given bit is constantly '1'/'0's across 1000 repeated measurements under a given operating condition, one can only empirically conclude that the BER for this bit is below 10^{-3} . However, this conclusion cannot be firmly held following a shift in operating conditions. Worse, identifying bits with a BER below 10^{-9} requires billions of physical measurements, which is infeasible in practice.

Hofer *et al.* [133] proposed a delay-based method to identify reliable SRAM cells, understanding that highly mismatched SRAM cells not only tend to always produce a certain start-up value but also take less time to settle down. This method requires low-level access to each cell to monitor the settling time, which is generally not supported by commercial devices.

Majority voting. Two types of majority voting on enrolment enhance reliability. Guajardo *et al.* [49] developed temporal majority voting (TMV), which involves sampling the device fingerprint several times before adopting the majority result as the enrolled reference. Similar to pre-selection, TMV requires multiple evaluations; however, it requires comparatively small numbers during the enrolment phase. According to [102], TMV is effective if the raw noise level is already relatively low. However, TMV becomes inefficient for fingerprints with a higher BER.

Spatial majority voting (SMV) has also been proposed. According to Maes *et al.* [102], in SMV, instead of sampling the same fingerprint bit multiple times, various nearby fingerprint bits can be grouped, with the majority value taken to produce a more reliable bit, improving a device fingerprint's uniformity by selecting a skewed threshold.

2.3 Memory PUFs or Memory Fingerprinting

2.3.4 Memory Fingerprinting and Memory PUF Datasets

This section describes and compares the datasets used in this thesis, both those contributed to by this research and those publicly available. Evaluation results are summarised in Table 2.1. The memory types associated with the listed datasets are pervasive, especially in low-end devices. Example COTS devices appear in Figure 2.11 alongside the evaluated memories.

We compare the evaluation conditions (temperature variation, supply voltage offset and chip ageing) under which experiments have been performed for each dataset and report the number of individual chips, size of physical chips and number of repeated evaluations in each dataset. The evaluation results focus on the three fundamental measures outlined in Section 2.3.2: uniqueness, reliability and uniformity.

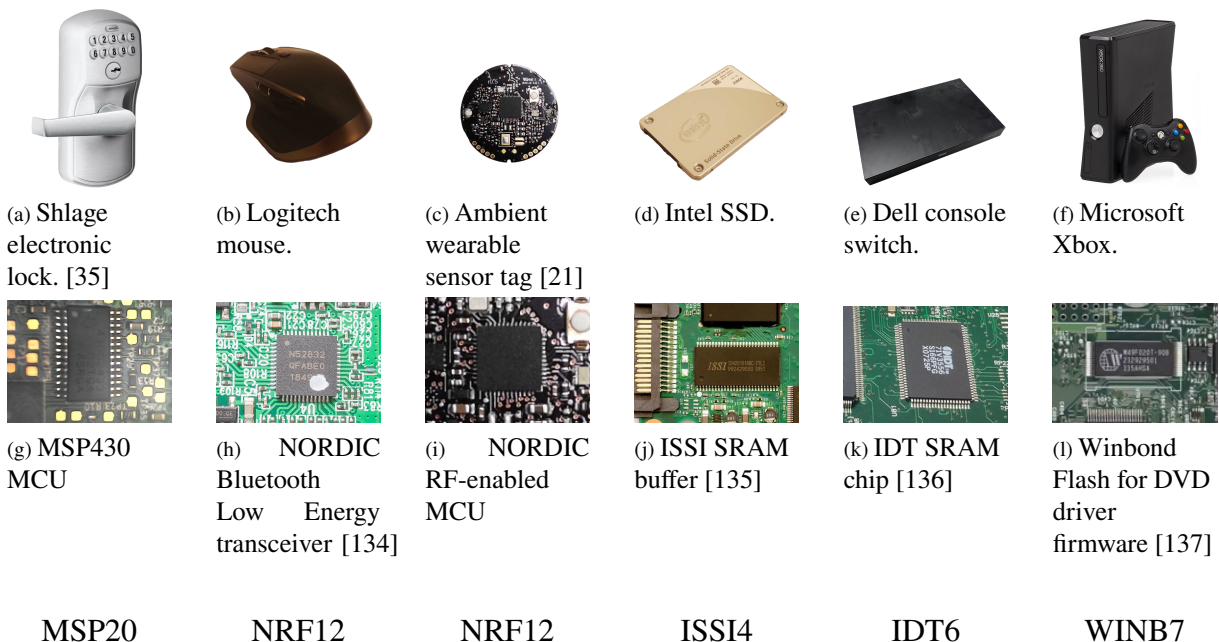


Figure 2.11: Evaluated memories (middle), example products (top) that use these types of memories and corresponding datasets (bottom).

To distinguish the datasets, we use the following naming convention: [Chip name abbreviation]+[number of chips]. For example, ‘MSP20’ indicates a dataset collected from 20 MSP430FR5969 MCUs.

MSP20 dataset (*Thesis Contribution*)

The dataset MSP20 collected in Chapter 3 evaluates the start-up states of the 2 KiB SRAM block inside 20 fresh MSP430FR5969 MCU chips. The data were collected at five temperature points: -15°C , 0°C , 25°C , 40°C and 80°C . The evaluation was repeated 100 times and the evaluation

Table 2.1: Overview of memory fingerprinting (or memory PUF) datasets, both those contributed to by this research and those publicly available.

Work Name[reference]	Experiment Setups				Evaluation Results		
	Evaluation conditions	Chip number	Memory size	Repeat times	Uniqueness	BER (worst-case)	Uniformity
Datasets contributed from this thesis							
MSP20 [Chapter 3]	-15 °C to 80 °C	20	2 KiB	100	45.30%	11%	49.99%
NRF12 [Chapter 6]	-15 °C to 80 °C	12	64 KiB	100	49.89%	6.09%	54.94%
Datasets publicly available							
ISSI4 [138]	25 °C to 80 °C	4	256 KiB	30	24.43%	8.29%	49.40%
IDT6 [138]	25 °C to 80 °C	6	512 KiB	50	49.74%	5.42%	48.66%
WINB7 [54]	0-100,000 P/E Cycles Aging	7	69,696 Bytes	99	48.75%	16.26%	49.34%

results were saved as binary (.bin) files . The average uniqueness of the 20 devices in the MSP20 dataset is 45.30%. The worst-case BER (11%) occurs at 80 °C , when the reference template is collected at 25 °C . The average uniformity of the dataset is 49.99%.

This dataset is available at: <https://dx.doi.org/10.21227/H27T0S>

NRF12 dataset (*Thesis Contribution*)

The dataset NRF12 collected in Chapter 6 evaluates the start-up states of the 64 KiB SRAM blocks inside 12 nRF52832 chips. The nRF52832 is a popular RF-enabled MCU and supports various protocols, including Bluetooth 5, Bluetooth mesh, ANT and NFC. It is worth mentioning that the NORDIC chips are typical of a low-cost device MCU. The data were collected at three temperature points: -15 °C , 25 °C , 80 °C . The evaluation was repeated 100 times at each temperature and the evaluation results were saved as binary (.bin) files .

The average uniqueness of the 12 devices in the NRF12 dataset is 49.89%. The worst-case BER (6.09%) occurs at 80 °C , when the reference template is collected at 25 °C . The average uniformity of the dataset is 54.94%.

This dataset is available at: <https://dx.doi.org/10.21227/ktc9-x515>

ISSI4 and IDT6

The ISSI4 and IDT6 datasets were collected by Guo et al. [138] from two COTS SRAM models: IS61WV25616BLL and IDT71V416S, respectively. The data were collected at two temperature

2.4 Chapter Summary

points: 25 °C and 80 °C . The evaluation was repeated a varying number of times at each temperature point for each dataset (see Table 2.1). The evaluation results were saved as Matlab workspace variable (.mat) files.

The average uniqueness of the four devices in the ISSI4 dataset is 24.34%. The worst-case BER (8.29%) occurs at 80 °C , when the reference template is collected at 25 °C . The average uniformity of the dataset is 49.40%.

The average uniqueness of the six devices in the IDT6 dataset is 49.74%. The worst-case BER (5.42%) occurs at 80 °C , when the reference template is collected at 25 °C . The average uniformity of the dataset is 48.66%.

These datasets are available at:

<https://www.trust-hub.org/#/data/memory-aging-recycling>

WINB7 Dataset

The dataset WINB7 was collected by Guo et al. [54] from seven W29N02GV COTS Flash chips. A partial program technique (described in Section 2.3) was used to test the Flash chips. Differing from the SRAM datasets, this Flash dataset uses ageing as an evaluation condition (from fresh to 100,000 program/erase cycles). The results were saved as Matlab workspace variable (.mat) files.

The average uniqueness of the seven devices in the WINB7 dataset is 48.75%. The worst-case BER (16.26%) occurs at 100,000th Program/Erase cycles where the reference template is collected when the chips are fresh. The average uniformity of the dataset is 49.34%.

This dataset is available at:

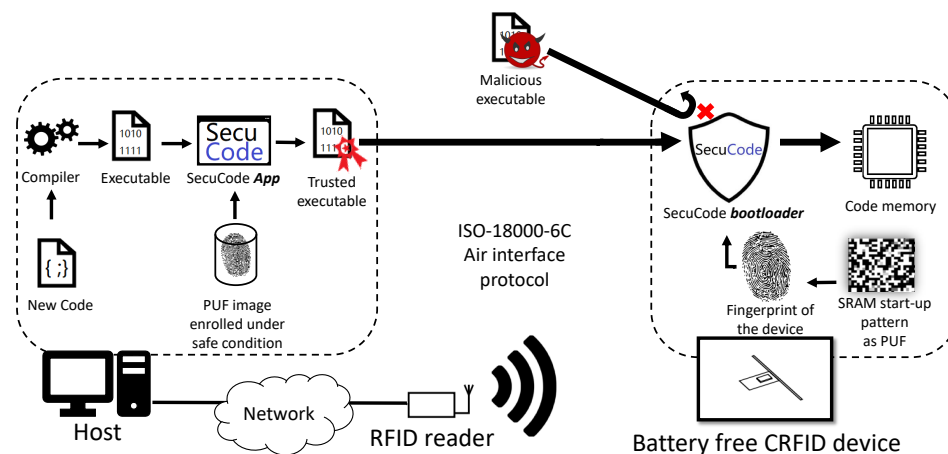
<https://www.trust-hub.org/#/data/memory-aging-recycling>

2.4 Chapter Summary

This chapter has defined the notations and conventions used in this thesis before detailing the fundamentals of CRFID (EPC-enabled UHF RFID technology, the development of CRFID devices and RF energy harvesting) and memory PUFs (or memory fingerprints; common memory PUFs, memory PUF evaluation measures and state-of-the-art methods used for reliable key generation using PUFs). The chapter also summarised the relevant observations of both publicly available memory-oriented datasets and the datasets produced by extensive chip measurements conducted as part of this research.

Chapter 3

Securing Wireless Code Updates to Intermittently Powered Devices



THE security of wireless code updates for resource-constrained and intermittently powered devices, such as the Computational RFID (CRFID), is a challenging problem due to the resource limitations and lack of hardware security features discussed in Chapter 1. This chapter develops, *for the first time*, a *secure* wireless code dissemination (SecuCode) mechanism for such an intermittently powered and resource-limited device by entangling a *device intrinsic hardware security primitive*—Static Random Access Memory Physical Unclonable Function (SRAM PUF)—to a firmware update protocol. The design of SecuCode: i) overcomes the resource-constrained and intermittently powered nature of the target CRFID devices; ii) is fully compatible with existing communication protocols employed by CRFID devices—in particular, ISO-18000-6C protocol; and iii) is built upon a standard and industry compliant firmware compilation and update method realised by extending a recent framework for firmware updates provided by Texas Instruments. We built an end-to-end SecuCode implementation and conduct extensive experiments to demonstrate standards compliance, evaluate performance and security.

3.1 Motivation and Contribution

Firmware updates can help to enhance devices' computational or storage performance, impart new functionality, fix software bugs or address system compatibility. In the absence of standard protocols or system-level support, *application specific software* in the form of firmware for resource-limited devices such as CRFID is typically updated using a wired programming interface [9], [14]. However, a wired connection not only negates the benefit of the battery-free feature and makes the process unscalable when potentially millions of devices need to be updated but is a more acute problem in the context of deeply embedded devices, such as when devices are deeply embedded in reinforced concrete structures [29], integrated with aircraft parts [22], [25] or inserted inside body organs [139]. In such a case, physical access to the device poses practical challenges and risks to the end-user. Therefore, wireless firmware updates are highly desirable for such kinds of applications.

The fundamental problem of a wireless firmware update for CRFID devices was addressed in two recent approaches [29], [140]; both focused on transmission reliability, miniaturisation of firmware code size, and energy efficiency. None of them addressed the difficult problem of assuring the security of wireless firmware updates, although Tan et al. [29] highlighted that secure wireless code dissemination remains the most urgent need to be addressed. This implies that both wireless approaches allow any party to remotely and wirelessly install code on a CRFID device, irrespective of their trustworthiness. Consequently, malicious firmware injection from an adversary remains a direct threat that can lead to, for example, private information leakages such as health conditions to unauthorised parties or the installation of malicious code in deeply embedded hardware such as a blood glucose monitor with devastating consequences for the victim.

Therefore, this chapter considers the challenging problem of a secure wireless firmware update to eschew the cumbersome and, often impractical, cable-connected download method and address the following problems:

Problem 1. How can we *securely update firmware* under *resource constraints* and *intermittent powering*, as illustrated in Figure 1.3 and Figure 1.4, relying only on harvested power and without additional hardware components?

Problem 2. How can we develop an update protocol *compliant with current communication protocols* to ensure translation into practice?

We summarise the contributions made in this chapter towards addressing the aforementioned problems below:

- **Develop a first secure wireless firmware update protocol.** SecuCode is the first *secure* wireless firmware update or reprogramming method for CRFID devices to prevent malicious firmware injection attacks⁴(**Problem 1**).
- **Develop a lightweight physically obfuscated key derivation mechanism.** We develop a key derivation mechanism using a physical unclonable function (PUF). A PUF converts hardware instance specific random variations such as gate and wire delays of circuitry to a binary value. In particular, we take advantage of an SRAM PUF built-upon random start-up state of intrinsic SRAM on microcontrollers to extract key material on the fly without extra hardware overhead and modifications. The key derived from such an SRAM PUF is therefore: i) intrinsically tamper-resistant (unclonability) compared to a permanently stored key in non-volatile memory; ii) unique, through the hardware instance specific nature of SRAM cells' startup state; iii) unpredictable, through the physical randomness leading to random startup states of SRAM cells; and iv) never stored permanently in NVM as it is derived on the fly and discarded after usage. The *contribution* in this chapter is to address the challenges faced in realising a lightweight, reliable and secure⁵ key generator using an on-chip SRAM PUF on a resource-constrained device (**Problem 1**).
- **Complete end-to-end design and implementation.** We demonstrate an end-to-end design from firmware compilation to a successful firmware update process supported by an execution model on a CRFID token to manage the transient nature of power availability. We develop a tool (SecuCode App) to update firmware and conduct a complete end-to-end implementation and evaluation on a resource-constrained and intermittently powered CRFID device (**Problem 1**). Demonstration video of the firmware update process is available at:

https://www.youtube.com/watch?v=nWcwGLs_jJK0



scan to watch

⁴The security provided by the wireless firmware update scheme prevents an attacker from injecting malicious code. We achieve this aim by establishing the *integrity* and *authenticity* of a firmware. To reduce the burden on a resource-limited device, we do not consider the provision of confidentiality protection.

⁵Specifically, we refer to the security of the fuzzy extractor scheme expressed as the complexity of recovering the derived key by an attacker. Eventually, the security (authenticity and integrity) of the firmware update scheme rests on the security of the key derived from the reverse fuzzy extractor method.

3.2 SecuCode: Protocol Design

- **A standards compliant method.** SecuCode is fully standards compliant: i) we implement SecuCode over the EPC C1G2v2 air interface protocol—ISO/IEC 18000-63:2015—commonly used by modern Radio Frequency Identification technology, including CRFID devices; and ii) given the specific Texas Instruments (TI) ultra-low power micro controller employed by CRFID devices, the bootloader implemented in this chapter is based on TI’s recent framework for wireless firmware updates—*MSP430FRBoot* [141]—to ensure a standard tool chain for compilation and update of new firmware whilst taking advantage of the features supported by MSP430FRBoot (**Problem 2**).
- **Public code and SRAM PUF dataset release.** We provide the complete end-to-end solution, including the SecuCode App source code, and research data collected on 20 devices (contributed as the MSP20, detailed in Section 2.3.4) to support future research in the field. The links of released source code and dataset are available in Section 1.4.

3.1.1 Chapter Overview

Section 3.2 presents the SecuCode protocol. Section 3.3 details the building blocks required to realise SecuCode and their instantiation on a CRFID device. Section 3.4 performs extensive experiments, including an end-to-end SecuCode implementation on a CRFID token as well as analyse its security. Section 3.5 discusses related work. Section 3.6 is dedicated for our collaborator who made remarkable contribution to this chapter. We conclude this chapter in Section 3.7.

3.1.2 Notations and Concepts

Adding to the general notations and conventions defined in Section 2.1 in Chapter 2, Table 3.1 summarises some key concepts introduced and referred to in this chapter.

3.2 SecuCode: Protocol Design

We ease into the design of the protocol by a formal description of the SecuCode protocol.

Table 3.1: Table of notations in this chapter.

\mathcal{P}	The prover \mathcal{P} is a single entity consisting of a host computer and a networked RFID reader.
\mathcal{T}	Token \mathcal{T} is a CRFID device.
\mathcal{A}	The adversary \mathcal{A} is an attacker seek to commit malicious code injection into the \mathcal{T} .
DB	Server's database, where each element is a two-tuple describing each CRFID token: i) the unique and immutable identification number id_i ; and ii) enrolled challenge response pairs $(\mathbf{c}_1, \mathbf{r}'_1), \dots, (\mathbf{c}_m, \mathbf{r}'_m)$.
firmware	The new firmware binary for the update.
firmware ^(j)	The j_{th} block of the firmware .
k	The device-specific PUF key (for more details, please refer to Section 3.2.2).
nonce	The nonce is a randomly generated number that to be used only once in secure communication to prevent replay attacks.
s	A MAC tag computed over the firmware and a nonce by a token \mathcal{T} (for more details, please refer to Section 3.2.2).
seq	The sequence number seq labels each block of the firmware , to store the received firmware block into the correct address of the code memory.
\square'	An apostrophe denotes a value computed by a different entity, e.g., s' the MAC tag computed by the prover \mathcal{P} .
\square_i	A subscript i denotes a specific entity out of a collection, e.g., \mathbf{c}_i is the i_{th} PUF challenge.
RNG()	The random number generator RNG() outputs a truly random number when invoked. (for more details, please refer to Section 3.3.2)
MAC()	MAC() computes a bit string s of fixed length to establish the authenticity and the integrity of a message (for more details, please refer to Section 3.3.4).
TEMP()	The temperature measurement function TEMP() sets an over-temperature flag (OTF) if the in-built chip thermometer reports a temperature outside of a legal range (for more details, please refer to Section 3.2.2).
PUF()	The SRAM PUF takes a challenge \mathbf{c} (memory address) as input and reacts with a corresponding response \mathbf{r} (SRAM start-up value) as output, where $\mathbf{r} \leftarrow \text{PUF}(\mathbf{c})$ (for more details, please refer to Section 3.2.2).
FE. \square ()	A fuzzy extractor FE is a noise compensation or error correction utility defined by two functions: key generation algorithm FE.Gen() and key reconstruction algorithm FE.Rep(). For more details please refer to Section 2.3.3.

3.2.1 Adversary Model

This chapter focuses the communication between the Reader and the CRFID transponder or Token \mathcal{T} as depicted in Figure 2.1 in Chapter 2. We assume that the communication between a Host and a Reader is secure using standard cryptographic mechanisms for securing communication between two parties over a network [89]; hence a Host computer and a Reader are considered as a single entity, the Prover, denoted as \mathcal{P} .

There is no previous CRFID firmware update protocol that considers security. Therefore, no existing adversary model has been reasoned. In this initial secure wireless firmware update investigation, we follow a relevant model and assumptions in PUF-based authentication protocols designed for resource-constrained platforms [62], [66].

Notably, a wireless firmware update of a CRFID device is only possible after: i) the commissioning of the device whereby an immutable program called the *bootloader* is installed

3.2 SecuCode: Protocol Design

on the device; and ii) the prover \mathcal{P} has enrolled—extraction and secure storage of—SRAM PUF responses in a secure environment using a one-time access wired interface. We assume that the wired interface is disabled after the installation of the bootloader and enrolment of the PUF responses. In other words, the adversary \mathcal{A} cannot directly access the SRAM PUF responses, only the immutable bootloader maintains this access at power-up and for a very short duration of time. After the commissioning of the device, both a trusted party and the adversary \mathcal{A} must use the wireless interface for installing new firmware on a token.

Subsequent deployment of a token \mathcal{T} will place it in an adversarial environment where only the prover \mathcal{P} remains trusted. We assume that the adversary \mathcal{A} can eavesdrop on the communication channel, isolate the CRFID transponder from the system and carry out a man-in-the-middle attack and forward tampered information from \mathcal{P} to \mathcal{T} and vice versa. Further, following the assumptions in [62], within the adversarial environment, the adversary \mathcal{A} may obtain any data stored in the NVM of the devices. However, as in [62], the adversary \mathcal{A} cannot mount implementation attacks against the CRFID, nor gain internal variables stored in the registers, for example, using invasive attacks and side-channel analysis. Similar to other adversarial models, we do not consider Denial of Service (DoS) attacks because, in practice, it is not possible to defend against an adversary that, for example, disrupts or jams the wireless communication medium [142], or attempts to heat the intrinsic SRAM cells to prevent a firmware update as a result of the proposed conditional firmware update method (detailed in Section 3.3.3).

3.2.2 SecuCode Protocol

SecuCode protocol described in Figure 3.2 relies on the simplicity of transmitting the firmware in plaintext and assumes the adversary \mathcal{A} can gain full knowledge of the firmware—this is consistent with the adversary model in this chapter which assumes that an adversary can read the contents of the NVM of a CRFID device. This chapter focuses to prevent malicious code injection attacks. SecuCode achieves this goal by facilitating the authentication of the prover by the token and ensuring the integrity of the firmware by the token before accepting the firmware. Therefore, only firmware issued by the trusted prover will be accepted.

The SecuCode protocol is designed to be implemented over the *EPC Gen2* protocol. We employ the recently defined extended *Access* command features for supporting future security services on RFID transponders. Consequently, firmware update initialisation employs `TagPrivilege` and `Authenticate` while downstream data transmissions in SecuCode are carried out by employing the *EPC Gen2* protocol specification of `BlockWrite` and `SecureComm` commands. The `SecureComm` command specification allows the encapsulation of other *EPC*

Gen2 protocol commands, such as `BlockWrite` but the payload is encrypted. Hence, we employ the `SecureComm` command to transport the message authentication code to the token. We employ `BlockWrite` command to write firmware to a memory space, or download area, allocated and managed by the bootloader on a CRFID transponder. Although the specification of `TagPrivilege`, `Authenticate` and `SecureComm` commands are defined in the *EPC Gen2* protocol, it is important to mention here that these commands *are yet to be supported* on CRFID transponders and this also constitutes one of the tasks in this study. The key phases in the proposed `SecuCode` protocol are summarised below:

- **Prover initialisation phase in a secure environment:** This phase is carried out in a secure environment. A publicly known unique identification string **id** is stored in a token's NVM. The prover \mathcal{P} enrolls in a database **DB** the **id** of the target token \mathcal{T} as well as the challenge-response pairs (CRP) from the PUF (also known as the enrolment phase [59]). The bootloader, immutable program stored in a write protected memory space, is installed on the token \mathcal{T} by the prover \mathcal{P} and, subsequently, the physical interface to \mathcal{T} is disabled. The bootloader is responsible for the `SecuCode` protocol implementation on the token.
- **Firmware update phase in a potential adversarial environment:** For each code dissemination session, there will be a compiled firmware at the prover \mathcal{P} to be transmitted along with a setup profile which describes the size of the firmware, starting memory address and the MAC method for the token \mathcal{T} . In particular, the following occurs: i) **lightweight physically obfuscated key derivation** on the token and the subsequent transmission of the token generated random challenge seed **c** and helper data **h** to the prover; ii) **firmware update** which includes the wireless transfer of firmware to the token, the establishment of the veracity of the firmware on the token to accept/abort the firmware update issued by the prover and update of firmware on the token. We elaborate on these stages below.

Lightweight Physically Obfuscated Key Derivation

After the token \mathcal{T} harvests adequate power from the prover \mathcal{P} , a **nonce** is generated for use in the firmware update session, meanwhile a random number \mathbf{c}_i is generated as the seed challenge; \mathbf{c}_i can be viewed as a challenge seed that determines the starting index into a byte level address in a block of highly reliable and unbiased SRAM PUF cells; for a detailed discussion please refer to Section 3.3.3. These responses are subsequently readout; $\mathbf{r}_i \leftarrow \text{PUF}(\mathbf{c}_i)$.

We propose a *conditional firmware update* method based on evaluating the on-chip temperature prior to the key derivation phase since response reliability of SRAM PUFs are more sensitive

to changes in temperature than supply voltage. This is to significantly reduce the computational burden on the PUF key generation overhead while meeting error correction and security bounds—details in Section 3.3.3. Immediately before key derivation, the `TEMP()` function sets an over-temperature flag (OTF) if the in-built chip thermometer reports a temperature outside of a legal range (0°C to 40°C is the chosen legal range in this chapter). Setting OTF will result in aborting key derivation and triggering a re-booting of the token.

A token \mathcal{T} operating under a legal temperature range will execute the private key \mathbf{k}_i derivation and helper data \mathbf{h}_i generation as $(\mathbf{k}_i, \mathbf{h}_i) \leftarrow \text{FE.Gen}(\mathbf{r}_i)$. The private key \mathbf{k}_i is used in the following firmware update process and only retained in the SRAM on the token \mathcal{T} for the duration of the protocol session and discarded: i) at the completion of a session; or ii) during a power loss event.

Firmware Update

The **id** of the token \mathcal{T} is checked by the prover \mathcal{P} to select the target token and once the target is confirmed to be visible to the prover and is singulated, the prover \mathcal{P} can employ *Access* commands `Authenticate`, `BlockWrite` and `SecureComm`—see Figure 2.1 for an illustration of the *EPC Gen2* protocol behaviour—to execute the firmware update. We describe the update phase below.

The prover issues an `Authenticate` command to deliver setup parameters. The token responds with the **nonce**, \mathbf{c}_i and \mathbf{h}_i back to the prover. The prover \mathcal{P} reconstructs the private key \mathbf{k}_i through $\mathbf{k}_i \leftarrow \text{FE.Rep}(\mathbf{r}'_i, \mathbf{h}_i)$; here, \mathbf{r}'_i is the enrolled response corresponding to \mathbf{c}_i . Now the prover \mathcal{P} and the token \mathcal{T} have a shared private key.

The firmware cannot generally be sent to the token \mathcal{T} in a single transaction. The *EPC Gen2* protocol implies a limitation on the length of a payload string to 255 words; this can be inadequate to encapsulate a practical CRFID firmware[29]. Therefore, we partition the firmware input into n chunks $\{\mathbf{firmware}^{(0)}, \mathbf{firmware}^{(1)}, \mathbf{firmware}^{(2)}, \dots, \mathbf{firmware}^{(n)}\}$ and transmit sequentially indexed chunks $\{\mathbf{seq}_0, \mathbf{seq}_1, \dots, \mathbf{seq}_n\}$ to the token \mathcal{T} using the `BlockWrite` command. Here, \mathbf{seq}_i indicates the relative offset of firmware chunk $\mathbf{firmware}^{(i)}$.

Before the firmware update is applied, the token \mathcal{T} must validate the authenticity of the prover \mathcal{P} and the integrity of the firmware. First, a message authentication code s' is computed by the prover \mathcal{P} using a `MAC()` function as $s' \leftarrow \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware} \parallel \mathbf{nonce})$, with \mathbf{k}_i the reconstructed PUF key. Second, the prover sends s' to the token. The token computes $s = \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware} \parallel \mathbf{nonce})$ locally to compare s and s' . If s and s' match, the token \mathcal{T} accepts and applies the firmware update. Success of a firmware update is signaled to the prover \mathcal{P} by the token backscattering an `ACK`. This process ensures:

1. The integrity of the received firmware at the token \mathcal{T} . Any corruption or mutation will lead to the failure of the integrity check on the token owing to the $\text{MAC}()$ and subsequent discarding of the firmware.
2. The authority of the prover \mathcal{P} . Only the trusted prover can obtain same secret key \mathbf{k}_i to issue a valid MAC tag \mathbf{s}' .

3.3 Implementation

In this section:

1. We generalise the SecuCode control flow on a token \mathcal{T} and provide an overview of the required functional blocks.
2. We describe the challenges in instantiating the functional blocks and propose approaches to address them.
3. We complete the end-to-end implementation based on the instantiated functional blocks.

We have selected the open-hardware and software implementation of WISP5.1-LRG [143] CRFID transponder as our token \mathcal{T} for a concrete implementation and experiments. This intermittently powered CRFID transponder is built using the ultra-low power microcontroller unit (MCU) MSP430FR5969 from Texas Instruments. Therefore, when required, we provide specific implementation examples on a CRFID transponder based on the WISP5.1-LRG and MSP430FR5969 MCU in the discussions that follow.

3.3.1 Protocol Control Flow on a Transponder

Following the SecuCode protocol in Section 3.2.2, the generalised control flow on a transponder is illustrated in Figure 3.1 and detailed below.

- ① When a token \mathcal{T} is adequately powered, it initialises the MCU hardware, and determines whether to run in firmware update mode or application execution mode. If OTF is not set (the token is operating in a legal temperature range), the token enters the key derivation phase and
- ② a challenge \mathbf{c}_i and a **nonce** are generated using the RNG.
- ③ The token \mathcal{T} reads the response \mathbf{r}_i corresponding to \mathbf{c}_i .
- ④ The token \mathcal{T} derives a key \mathbf{k}_i and computes helper data \mathbf{h}_i through

3.3 Implementation

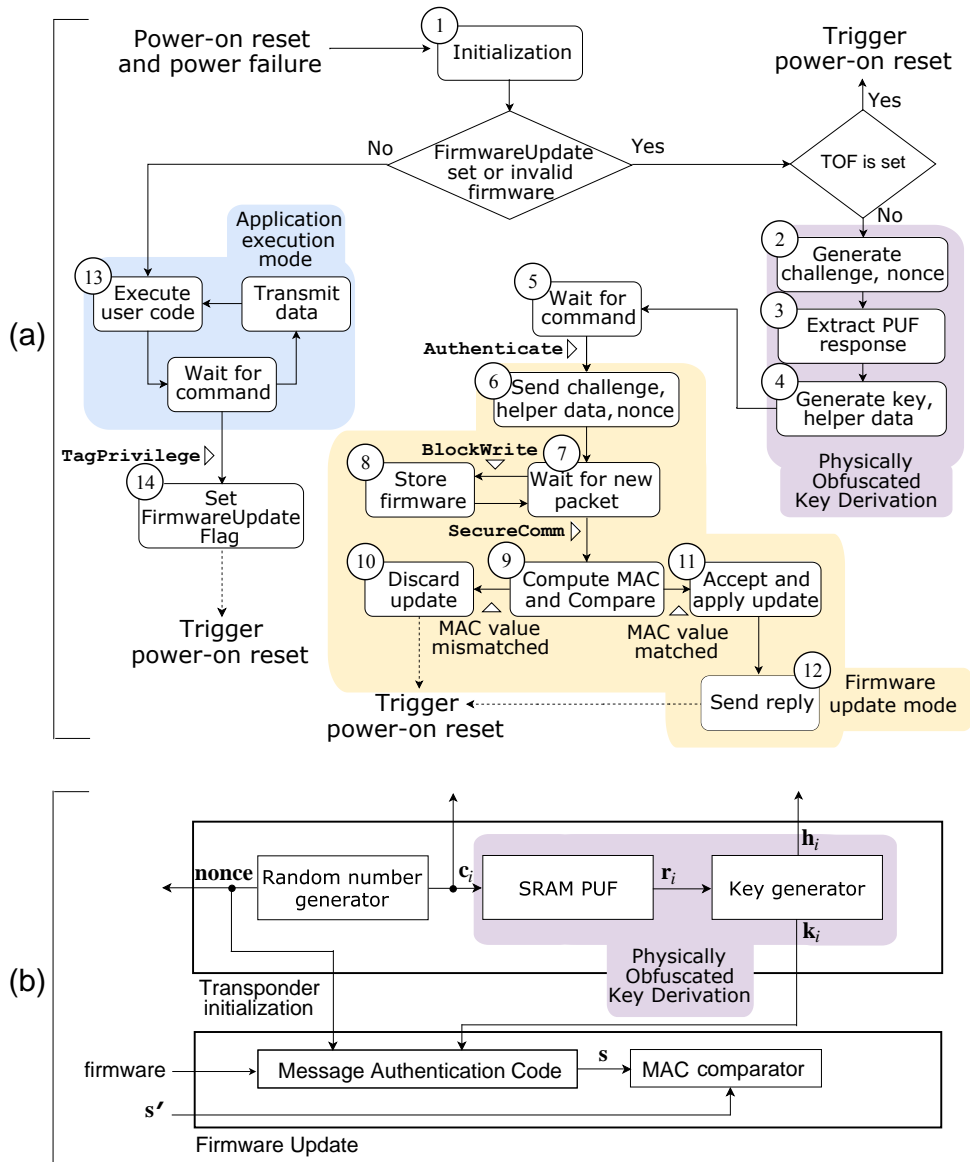


Figure 3.1: (a) Control flow and (b) Protocol functional blocks implemented to instantiate SecuCode protocol on a CRFID transponder.

$(\mathbf{k}_i, \mathbf{h}_i) \leftarrow \text{FE.Gen}(\mathbf{r}_i)$. These steps complete the key derivation phase and ⑤ the token \mathcal{T} awaits for further commands from the prover \mathcal{P} .

⑥ The token \mathcal{T} responds to an `Authenticate` command consisting of the firmware update setup parameters with \mathbf{h}_i , \mathbf{c}_i , and `nonce` to enable the prover \mathcal{P} to reconstruct the PUF key \mathbf{k}_i . The `Authenticate` command also directs the token \mathcal{T} to enter the **Firmware Update** mode.

⑦ Whilst in this state, the prover \mathcal{P} can transmit new firmware in chunks.

⑧ Given a firmware chunk, `firmware(i)`, it is transmitted using `BlockWrite` commands and stored in the download area in a receive and store process. At the completion of the

wireless firmware transmission, the MAC tag s' computed by the prover is encapsulated in a `SecureComm` command and sent to the token \mathcal{T} .

⑨ The MAC value s computed by the token using the received **firmware** and **nonce** with the secret key k_i is compared; if the integrity of the firmware and authenticity of the prover is established, ⑪ the **firmware** is accepted and ⑫ the token sends an ACK to the prover. Otherwise, ⑩ the update is discarded. Regardless of the acceptance or rejection decision, the token exits the **Firmware Update** mode by triggering a power-on-reset.

⑬ If a firmware update is not required, the token \mathcal{T} executes the user code, ⑭ if the token receives a `TagPrivilege` command, indicating entry into firmware update mode, the token restarts in firmware update mode by setting a firmware update flag—*FirmwareUpdate*—and triggering a power-on-reset. Whenever the token \mathcal{T} is rebooted by a `TagPrivilege` command to enter the **Firmware Update** mode, **nonce** and derived key k_i is refreshed. The k_i changes because: i) the challenge seed is refreshed; and ii) a varying response is produced even for the same challenge as a consequence of the naturally noisy nature of the response bits.

A power failure such as a brownout event, as shown in Figure 1.3 in Chapter 1, during the execution of the protocol will result in a reset and rebooting of the token \mathcal{T} . In such an event, the immutable bootloader's functionality is preserved. Therefore, a prover \mathcal{P} can attempt another secure firmware update. In the following sections, we describe the instantiation of each functional block shown in Figure 3.1(b).

3.3.2 Random Number Generator

The implementation of the SecuCode protocol is shown in Figure 3.2 employs an 8-bit challenge seed and a 128-bit **nonce**. As with other low-end computing platforms, acquiring true random numbers on a CRFID device is challenging given the lack of resources to implement a cryptographically secure random number generator (RNG). Ideally, the RNG should be a true random number generator (TRNG) and its implementation should require no modifications to existing hardware. We evaluate and summarise the performance of three RNGs in Table 3.2 in terms of random bits per request, time overhead, power consumption and required hardware.

We provide a summary of the random number generator methods we have evaluated.

CTR-DRBG. Texas Instrument has demonstrated a Counter Mode Deterministic Random Byte Generator (CTR-DRBG) in [144] to meet the needs of security mechanisms on MSP430 MCUs. The CTR-DRBG is developed by National Institute of Standard and Technology

3.3 Implementation

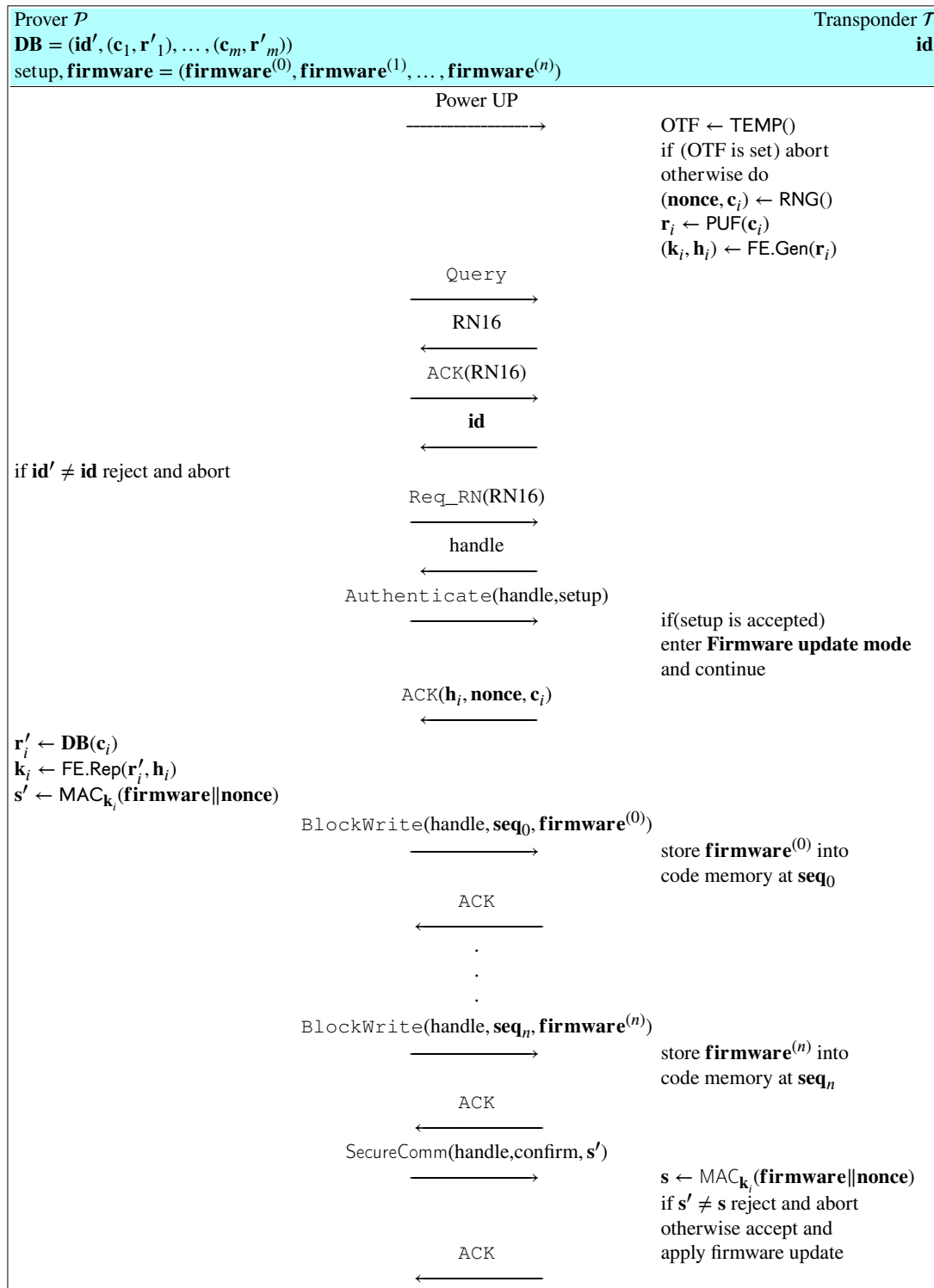


Figure 3.2: SecuCode protocol.

(NIST) in NIST SP 800-90A [145] and is built upon a block cipher algorithm (e.g., AES256). The CTR-DRBG passes all NIST randomness test criteria; implying that the CTR-DRBG has comparable performance to a true random number generator.

Table 3.2: Random Number Generator Comparison.

Name	Bits per request	Time overhead	Energy consumption	Hardware requirement
CTR-DRBG	128	214 us	14.83 nJ	AES ,FRAM
Thermal noise	16	27.2 us	2.69 nJ	ADC
SRAM-TRNG	128	23.10 us	2.32 nJ	–

Note: the calculations are based on the technical details obtained from MSP430FR5969 datasheet for a 1 MHz MCLK and 3.3 V power supply.

Thermal noise. Thermal noise (Johnson–Nyquist noise) has been exploited [146][147][148] as an entropy source for a TRNG. Thermal noise results in unpredictable small voltage fluctuations in resistive components at any temperature above 0°K. The least significant bit (LSB) of sampled data is significantly affected by thermal noise over other bits[149][150]. Therefore, random bits can be extracted by sampling the LSB from a noisy sensor or measuring signal propagation variations [69]. Random number generators from this method has passed the NIST randomness tests [69], [151].

Noisy SRAM responses. The noisy SRAM PUF responses can be used to generate true random numbers; this method has been extensively studied, for example in [50], [62]. Sufficient entropy can be extracted through, e.g., exclusive-or and bit shift operations[50], [62] over a number of SRAM PUF responses. Random bit streams from this method has passed the NIST random number generator test suite [50].

We can see that the SRAM TRNG, implementation based on the study in [62], outperforms the rest with regards to time and energy overhead. Most importantly, it requires *no extra hardware*. Therefore, an SRAM TRNG is chosen for implementing the SecuCode protocol.

3.3.3 Lightweight Physically Obfuscated Key Derivation

Deriving and sharing a private key between the prover \mathcal{P} and the token \mathcal{T} should also be: i) lightweight; and ii) secure. Realising both requirements on a resource-limited token is challenging. Thus we: i) employ an SRAM PUF to derive a key instead of a stored key in non-volatile memory—prone to extraction through physical means and requiring expensive secure NVM—with the ability to refresh the key between protocol sessions; and ii) employ a reverse fuzzy extractor to realise a lightweight FE.Gen() on the token to compute helper data necessary for the server to independently reconstruct the shared key with very high probability of success. In particular, we propose the following mechanisms to derive a lightweight and secure PUF key:

3.3 Implementation

- **Enhancing reliability of key material.** Time complexity of the generator function $\text{FE.Gen}()$ and security—information leakage—is related to the amount of helper data needed to correct noisy SRAM PUF response bits. Therefore, we extract SRAM PUF responses with high bit specific reliability using *response pre-selection with an on-chip selection meta-data storage structure* together with the proposed *conditional firmware update* method to significantly reduce the demand on helper data (see Section 3.3.3).
- **Removing information leakage through response bias.** PUF response bias, an imbalance between the number of zeros and ones, has shown to leak additional information [73], [152]. Therefore, to guarantee the security bounds of the reverse fuzzy extractor, we propose a Hamming weight-based response de-biasing method to eliminate response bias (see Section 3.3.3).

Enhancing Reliability

We consider a syndrome-based construction as in [66] and use a $\text{BCH}(n, k, t)$ linear block code encoder to build $\text{FE.Gen}()$. The $\text{FE.Gen}()$ function is responsible for generating the helper data \mathbf{h} used by the prover to reconstruct the PUF response extracted by the token using a previously enrolled response that is securely stored at the prover. Here, t denotes the number of errors a $\text{BCH}(n, k, t)$ code is capable of correcting, n denotes the number of bits extracted from a PUF or the length of the response \mathbf{r}_i where the length of the helper data is $|\mathbf{h}| = (n - k)$ — which also defines the well known upper bound on information leakage.

Although we can select a $\text{BCH}(n, k, t)$ code with appropriately large parameter values to achieve the desired attack complexity, error correcting capability to achieve an industry standard key failure rate of less than 10^{-6} and number of key bits k , the computational time complexity of a BCH encoder, $\mathcal{O}(n^2)$, forces the use of parallel blocks of $\text{BCH}(n, k, t)$ code with smaller values of n . For $|\mathbf{r}_i|/n$ parallel blocks of $\text{BCH}(n, k, t)$ code using a syndrome construction, the complexity of finding \mathbf{r}_i is $2^{k|\mathbf{r}_i|/n}$ [73].

The key failure rate when employing a $\text{BCH}(n, k, t)$ code is expressed as:

$$P_1 = 1 - \text{binocdf}(t, n, \text{BER}) \quad (3.1)$$

where binocdf is the binomial cumulative distribution function, BER describes the response unreliability as introduced in Section 2.3.2.

As we use multiple blocks, the failure rate when $|r_i|/n$ blocks are employed is expressed as:

$$P_r^{\text{Fail}} = 1 - (1 - P_1)^{|r_i|/n} \quad (3.2)$$

Therefore, we can see that it is imperative to reduce BER to significantly decrease the key failure rate P_r^{Fail} and reduce the complexity of the BCH encoder required. We devise the following **two** methods to significantly reduce the complexity of the BCH encoder required on the resource-constrained token: i) response pre-selection together with on-chip selection meta-data storage structure; and ii) a conditional firmware update strategy. As shown by detailed experimental evaluations in Section 3.4.1, our approaches significantly reduce the expected BER to be less than 1%.

Pre-selection. SRAM PUF pre-selection was first noted by Hofer et al. in [132]. The idea is to locate SRAM cells which tend to generate stable PUF responses. During the enrolment phase, unstable responses are identified and discarded for key generation [153].

We employ an approach similar to the multiple-readout method in [153]. Given that a microcontroller's SRAM memory is byte addressable, we employ a byte-level response selection method illustrated in Figure 3.3. In particular, we first select response bytes that are reliably reproduced under repeated measurements generated under two corner temperatures—selected as 40 °C and 0 °C in our implementation. Subsequently, under nominal temperature (25 °C), the former response bytes are further subjected to multiple readouts and majority voting (e.g., if 8 out of 10 readouts of a bit yields logic '1', then this bit is enrolled as a logic '1') is applied enrol values for the response bytes.

Conditional firmware update. It is recognised that the BER of SRAM PUF is sensitive to temperature but insensitive to supply voltage variations [154]. Therefore, in addition to reliable response pre-selection, we propose performing a conditional firmware update based on the core operating temperature of the token. A firmware update is executed only when the core temperature of the chip is within a legal temperature range; considering most practical applications, we selected 0 °C to 40 °C range for SecuCode. Without extra hardware overhead, we take advantage of the available temperature sensor within the MSP430FR5969 microcontroller used by the CRFID transponder to sample the temperature before the PUF responses are readout. A temperature breaching the legal range terminates further execution and triggers a power-on-reset.

3.3 Implementation

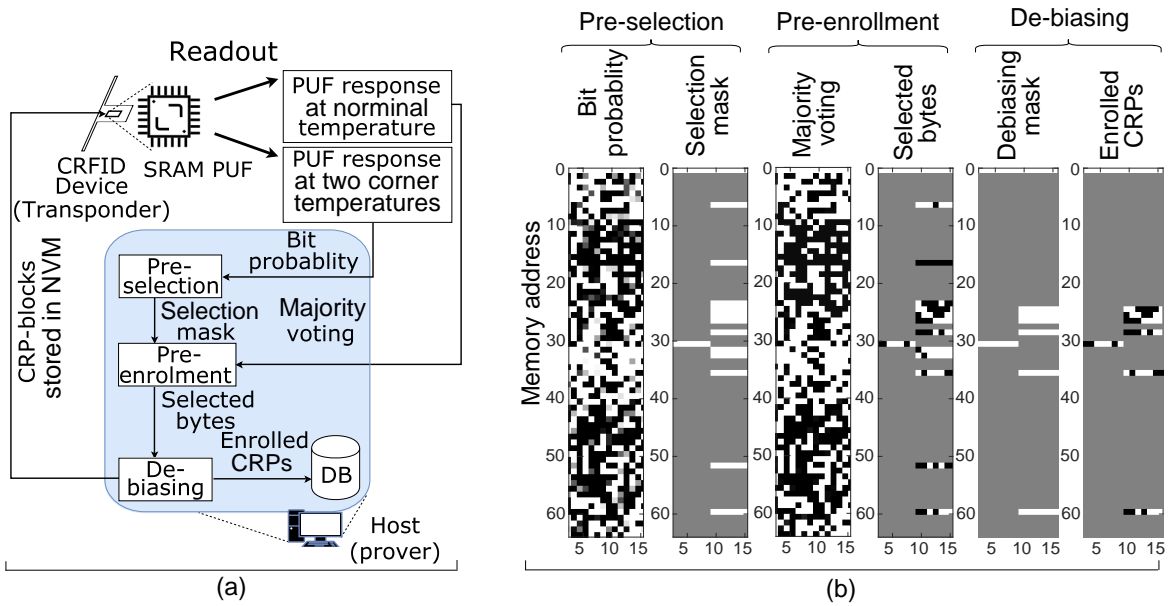


Figure 3.3: (a) Procedure for pre-selection, enrolment and de-biasing to enhance the response reliability and generate unbiased PUF response, and (b) Memory maps detailing each step—only the first 64 words are shown here.

Proposed De-biasing Scheme

Most studies assume that PUF responses are uniformly distributed and hence an n -bit response has the fully entropy of n bits. Using a $BCH(n, k, t)$ code, $(n - k)$ -bit helper data \mathbf{h} will be publicly known. Under the disclosure of helper data \mathbf{h} , there is no less than k -bit entropy remaining. However, as highlighted recently [152], [155], response bias incurs extra entropy loss. Thus, the k -bit min entropy bound guaranteed by a $BCH(n, k, t)$ encoder might be decreased. To prevent extra entropy leakage, response de-biasing methods can be employed [155], [156]. In general, de-biasing converts a response \mathbf{r}_x into an unbiased enrolled response \mathbf{r}_y , where response \mathbf{r}_x might have a bias and $|\mathbf{r}_x| \geq |\mathbf{r}_y|$. De-biasing needs to consider four aspects: i) *de-biasing should not deteriorate or increase response error rate*; ii) *efficiency*; iii) *information leakage*; and iv) *reusability* [152], [155]; to this end, we propose the following Hamming weight (HW)-based de-biasing method.

HW-based De-biasing. In our application scenario, there are three design specific requirements: i) minimise the de-biasing computational overhead on the token \mathcal{T} ; ii) avoid any additional data transmission overhead between the prover \mathcal{P} and the token \mathcal{T} ; and iii) reduce additional de-biasing information stored on the token \mathcal{T} . We discuss how we achieve the first two goals in this section and explain how we achieve the third goal in Section 3.3.3.

To achieve the first two goals, we offload the computational burden to the prover \mathcal{P} by performing a one-time de-biasing during the enrolment phase. In the Pre-selection stage described in Section 3.3.3, stable SRAM PUF bytes are identified. We further refine HW balanced bytes from those stable bytes in our one-time de-biasing process. In general, we determine the address of bytes which are not only reliable but also HW balanced; HW close to 0.5. This approach leads to PUF responses—a linear combination of de-biased stable bytes—to be HW balanced. The HW-based process applied post pre-selection is shown in Figure 3.3.

Remark. The proposed HW-based scheme does not deteriorate reliability of PUF responses, eschews leakage from potential bias and enables re-usability. As a trade-off, efficiency is decreased as the $\frac{|r_x|}{|r_y|}$ is low as demonstrated by our experimental results in Section 3.4.1. However, efficiency is not a concern in our implementation of SecuCode as there are always sufficient SRAM responses while we only need to use a small fraction of them. In [155], due to the de-biasing method—i.e., classic von Neumann de-biasing (CVN)—the token \mathcal{T} generates different de-biasing data for each re-evaluation of the response r_x . Thus, multiple observation of the de-biasing data given the re-generation of r_x leads to extra entropy loss. Therefore, the key generator based on such a de-biasing method is not reusable unless further optimisation, e.g., pair-output VN de-biasing with erasures (ϵ -2O-VN), is implemented to prevent extra entropy loss from the multiple production of the de-biasing data. In contrast, in our HW-based de-biasing method, the de-biasing data is generated only *once* during the enrolment phase, we do not generate multiple de-biasing data and thus incurs no further entropy leakage from the de-biasing data. This implies that our HW-based de-biasing is reusable.

On-chip meta-data storage structure

It is inefficient to store the absolute addresses of winnowed bytes post pre-selection and de-biasing process, described in Figure 3.4, on the CRFID device and to subsequently perform an exhaustive search to find them on demand. We propose the data storage structure named *CRP-block map* to refer to those reliable and HW balanced SRAM PUF responses.

The CRP-block map data structure is illustrated in Figure 3.4(c). The CRP-block data structure divides the SRAM memory into multiple blocks. Each block has an integer index such as Block 0, Block 1, and Block 2. The size of each block may vary, however, each block is designed to contain an equal number of reliable and HW balanced bytes required for the physically obfuscated key derivation mechanism. To map a CRP-block into a physical memory address, each block has an absolute starting address and several offsets pointing to the winnowed bytes. The starting address of each block and the offsets are stored in a look up table LUT in a token's

3.3 Implementation

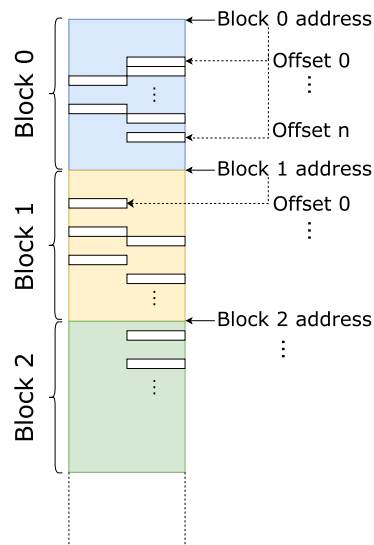


Figure 3.4: CRP-block map, a compact data structure to store the mask configuration on the CRFID transponder for fast response readout.

NVM indexable via block number. To produce a PUF key, the $RNG()$ generates a seed challenge that is a block number. Consequently, a look up of a block number will resolve the targeted block's starting memory address and the offsets point to the response bytes. Hence a random challenge c_i can be used to select a CRP-block and, subsequently, the power-up states of the bytes in the selected CRP-block are readout and concatenated as the response r_i that is both highly reliable and unbiased.

3.3.4 Message Authentication Code

SecuCode requires one cryptographic primitives: a keyed hash function to build a message authentication code (MAC) to realise the $MAC()$ function. The two dominant factors determining their selection are:

1. The CPU clock cycles required for execution. For example, from the figure Figure 1.4 we have seen in Chapter 1 that at a 50 cm distance, we may not expect more than 500,000 to 600,000 clock cycles before harvested power is exhausted.
2. Memory (RAM) budget. Since available on-chip memory is shared by the RFID communication stack, sensor data, user code, and SecuCode, the state space available for a cipher execution is limited.

Therefore, power and memory efficient primitives are highly desirable. The construction of a *secure*, computation and power efficient message authentication code MAC() on the token is extremely challenging as the entropy compression task is resource intensive and attacks, such as birthday attacks, demand that we use a MAC function with a large enough output size.

Given the lack of MAC benchmark data for MSP430 microcontroller series, we selected and implemented a set of existing secure keyed hash functions expected to yield a computationally efficient software implementation. Notably, benchmarks of software implementations of MAC function on desktop platforms are not suitable since these implementations use specific CPU instructions, such as the Streaming SIMD Extensions (SSE) instruction set, where SIMD stands for single instruction, multiple data [157], and advanced paradigms like out-of-order execution, which are not supported on resource-constrained embedded systems. We implemented and evaluated: BLAKE2s-256, BLAKE2s-128, and hardware-AES (HWAES) functions HWAES-GMAC and HWAES-CMAC for comparison; here HWAES functions benefited from the AES hardware accelerator module on the MSP430 MCU.

The MAC function tests were based on two WISP firmwares; i) LED firmware (short string); and ii) 3-axis accelerometer firmware (long string). Table 3.3, and 3.4 detail our results. We selected HWAES-CMAC to obtain a 128 bit MAC in our SecuCode implementation.

Table 3.3: MAC evaluation with LED firmware size = 153 bytes.

MAC	Digest size (bits)	Byte in digest	Clock Cycles	Cycle per message byte	Code size (bytes)	Internal state size (bytes)
BLAKE2s-256	256	32	81,036	530	4,964	238
BLAKE2s-128	128	16	79,276	518	4,961	238
HWAES-GMAC	128	16	432,337	2,826	3,538	268
HWAES-CMAC	128	16	15,876	104	3,198	58

Table 3.4: MAC evaluation of Accelerometer firmware size = 419 bytes.

MAC	Digest size (bits)	Byte in digest	Clock Cycles	Cycle per message byte	Code size (bytes)	Internal state size (bytes)
BLAKE2s-256	256	32	183,230	437	4,964	238
BLAKE2s-128	128	16	181,470	433	4,961	238
HWAES-GMAC	128	16	1,106,538	2,641	3,538	268
HWAES-CMAC	128	16	37,041	88	3,198	58

Based on results above, we selected a 128 bit MAC using *HWAES-CMAC*, Cipher-based Message Authentication Code ⁶ built using AES since it yields the lowest clock cycles per byte.

⁶We used the implementation detailed in NIST Special Publication 800-38B, *Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication*

3.3 Implementation

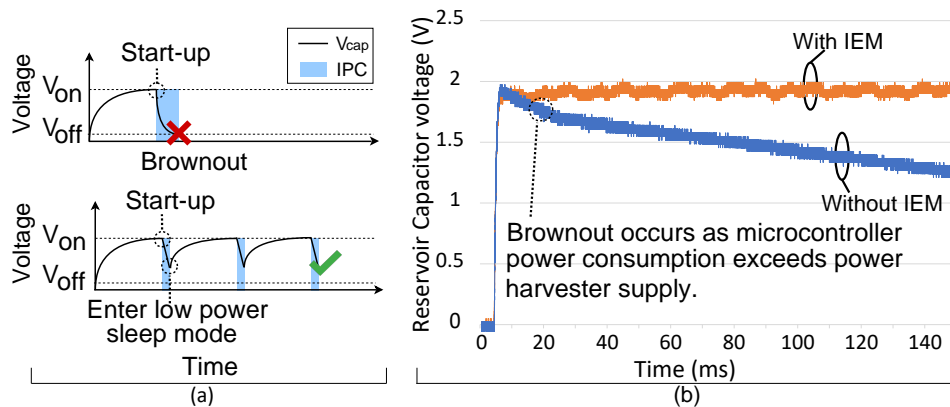


Figure 3.5: (a) Illustration of a brownout event and the proposed intermittent execution model (IEM) to prevent brownouts during computationally intensive operations (notations are given in Figure 1.3) and (b) Comparison of a PUF key derivation on a CRFID transponder at 50 cm from an RFID reader antenna with and without our proposed IEM.

3.3.5 Intermittent Execution Model

The nature of intermittently powered devices that rely on harvested power is typically described by a power harvesting and charging phase where energy is generally stored in a reservoir capacitor and then released for powering computations. We refer to this cycle as the intermittent power cycle (IPC). A brownout event can occur when the available energy in an IPC subceeds the energy needs of the computations, and the power harvester is unable to replenish energy as rapidly as it is consumed. A brownout results in state loss and termination of the execution thread as highlighted in Figure 1.3. Hence, as illustrated in Figure 3.5, we cannot always expect to continue a computation task to completion.

Studies such as Alpaca [158], Mementos [159], CCCP [160] and Clank [161] consider the problem of continuing the thread of execution over periods of power loss. In general, these studies employ checkpoint-based methods with various degrees of programmer support. The basic concept is to save state—checkpoint—and to restore state to a previously valid checkpoint to allow the resumption from a previous state of execution after a power loss event. With the exception of CCCP, state is saved in a device’s non-volatile memory; CCCP proposes the saving of state in an untrusted server. However, in SecuCode, process state such as the PUF key is intentionally volatile to eliminate the need to protect key storage. If power-loss or brownout does occur, all unfinished process state based on old key material should be discarded. Therefore, approaches to save and restore state from non-volatile sources are not desirable. Further, writing to NVM is energy intensive and we would like to avoid the additional overhead of checkpointing. In contrast, Dewdrop [162] considers execution under frequent power loss by attempting to *prevent* a brownout event by solving a task scheduling problem; execute tasks only when they

are likely to succeed by monitoring the available harvested power. Dewdrop provides an elegant dynamic scheduling method, however, requires the overhead of sampling the harvester voltage within the application code and task scheduling.

In order to deal to with frequent intermittent power loss, we consider the following intermittent execution model (IEM) for computation intensive building blocks of the protocol: `MAC()` and `FE.Gen()` functions. The IEM proposed in this chapter is built on the basic concept of a *task* in Alpaca [158]—a code block proportioned to execute to completion under a minimum number of available clock cycles or energy— and the concept in Dewdrop [162] attempting to *prevent* brownouts. We first identify the computationally intensive functions in the firmware, and construct an execution plan based on factoring the function to sub-tasks. These subtasks are then interleaved with low power sleep states in the code. We selected the lowest power consuming microcontroller sleep state from the target microcontroller such that the memory state is maintained during sleep. The duration of the imposed sleep state—termed the *intermittent operating setting*—is realised using an on-chip timer interrupt to wake up the transponder at the conclusion of a given intermittent operating setting; upon wake-up the transponder continues the execution of the following sub-task.

Immediate benefit of the IEM is the possible prevention of an intentional brownout event by allowing the replenishment of the reservoir capacitor energy by the power harvesting and charging circuitry as illustrated by the graph at the bottom of Figure 3.5(a). Figure 3.5(b), shows an experimental validation of of our IEM; here, we plot two captures of reservoir capacitor voltage with and without the IEM. The voltage capture traces show that the IEM restores power during low-power sleep states interleaved between sub-tasks—shown by the recovery of the voltage developed across the reservoir capacitor. In contrast, as shown by a falling reservoir capacitor voltage, the CRFID device without IEM fails due to brownout. Here, the CRFID device is unable to harvest adequate power to replenished the reservoir capacitor fast enough to provide the minimum operating voltage (1.8 V) necessary for the MCU.

In this chapter we hard-code the IEM setting. We further investigated the possibility of using the read-rate reported from the RFID reader to dynamically adjust the IEM setting to suit different powering channel conditions in Appendix A.

3.3 Implementation

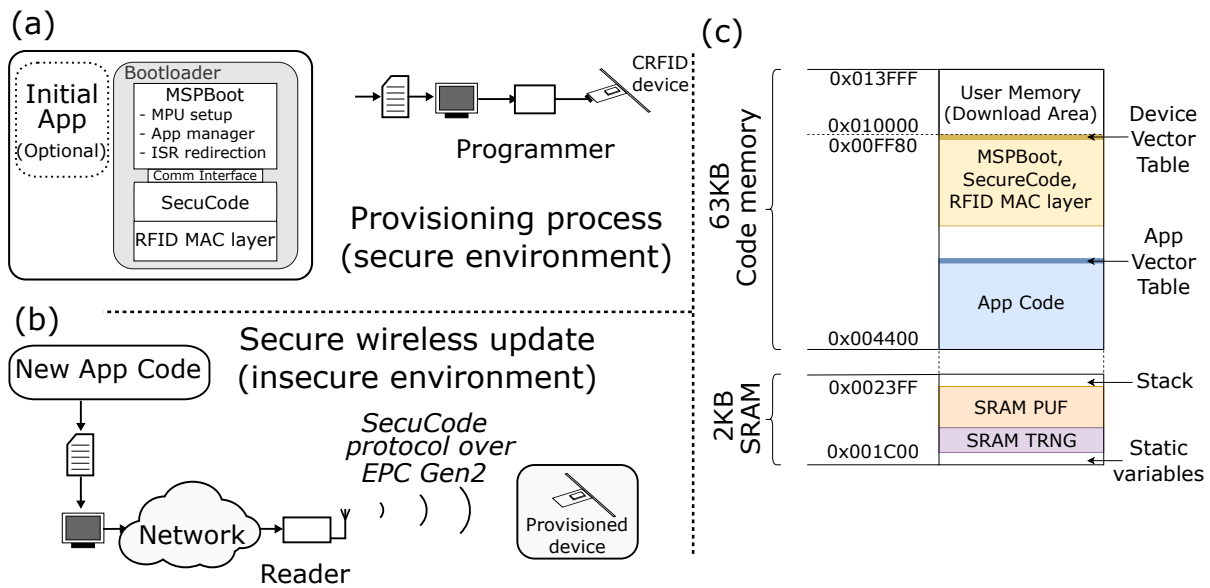


Figure 3.6: (a) Overview of bootloader provisioning; (b) Secure wireless firmware update and (c) Memory arrangement for the 2 KB SRAM and 64 KB FRAM.

3.3.6 End-to-end Implementation

The complete SecuCode-based firmware update process is illustrated in Figure 3.6. We describe below the memory arrangement, the development of the bootloader together with the complete tool chain to realise a standards compliant secure firmware update process.

Memory Arrangement. As shown in Figure 3.6 (c), the 2 KB SRAM embedded in the MSP430FR5969 MCU is divided into two sections. The lower address space is used for the SRAM TRNG and occupies 64 words of SRAM. The address space above the SRAM TRNG forms the SRAM PUF. The SRAM PUF and SRAM TRNG do not span the full space of the SRAM memory. This is necessary to allocate space for initialisation routines and PUF state variables. One hundred and sixty bytes of higher addresses are allocated as stack space, and 480 bytes from the lower address space is designated for static variables such as c_i , $nonce$ and h_i . The SRAM PUF and SRAM TRNG are only active during the **Physically Obfuscated Key Derivation** stage in Figure 3.1; once the response r_i is readout and k_i and h_i are generated, the SRAM memory is released for regular operations.

The CRFID device we employed has an embedded 64 KB Ferroelectric Random Access Memory (FRAM) as NVM. FRAM is partitioned into 64 KB code memory and 1 KB for Device Descriptor Info [163]. In this chapter, we only employ the 64 KB code memory space. The FRAM memory layout of the specific CRFID device we used is shown in Figure 3.6(c). FRAM

is divided into three sections: i) the bootloader; ii) Application code, and iii) User Memory (Download Area). The User Memory segment can be manipulated using `BlockWrite` commands, however only the bootloader can write to the Application Code memory space.

Bootloader. The SecuCode implementation for the CRFID device requires a bootloader to be provisioned onto the device. This is because the CRFID device has no supervisory control of an operating system and is designed in the manner of a low-end and low-cost device.

The bootloader developed in this chapter is based on TI's bootloader framework, MSPBoot [141]. The `Comm` interface in MSPBoot is designed to operate on trusted data, and therefore, should not directly communicate with the RFID MAC layer. Instead, the received firmware is buffered in a Download Area in memory. After the SecuCode protocol verifies the authenticity and the integrity of the firmware, it is passed to the MSPBoot via the `Comm` interface. In our implementation, we adopted the WISP5 firmware⁷ for the RFID MAC layer.

In order to realise an immutable bootloader, we considered the full memory protect mode (FMPM) and partial memory protect mode (PMPM) offered by the microcontroller together with setting an e-Fuse [164] to disable the potential for wired re-programming using the on-chip Joint Test Action Group (JTAG) interface after deployment in the field. In FMPM, the MPU (memory protection unit) is configured to prevent writing to the bootloader. Most importantly, the MPU is locked from being accessed. These actions are performed during the initialisation process, before execution of user code, and therefore, it is infeasible for the bootloader to be modified in FMPM mode. In PMPM, writing to the bootloader is still prevented, however, the MPU is not locked. Such an MPU configuration provides basic protection for the bootloader against programming errors, while still allowing the bootloader to be remotely updated. In this context, the firmware temporarily disables memory protection, overrides parts of the original code, then re-enables memory protection. We employ FMPM mode to realise an immutable bootloader.

Secure Firmware Update Process. The provisioning and the secure wireless update process is illustrated in Figure 3.6. A CRFID device is first provisioned whereby the immutable program called the **bootloader** is installed on the device in a secure environment. We assume the wired interface is disabled after the installation of the bootloader, and therefore, wireless code dissemination is the only practical mechanism by which to alter the firmware. The provisioned CRFID device is deployed in the field and can be subsequently updated with new firmware following the process below.

⁷<https://github.com/wisp>

3.4 Experimental Results and Analysis

- Compile the new firmware using a standard MSP430 compiler and pass the output, together with a linker map that specifies the memory allocation scheme shown in Figure 3.6, to an MSP430 linker to generate a binary file in ELF format⁸.
- Use our SecuCode App, available from [165], to load and parse the resulting ELF file to wrap the LOAD segments into MSPBoot commands specified in the MSPBoot framework [141]. The resulting binary is broken up into 128-bit blocks for transmission to a CRFID transponder.
- The SecuCode App subsequently uses LLRP commands to construct `AccessSpecs` and `ROSpecs`—refer to Section 2.2.2 for LLRP. These encodes *EPC Gen2* protocol commands employed by SecuCode such as `Authenticate`, `BlockWrite` and `SecuComm` commands⁹ to setup a networked RFID reader to execute the SecuCode protocol to wirelessly and securely update the firmware of the target device.

3.4 Experimental Results and Analysis

In this section, we describe the extensive set of experiments conducted to evaluate SecuCode. Here, we employed 20 MSP430FR5969 MCU chips (we release the data set as MSP20, more details please refer to Section 2.3.4) and a WIPS5.1LRG CRFID device built with the same microcontroller. The MSP430FR5969 microcontrollers consists of 2 KB (16,384 bits) of internal SRAM memory and 64 KB of internal FRAM memory. In our implementation we employed an 8-bit challenge c_i and a 128-bit **nonce**. We summarise our experiments below:

1. We evaluate and validate the response pre-selection and HW-based de-biasing methods, and subsequently, determine the BER of the CRP-block maps to determine the parameters n , k , t for the BCH encoder used to realise the `FE.Gen()` function on the CRFID transponder (Section 3.4.1).
2. Given that: i) security protocol implementation costs are rarely examined in the literature; and ii) the resource-constrained nature of the CRFID devices demand security, performance as well as practicability; we evaluate the memory requirements (code size in FRAM and state space in SRAM) and computing clock cycles for the security functional

⁸Alternatively, Texas Instrument's Code Composer Studio integrated development environment which bundles all the necessary tools can also be used to simplify the task

⁹Given that Impinj R420 RFID readers do not yet support the recent *EPC Gen2* protocol changes, we implement the unsupported commands using `BlockWrite` commands. The implementation of the employed commands are detailed in Appendix.C.1

blocks (those in addition to MSPBoot and RFID MAC Layer) necessary for SecuCode (Section 3.4.1 and Section 3.4.2).

3. We evaluate the impact of the most energy and computationally demanding SecuCode functional blocks—key derivation and MAC function—on performance and evaluate the effectiveness of our intermittent execution model on these energy intensive building blocks. (Section 3.4.3 and Section 3.4.3).
4. We present a case study to demonstrate the end-to-end implementation of SecuCode in an application scenario—the source code of our case examples are released on our project website to facilitate further research, experimentation and adoption of our SecuCode scheme (Section 3.4.3).
5. We evaluate the security of SecuCode under the adversary model detailed in Section 3.2.1 (Section 3.4.5).

3.4.1 Physically Obfuscated Key Derivation

In this section, we validate the lightweight physically obfuscated key derivation method using experimental data. Details about the evaluated MSP20 dataset can be found in Section 2.3.4 in Chapter 2.

SRAM PUF Reliability. Considering that SRAM PUFs are insensitive to voltage variations [67] and the voltage can be eventually controlled well in practice, we focus on its performance under differing temperature corners (-15°C , 0°C , 25°C , 40°C and 80°C). The evaluated MSP20 dataset has a average BER of 11% when referenced at 25°C and reevaluated at 80°C . The applied pre-selection reduces the BER significantly to no more than 0.94% over the defined legal temperature range of 0°C to 40°C —see Section 3.3.3.

SRAM PUF Bias. We follow the method described in Section 2.3.2 to evaluate the bias of our SRAM PUF. Ideally the bias distribution should close to 0.5, otherwise, more close to 0 or 1 suggests more bias.

After response pre-selection, we implemented our HW-based de-biasing method. As shown in Figure 3.7(d), experimental results from the 20 SRAM PUFs show that the mean of the bias is 0.499; very close to the ideal value of 0.5.

3.4 Experimental Results and Analysis

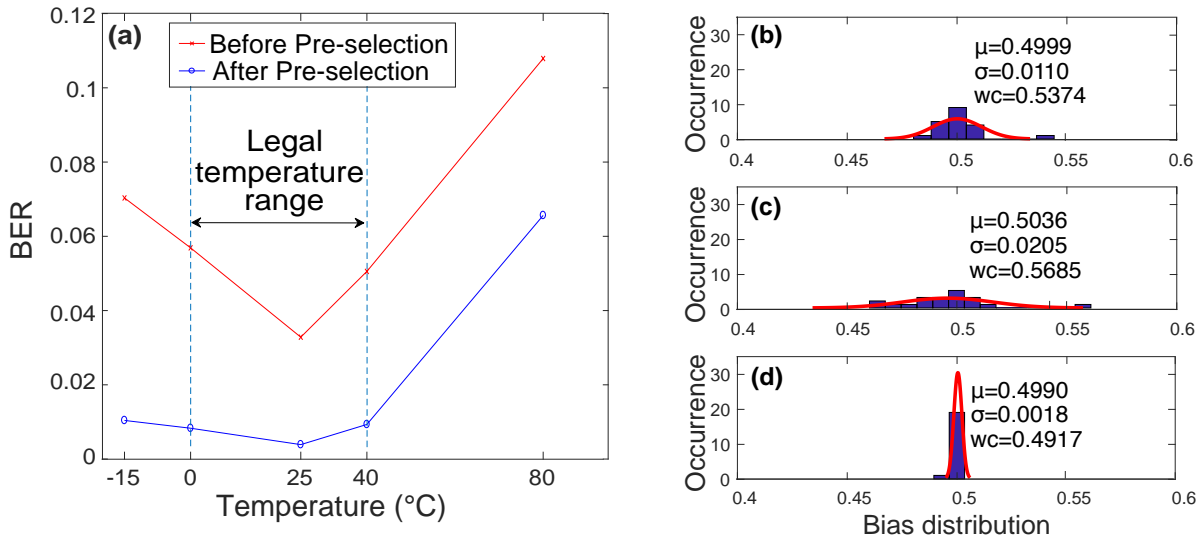


Figure 3.7: (a) BER across differing operating temperatures. Bias distribution of 20 tested chips: (b) raw PUF responses; (c) after pre-selection and (d) after HW-based de-biasing. Where μ stands for the mean value, the corresponding standard deviation value is denoted as σ and the worst-case is marked with wc . We can observe the HW-based de-biasing could effectively reduce the bias of the SRAM PUF response.

Table 3.5: CRP-block efficiency.

Chip ID	Blocks produced	Available PUF bits	Efficiency
1	4	992	11.2%
2	8	1,984	22.3%
3	2	496	5.58%
4	2	496	5.58%
5	8	1984	22.3%
6	7	1736	19.5%
7	2	496	5.58%
8	5	1,240	13.9%
9	3	744	8.36%
10	6	1,488	16.7%
11	6	1,488	16.7%
12	6	1,488	16.7%
13	6	1,488	16.7%
14	7	1,736	19.5%
15	6	1,488	16.7%
16	8	1,984	22.3%
17	7	1,736	19.5%
18	4	992	11.2%
19	6	1,488	16.7%
20	6	1,488	16.7%
WISP5.1 LRG	7	1,736	19.5%

SRAM PUF Pre-selection Efficiency. Efficiency evaluates the ratio of number of the selected reliable bits in the CRP Blocks selected over the total number of possible response bits (8,896 bits; obtained by deducting 2,048 bits reserved for the stack, 5,440 bits for the SRAM-TRNG and static variables from 16,384 SRAM bits). We tested 20 new MSP430FR5969 chips and one

Table 3.6: Memory & execution load of SecuCode functional blocks. Data measured from MSP430FR5969 embedded platform, under the Code Composer Studio (CCS) 7.2.0 development environment, with GNU Compiler Collection (GCC) for MSP430 version 6.2.1.16. Optimisation settings: -O = 3; -opt_for_speed = 5.

Protocol Steps	Memory Footprint (Byte)		Clock Cycles
	Data(SRAM)	Code(FRAM)	
$\text{nonce} \leftarrow \text{RNG}()$	6	68	375
$\mathbf{r}_i \leftarrow \text{PUF}(\mathbf{c}_i)$	6	32	615
$\text{OTF} \leftarrow \text{Temp}()$	10	204	734
$\mathbf{h}_i \leftarrow \text{FE.GEN}(\mathbf{r}_i)$ ¹	53	621	109,234
$\mathbf{s}' \leftarrow \text{MAC}_{\mathbf{k}_i}(\text{firmware} \parallel \text{nonce})$ ²	58	3,198	22,197 ³
SecuCode total	133	4,123	133,155

¹FE.Gen() based on BCH(31,16,3) code.

²using a 128-bit HWAES-CMAC message authentication code.

³For a moderate firmware size of 240 bytes.

WISP5.1LRG. The result are summarised in Table 3.5. The efficiency depends on the noise level of the SRAM cells in the candidate chip, if a chip cannot provide adequate number of CRP Blocks, it should be excluded before deployment. Nonetheless, for 20 tested chips, at least two independent CRP-blocks—each consisting of 248 bits—are obtained.

Reverse Fuzzy Extractor using BCH(n, k, t). Although our BER is less than 1%, as experimentally evaluated and shown in Figure 3.7, we selected to evaluate: i) 8 blocks of BCH(31,16,3) code capable of correcting up to $\frac{3}{31}$ or 10% of bits with $P_{\mathbf{r}}^{\text{Fail}} = 1.6 \times 10^{-3}$ for 8 parallel blocks; and ii) 6 parallel blocks of BCH(63,24,7) code capable of correcting up to $\frac{7}{63}$ or 11% of bits with $P_{\mathbf{r}}^{\text{Fail}} = 8.94 \times 10^{-7}$ for 6 parallel blocks according to Equation 3.2. Executing eight BCH(31,16,3) blocks to derive a 128-bit key requires 146,535 clock cycles while computing 6 blocks of BCH(63,24,7) to obtain a 144-bit key consumes 346,776 clock cycles.

Although employing a BCH(63,24,7) code allows us to achieve an industry standard key failure rate of less than 10^{-6} , we will see in Section 3.4.3 that protocol failure due to external conditions such as intermittent powering is more likely than protocol failure due to a failed key recovery. Therefore, we employed parallel blocks of BCH(31,16,3) code as a compromise between error correction capability, computational complexity and performance in practice. In general, for each block, the computational complexity of finding the 31-bit response \mathbf{r}_i through the corresponding 15-bit public \mathbf{h} is 2^{16} —assuming that the response bits are uniformly distributed and have no correlations with each other. In order to achieve a 128-bit security level with an attack complexity of 2^{128} , we need eight such blocks as discussed in Section 3.3.3. Thus, an SRAM PUF response \mathbf{r}_i with $31 \times 8 = 248$ bits need to be readout from the SRAM PUF for each protocol session.

3.4.2 SecuCode Implementation Footprint

In Table 3.6, we summarise the cost, in terms of CPU clock cycles, FRAM memory usage for code size and SRAM memory usage for state space size, for implementing each security related building block of SecuCode on a CRFID transponder. These blocks represent the necessary overhead imposed on the CRFID transponder to build our secure update method to prevent malicious code injections attacks. While the computational cost of the reverse fuzzy extractor is significant, it is fixed for each iteration of an update session. In contrast, the computational cost of the MAC() function can increase with larger firmware code blocks (see our evaluations in Section 3.3.4).

3.4.3 SecuCode Overhead and IEM Settings

Table 3.6 shows us that the most computationally intensive security building block are the FE.Gen() (key derivation building block) and MAC() (needed for firmware integrity checks and prove authentication) implementations. Hence these building blocks are likely to be most impacted by as well as be the cause for brownout events. Therefore, both FE.Gen() and MAC() implementations operate under our IEM.

A cold start-up is when a CRFID tag is activated from a completely powered down state—i.e., where the reservoir capacitor of the CRFID device has been discharged in the absence of a wireless powering source. Cold start time is defined as the time taken by a CRFID device to respond to an RFID reader inventory command from a cold start-up. As illustrated in the control flow diagram in Figure 3.1, physically obfuscated key derivation steps occur after a cold start-up. Therefore, the FE.Gen() or key derivation method will place additional energy and computation burdens on a cold start-up initialisation process and potentially lead to unsuccessful device start-ups at the initiation of a firmware update. Hence, we evaluate the *cold start-up success and time overhead* of a key derivation operation.

We also evaluate the success rate of our FE.Gen() and MAC() functions under wireless powering conditions—different distances from a wireless powering source, i.e., RFID reader antenna—and intermittent operating settings to understand the overhead imposed by our IEM as well as the effectiveness of IEM to mitigate brownouts. We detail our experimental method in Appendix C.1.1

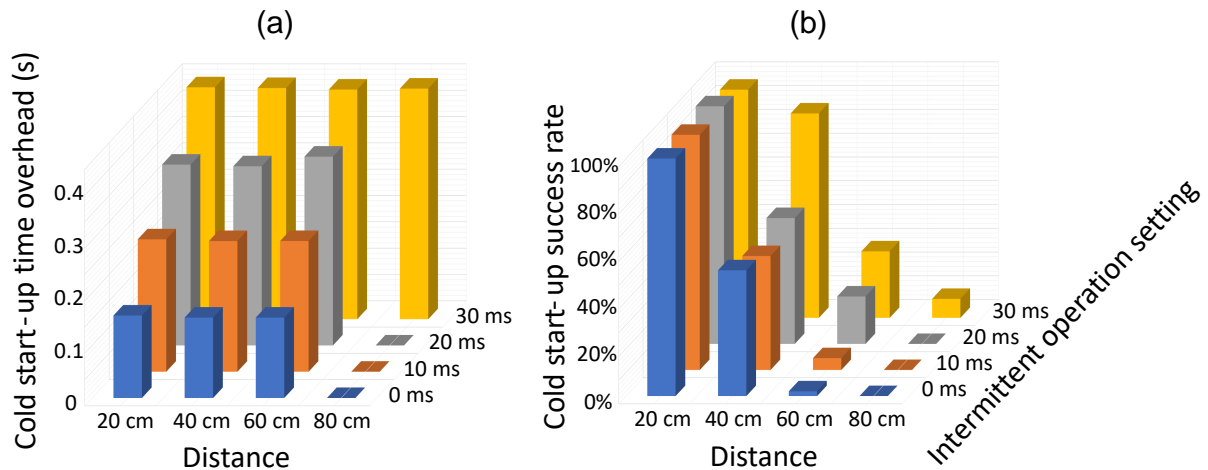


Figure 3.8: Physically obfuscated key derivation: (a) Cold start-up time overhead and (b) success rate.

Physically Obfuscated Key Derivation

In this experimental setting, we place the CRFID transponder (WISP5.1-LRG) provisioned with our bootloader at 20 cm, 40 cm, 60 cm and 80 cm apart from a 9 dBi circularly polarised reader antenna oriented towards a high ceiling to minimise interference from multipath signals on our observations. We used each of the eight BCH code computations as subtasks for our IEM.

The results of cold start-up times and success rates are plotted in Figure 3.8(a) and (b), respectively; here we plot mean cold start-up times and success rates over 100 repeated measurements collected for each intermittent operating setting and distance pair. At 20 cm interrogation range, the CRFID transponder completes key derivation from a cold start-up with a high success rate; approximately 100%. When the distance is 40 cm, the success rate witnessed a dramatic drop down to less than 50%, unless we increase the intermittent operation setting to 30 ms. When the distance is beyond 60 cm, the success rate cannot be guaranteed even using a large intermittent operation setting. Achieving higher success rates beyond 60 cm will require further division of subtasks capable of utilizing the available clock cycles before brownout. Most importantly, we observe that the IEM can increase the success rate of completing the key derivation process from a cold start-up but the trade-off is an increase in the cold start-up time overhead as illustrated in see Figure 3.8(a).

MAC Computation

We define *MAC function latency* as the time interval between calling a MAC function to perform a calculation over a block of pre-loaded data on the CRFID transponder and the generation of a valid result. We employ a single data block size. Following the definition in [166], the data size is large enough such that the latency is mainly dominated by multiple compression rounds.

3.4 Experimental Results and Analysis

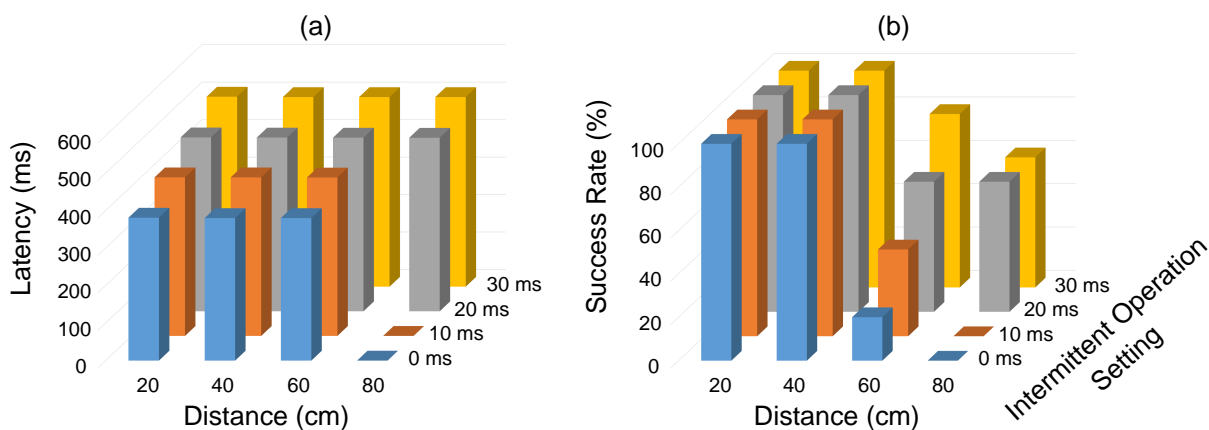


Figure 3.9: (a) MAC function success rate and (b) success rate at different distances from an RFID reader antenna under different intermittent operation settings. With data size 1280 bytes.

In this experiment, a random string of 1,280 bytes is used. Notably 1,280 bytes is twice the size of the largest firmware update used in our experiments. Since the hardware AES accelerator is very efficient in terms of clock cycles, instead of considering each AES compression round as a subtask, we employ 32 rounds of AES computations as a subtask in our IEM implementation.

We report the mean latency and success rate obtained over 10 repeated MAC computations collected for each intermittent operating setting and distance pair in Figure 3.9. As expected, the observed latency increases linearly along with the intermittent operating setting. Overall, increasing the intermittent operation setting improves the mean success rate of the MAC() computation. For example, in Figure 3.9(b), we can see that at a 60 cm distance, the success rate of the MAC computation improves from 20% to 80% with an intermittent operating setting of 30 ms. Further, we can achieve success of no less than 60% at 80 cm distance when the intermittent operating setting is greater than 20 ms.

SecuCode Case Study

As a demonstration of SecuCode in an application scenario, we evaluate SecuCode in the following setting: *There are several CRFID transponders embedded in plasterboards mounted as ceiling tiles in a chemical warehouse. We require three different types of services from these transponders. Some are to be programmed with Accelerometer service code to monitor potential structural failures, some are to be programmed with Thermometer service code to detect potentially dangerous thermal storage conditions while others are to be programmed with basic firmware to respond with a fixed Global Location Number (GLN) as anchor points to identify storage locations. The embedded CRFID transponders need to be reprogrammed wirelessly to support monitoring and location service needs of the warehouse; further, over*

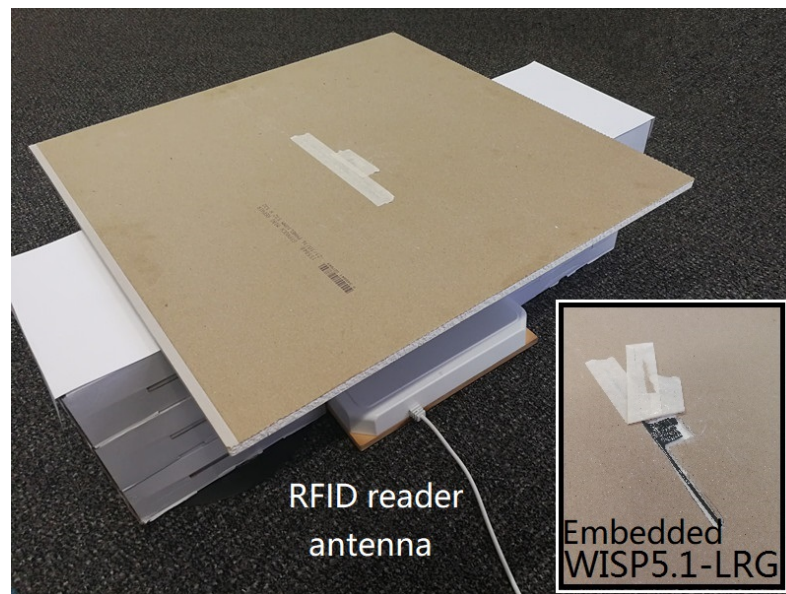


Figure 3.10: SecuCode case study set-up. The CRFID transponder is embedded in a plaster tile and placed 20 cm above the antenna to emulate the target application scenario.

time, changes to the warehouse layout and monitoring needs such as hazardous temperature levels can require alterations to existing firmware.

As shown in Figure 3.10, to emulate such a scenario, we embedded a WISP5.1-LRG into a plaster tile after provisioning the device with our bootloader. The plaster tile with the embedded WISP is then placed 20 cm above an RFID reader antenna. We used an IEM setting of 30 ms as it provided a good compromise between latency (time to complete a firmware update) and a high success rate over a range of wireless transfer distances. The results from 100 repeated wireless firmware updates based on the three firmwares specific to three distinct applications in our scenario description is summarised in Table 3.7. We successfully reprogrammed all three application firmwares within a few seconds. Most notably we observed an increased number of failures due to brownout for the larger firmware owing predominately to the increased energy required for the FE.Gen() computation. Further, as expected, all of the failures were due to brownout as opposed key recovery—recall that the key failure rate for the BCH code used is less than 0.2% as shown in Section 3.4.1. Demonstration video of the firmware update process and the source code of SecuCode App are available from[165].

3.4.4 Summary

Results in this chapter demonstrate: i) the practicability and robustness of the developed SecuCode protocol; ii) the relationship between operating conditions, intermittent operation

3.4 Experimental Results and Analysis

Table 3.7: SecuCode case study performance measures.

Application code	Size (Bytes)	Latency (mean, std) ¹ (s)	success rate (%)
Accelerometer service code	399	(3.1, 1.0)	81
Thermometer service code	273	(2.5, 0.6)	92
GLN identifier code	223	(2.1, 0.5)	90

¹Latency was measured as the time interval between the SecuCode App transmitting the firmware to an RFID reader using LLRP commands and the time SecuCode App confirms the acknowledgement of a successfully updated firmware.

settings and protocol performances; and iii) the effectiveness of the proposed IEM on realizing successful secure firmware updates under intermittent powering from harvested energy.

3.4.5 Security Analysis

We analyze the security of SecuCode under the adversary model detailed in Section 3.2.1 in the following.

Man-in-the-middle Attack

Here, security is related to the complexity of the adversary \mathcal{A} fooling the token \mathcal{T} in order to inject malicious code. Such an adversary is faced with the task of determining the secret key \mathbf{k}_i and therefore, we evaluate the security of the reverse fuzzy extractor which leaks helper data during a firmware update session.

We employ a reverse fuzzy extractor where the helper data computation is placed on the resource-constrained token \mathcal{T} . In this context, different helper data corresponding to the same response \mathbf{r}_i will be generated during different firmware update sessions; thus, multiple helper data are exposed. It has been proved that the entropy leakage from the helper data is independent of the number of enrolments for SRAM PUF [167]. In other words, one can evaluate the entropy leakage from the helper data of the reverse fuzzy extractor for a single helper data generation case; or the entropy leakage from multiple helper data observations is same as the entropy leakage from a single helper data observation, as with a traditional fuzzy extractor. Further, we assume that the de-biasing mask does not leak information. This is a reasonable assumption as the mask is limited to indicating the reliable CRP-blocks only and not the response itself. In addition, it has been demonstrated through extensive experiments that modern SRAM PUFs possess good uniqueness properties [168]. In other words, the highly reliable bits are not

correlated across different instances of SRAM PUFs. Therefore, knowledge of an SRAM PUF instance's mask can safely be assumed to not leak responses information of other SRAM PUFs.

Then, given the $BCH(n,k,t)$ code used to implement the reverse fuzzy extractor helper data computation, no more than $(n - k)$ -bit entropy is leaked. To be precise, the residual min entropy [152] of the n -bit response is given by $H_\infty = -n \cdot \log_2(\max(b, 1 - b)) - (n - k)$ when the helper data is public, where b is the bias or the uniformity of the PUF response, more details please refer to Section 2.3.2. Ideally, the bias b is 50%. However, as we have seen, PUF responses can display a slight bias. Therefore to guarantee the $(n - k)$ -bit entropy leakage we employed the de-biasing method proposed in Section 3.3.3.

As we use 8 blocks of $BCH(31,16,3)$ code, we achieve a key \mathbf{k}_i with 128-bit entropy—strictly, 127.6 bits entropy considering the experimentally evaluated bias of 0.4990 post our de-biasing method. Therefore, without knowledge of \mathbf{k}_i , the probability of fooling the token \mathcal{T} to update a malicious firmware by \mathcal{A} is no more than the brute-force attack probability of $2^{-127.6}$.

Helper Data Manipulation Attack

We are aware of helper data manipulation attacks based on exposed helper data reported in [73], [169]. However, both works acknowledge that such an attack is error correction code dependent and cannot be mounted on a linear code such as BCH employed in SecuCode [73], [169]. Further, mounting such an attack on SecuCode is difficult; in the first place, we notice that such a helper data manipulation attack is very easy to be detected by the prover \mathcal{P} . The reasons are as follows:

1. The frequency of normal firmware updates is very low in comparison with other services that are built with reverse fuzzy extractors, such as authentication or attestation services. However, helper data manipulation attacks require a large number of queries. Recall, that helper data is generated by the token \mathcal{T} and then sent to the prover \mathcal{P} and the firmware update process is initiated by the prover. Hence the opportunity to mount a helper data attack is limited to the few occasions during which a firmware update is required by the prover. In a reverse fuzzy extractor attack, the adversary is forced to perform a trial-and-error measure by continually sending malicious helper data to the resource-rich prover \mathcal{P} . The prover can perceive such malicious behaviour and stop responding to a firmware update session that is hijacked by an adversary.

3.5 Related Work and Discussion

2. There exists a built-in throttling and obfuscation mechanism that prevents rapid helper data submissions needed for such an attack. Recall that in a man-in-the-middle attack where the adversary is able to submit manipulated helper data, the success or failure is masked by failure from external factors such as powering and the determination of success or failure is only evident at the near conclusion of the update session—this can be 2-4 seconds.
3. Under tampered helper data queries, the failure rate of reconstructing \mathbf{k}_i will significantly increase. Thus, the prover \mathcal{P} can detect such an abnormal failure rate as a potential attack [72]. This can deem the target CRFID as being compromised. Alternatively, detection can also be achieved by extending the protocol to allow a request to the token \mathcal{T} to send back $\text{MAC}_{\mathbf{k}_i}(\text{nonce}||\mathbf{h}_i)$ that the prover \mathcal{P} can subsequently compare using its computed \mathbf{k}_i from the securely stored response \mathbf{r}' and the provided helper data \mathbf{h}_i .

Modeling Attack

We are also aware that modeling attacks on the reverse fuzzy extractors have been shown in [129], [170]. These attacks are applicable to arbiter PUFs (APUFs) [59], mainly due to the fact that the responses from APUFs are correlated. However, modeling attacks cannot be mounted on SRAM PUFs [62] because PUF responses are information-theoretically independent since each response is derived from a spatially separate SRAM cell.

3.5 Related Work and Discussion

Table 3.8: Comparison between related works.

Protocol	Passively powered	In-application behaviour modification	Wireless firmware update	Broadcast to multi-CRFID	Security
Bootie [171]	✗	✗	✗	✗	✗
FirmSwitch [172]	✓	✓	✗	✗	✗
R2/R3 [173]	✓	✓	✓	✗	✗
Wisent [29]	✓	✓	✓	✗	✗
Stork [22]	✓	✓	✓	✓	✗
MSPboot [141]	✗	✓	✓	✗	✗
SecuCode	✓	✓	✓	✗	✓

For emerging battery-free computing devices such as CRFID platforms including WISP[13], MOO[14] and Farsens Pyros[15], firmware updates are usually done with a wired programming interface; for example by way of a JTAG [14] interface or a Serial interface [174]. The main difficulties that hinder CRFID platform to be reprogrammed using a wireless method

are: i) the transiently powered nature where encountering power failures are highly likely[29]; ii) reprogramming code memory such as FLASH that require the device attaining an adequate voltage level from harvested power[173]; and iii) the lack of supervisory control of an operating system for managing a devices' tasks [171]. We have seen recent efforts to bring wireless reprogramming to CRFID transponders[29], [171]–[173], however, to the best of our knowledge, SecuCode is the *first* work to resolve the requirement for security for wireless code dissemination for intermittently powered passive CRFID devices. Therefore, in the following we review studies to: i) develop a bootloader and modify application behaviour; and ii) progress towards developing on-the fly wireless firmware update methods. We summarise the key characteristics of these studies in Table 3.8 and provide benchmark results for SecuCode with the *non-secure* wireless update method, Wisent [29], in Section 3.5.3.

3.5.1 In-application Behaviour Modification

An early version of a bootloader for a CRFID platform, *Bootie* was proposed by Ransford in [171]. *Bootie* was designed to accept two (or more) firmware and cross-compile them into one executable to be preloaded onto a CRFID transponder. The compiled firmware was then downloaded and tested on Olimex MSP430-H2131 minimum system board. The author showed that *Bootie* could be used as a basis for wireless firmware updates. However, as a proof-of-concept, *Bootie* only enables the platform to execute pre-loaded firmwares one-by-one and did not allow responding to user demands nor operating conditions to determine the switching between firmware.

The *FirmSwitch* scheme was later demonstrated in [172] to offer firmware flexibility for CRFID transponders. This approach allowed a user to switch between *pre-loaded* firmware instances on a CRFID platform using downstream commands to the CRFID transponder. However, *FirmSwitch* was not developed to support wireless firmware updates.

3.5.2 Wireless Code Dissemination

More recently, the *Wisent* method by Tan et al. [29], R^2 and R^3 method by Wu et al. [140], [173] demonstrated a robust wireless firmware update method for CRFID transponders using WISP platforms. In particular, R^3 was implemented on three different types of CRFID transponders. Subsequently, in *Stork*, Aantjes et al. [22] proposed a fast Multi-CRFID wireless firmware transfer protocol that involves ignoring the RN16 handle sent from an RFID transponder (i.e., the tags still save the downstream data even if their handle does not match the one specified by

3.5 Related Work and Discussion

the reader). Stork enables an RFID Reader to simultaneously program multiple CRFID devices in the field to reduce the time to update multiple devices. Although these works achieved on the fly wireless update of firmware along with a bootloader design, none of wireless firmware update approaches address the issue of security and the trustworthiness of the prover, therefore, malicious firmware injection remains an open issue.

In late 2016, Brown and Pier from TI presented an application port[141] extending TI's previous work, *MSPBoot*[175]. In this chapter, wireless updating was demonstrated in two examples; using UART or SPI bus to interconnect an MSP430 16-bit Reduced Instruction Set Computer (RISC) microcontroller and a CC1101 sub-1GHz RF transceiver. The enhanced bootloader design supported: i) application validation; ii) redirecting interrupt vectors; and iii) code sharing via pre-configured callbacks. Additionally, a dual image fail-safe mechanism is introduced; here, before the application in executable area is overwritten, the new image would be verified in a download area. Therefore any interruption in communication would not affect the function of the device. However, like other boot-loaders and wireless firmware update methods, security is not considered.

3.5.3 End-to-end Comparison

In the absence of another secure method, we selected Wisent [29] as the *non-secure* protocol to benchmark SecuCode against, because: i) Wisent focuses on the dissemination of firmware, albeit non-secure, to a *single* device, as we do; ii) both projects employ the same CRID device; and iii) Wisent code is publicly available. Notably, a comparison of Wisent to Stork [143], capable of broadcasting code to many CRFID devices, is found in [143] already.

It is difficult to make a direct comparison since the secure and non-secure approaches have fundamentally different goals and tradeoffs. For example: i) the write and check method in Wisent vs. write all and validate method of SecuCode; and ii) use of a custom bootloader in Wisent vs. our bootloader design built on the MSPboot framework of Texas Instruments for industry compliance.

However, to provide an understanding of the overhead the secure method, we provide measurements for three performance measures: i) mean latency to successfully transfer a given firmware; ii) success rate; and iii) throughput (successful firmware bytes written/time taken).

Both Wisent and SecuCode require a pre-installed bootloader which includes the WISP 5 base firmware (5,600 to 5,700 bytes). The Wisent bootloader requires an additional 608 bytes of code

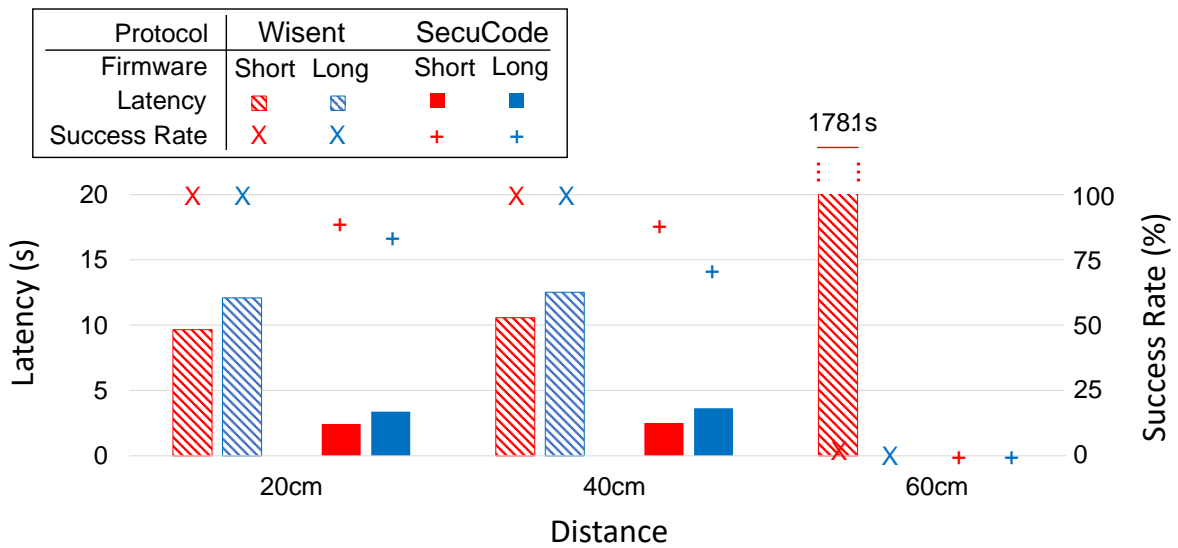


Figure 3.11: Comparing latency and success rate of SecuCode and Wisent (non-secure method) [29].

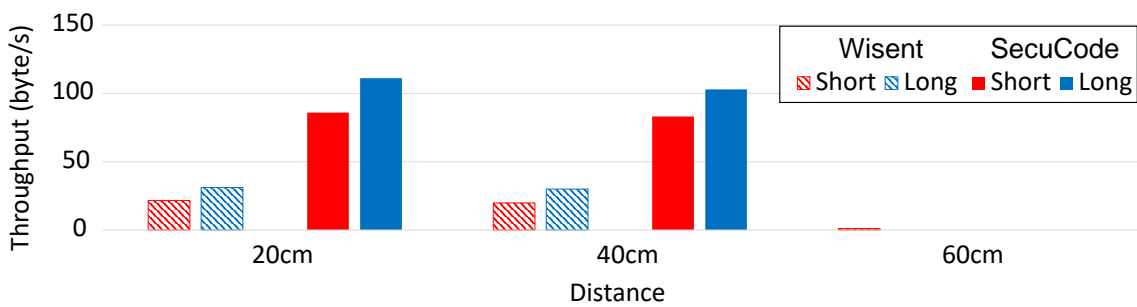


Figure 3.12: Comparing the throughput of SecuCode and Wisent (non-secure method) [29].

memory, while the SecuCode bootloader requires 6,024 bytes of code memory as a consequence of additional security routines¹⁰

In Figure 3.11 and in Figure 3.12, we compare the end-to-end performance at three different operating distances. These results are obtained from evaluations based on 100 repeated firmware update attempts. We used two firmware sizes: i) Short (210 bytes); and ii) Long (376 bytes).

SecuCode and Wisent configure the reader to transmit the same `BlockWrite` multiple times to increase `BlockWrite` success rate without the overhead of checking the result on the host. In addition Wisent includes a checksum for each block, if the checksum does not match then Wisent re-sends the block. Notably, SecuCode only checks that the `BlockWrite` command was ACKed by the tag.

¹⁰These values are dependent on optimisation settings/compiler versions and are only approximate. Tested under the Code Composer Studio (CCS) 7.2.0 development environment, with GNU Compiler Collection (GCC) for MSP430 version 6.2.1.16. Optimisation settings: `-O = 3; -opt_for_speed = 5`

3.6 Acknowledgement

Wisent is able to achieve 100% success rate at 20 cm and 40 cm, however the overhead of the per-block checksum and message header decreases the throughput. SecuCode only validates data integrity once the firmware has been completely transmitted; and refusing the firmware update if the integrity check fails. Further, any power loss event causes SecuCode to enter into a new firmware update session. Consequently, the success rate for SecuCode varies between 73% and 89% and depends on firmware size. However, compared to Wisent, SecuCode has better throughput. Notably, Wisent attained only one successful firmware update out of 100 trials at 60 cm and the update took 178.1 seconds to complete with a resulting throughput of 1.18 bytes per second. While SecuCode ceased to complete the firmware update successfully at 60 cm due to a brownout.

3.6 Acknowledgement

Michael Chesser contributed to the development of the SecuCode App (the desktop application) and interfacing with the RFID reader, and the immutable on-device SecuCode bootloader described in Section 3.3.6.

3.7 Chapter Summary

This chapter presents the first secure wireless firmware update scheme, SecuCode, for resource-constrained and intermittently powered CRFID devices. We derived a volatile secret key on demand and discard it after usage to remove the difficulty of permanent secure key storage in NVM. The SecuCode protocol developed in this chapter only allows an authorised party to perform a wireless firmware update and does not require any hardware modifications whilst being standards compliant. As noted in [62], cryptographic engineering of a protocol must consider the complex environment for physical devices, such as noise and energy constraints, performance and cost of protocol instantiation. To this end, we have successfully addressed security and implementation challenges, realised an end-to-end SecuCode implementation on the popular CRFID transponder and extensively evaluated the cost and performance of our realisation, including an application case study along with a complete public release of code and experimental data.

The next chapter focuses on the following issues. As the first attempt towards securing wireless firmware update for CRFID, SecuCode only considers the dissemination of code to a single device at a time. However, this will largely limit the efficiency of the firmware update if there

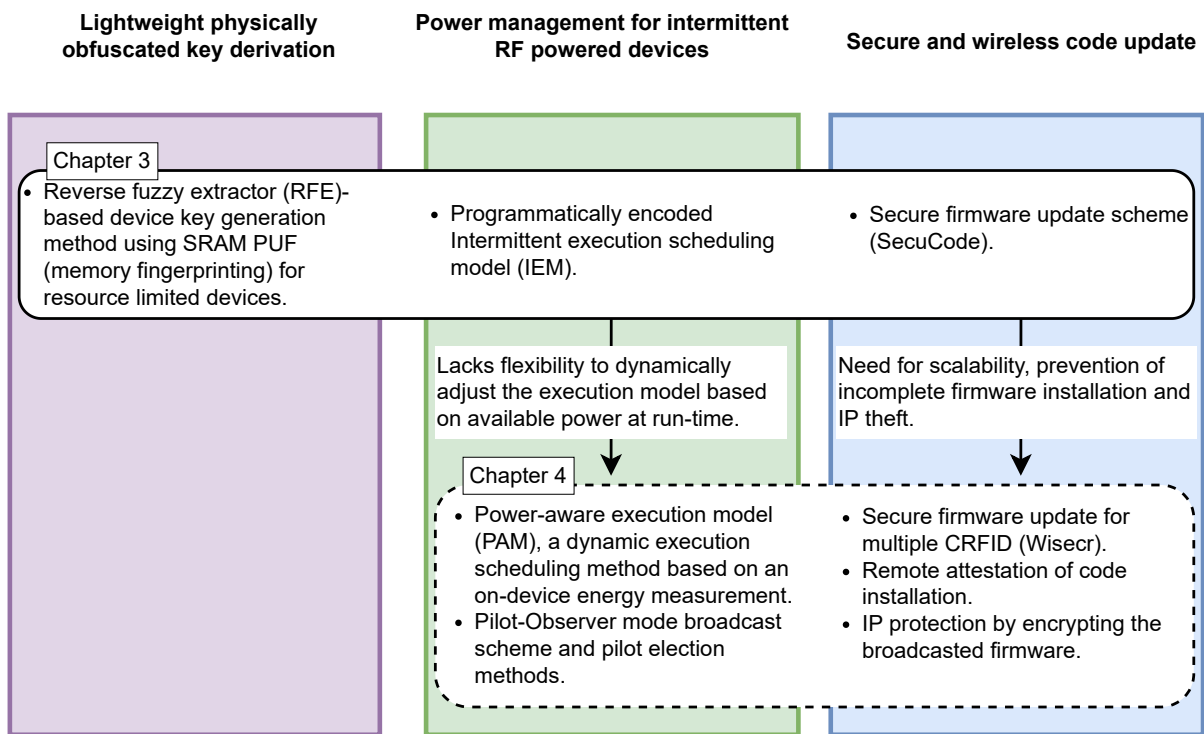
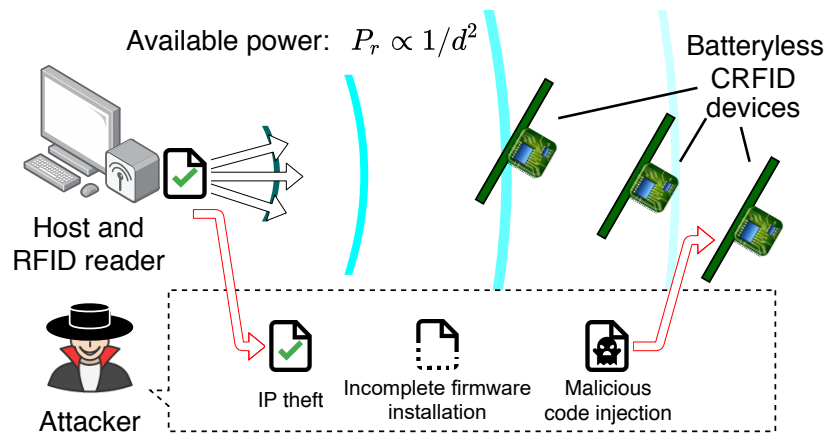


Figure 3.13: Upcoming chapter sneak peek.

are large number of devices. Although SecuCode has addressed the problem of malicious code injection attacks, the privacy or intellectual property (IP) protection goals are not addressed by SecuCode since the firmware is sent as plaintext over the wireless channel. In SecuCode, the server is notified whether the firmware update is successful or not by a simple ACK response, however an attacker can easily mimic this command and the server cannot validate the firmware installation. It is important to provide a mechanism for the server to verify the firmware installation. Meanwhile, the IEM developed in this chapter, used programmatically encoded intermittent operating settings (low power deep sleep duration) for the FE.Gen() and MAC() functions. These settings are determined when the bootloader is provisioned, and therefore, lacks flexibility to dynamically adjust the execution model based on available power at run-time. For example, the method can unnecessarily slow down firmware updates at short operating distances. Therefore an execution model aware of harvested energy is desirable. We summarise the issues above in Figure 3.13 and provide a sneak peek into the upcoming Chapter 4 where the issues raise herein are addressed.

Chapter 4

Secure Simultaneous Code Updates to Multiple CRFID Devices



The first secure wireless firmware update scheme for CRFID, SecuCode was demonstrated in Chapter 3. However, in SecuCode, only one CRFID device can be updated at a time. This chapter proposes Wisecr, the first secure and simultaneous wireless code dissemination mechanism to multiple devices that prevents *malicious code injection attacks* and *intellectual property (IP) theft*, whilst enabling *remote attestation of code installation*. Importantly, Wisecr is engineered to comply with existing ISO compliant communication protocol standards employed by CRFID devices and systems. We comprehensively evaluate Wisecr’s overhead, demonstrate its implementation over standards compliant protocols, analyse its security and implement an end-to-end realisation with popular CRFID devices.

4.1 Motivation and Contribution

Wired programming interfaces, such as the Joint Test Action Group (JTAG), are convenient tools for developing electronic products. For security reasons, such kind of wired programming interface shall be disabled at manufacture to prevent further access, as discussed in Chapter 1 and Chapter 3. Hence, it leaves the wireless update option as the only method to alter the firmware or to re-purpose the devices post-manufacture.

When a large number of devices are required to be reprogrammed, with their wired programming interface disabled, a secure and simultaneous firmware update mechanism to many CRFID devices is highly desirable. For example, Federal Aviation Administration (FAA) of the United States granted the installation of RFID tags and sensors on aeroplanes in 2018 [176]; consequently, an increasing number of CRFID sensors are integrated with small aircraft and commercial airliners for maintenance history logging [22] and aircraft health monitoring [24]. In such a scenario, a high-efficiency (simultaneous) and secure update is desirable to ensure operational readiness and flight safety.

Although the Chapter 3 has addressed the problem of malicious code injection attacks whereby a single CRFID device is updated in turn, it neither supports simultaneous updates to many devices nor protects firmware IP and lacks a mechanism to validate the installation of code on a device. While the recent study Stork [22] protocol addressed the challenging problem of fast wireless firmware updates to multiple CRFID devices, Stork allows any party, authorised or not, armed with a simple RFID reader to: i) mount malicious code injection attacks; ii) result in an incomplete firmware installation by cheating the Host with a fake indication of a successful firmware update; and iii) steal intellectual property (IP) by simply eavesdropping on the non-secure over-the-air communication channel.

Therefore, this chapter considers the following problems:

Problem 1. How can we develop a *secure and simultaneous* code dissemination to multiple passively powered CRFID devices operating under *constrained protocols, device capability, and extreme on-device resource limitations*—computing power, memory, and energy as introduced in Section 1.2 in Chapter 1?

Problem 2. How can we address the security threats including: *Malicious code injection* (code alteration, loading unauthorised code, loading code onto an unauthorised device, and code downgrading), *Incomplete firmware installation*, and *IP theft* (reverse engineering from plaintext binaries)?

Table 4.1: Table of Notations in This Chapter

S	Server S is a single entity consisting of a host computer and a networked RFID reader.
\mathcal{T}	Tokens \mathcal{T} is a collection of individual (CRFID) devices T_i .
\mathcal{A}	The adversary \mathcal{A} is an attacker seek to commit IP theft, cause incomplete firmware installation and conduct malicious code injection, by interposing the communication between S and \mathcal{T} .
DB	Server's database, where each element is a three-tuple describing each CRFID token: i) the unique and immutable identification number \mathbf{id}_i ; ii) the device specific secret key \mathbf{k} ; and iii) a flag denoting a device requiring an update valid .
firmware	The new firmware binary for the update.
firmware^(j)	The j_{th} block of the firmware .
nver	The new version number (a monotonically increasing ordinal number with a one-to-one correspondence with each updated firmware).
ver_i	The current firmware version number of token T_i .
sk	The broadcast session key (for more details, please refer to Section 4.2.2).
s	A MAC tag computed over the firmware , ver_i and nver , computed by the Server (for more details, please refer to Section 4.2.2).
c, r	Attestation challenge and response, respectively (for more details, please refer to Section 4.2.2).
\square'	An apostrophe denotes a value computed by a different entity, e.g., s' the MAC tag computed by a Token.
\square_i	A subscript i denotes a specific entity, e.g., \mathbf{k}_i is the device key of the i_{th} Token.
$\langle \square \rangle$	Encrypted data, e.g. $\langle \mathbf{sk}_i \rangle$ denotes the encrypted session key \mathbf{sk} , with token T_i 's device key \mathbf{k}_i .
RNG()	A cryptographically secure random number generator (for more details, please refer to Section 4.2.2).
SKP.Enc()	Symmetric Key Primitive encryption function described by $\langle \mathbf{m} \rangle \leftarrow \text{SKP.Enc}_{\mathbf{sk}}(\mathbf{m})$. Here, the plaintext \mathbf{m} is encrypted with the \mathbf{sk} to produce the ciphertext $\langle \mathbf{m} \rangle$ (for more details, please refer to Section 4.2.2).
SKP.Dec()	Symmetric Key Primitive decryption function where $\mathbf{m} \leftarrow \text{SKP.Dec}_{\mathbf{sk}}(\langle \mathbf{m} \rangle)$ (for more details, please refer to Section 4.2.2).
MAC()	Message Authentication Code function. By appending an authentication tag \mathbf{s} to the message \mathbf{m} , where $\mathbf{s} \leftarrow \text{MAC}_{\mathbf{k}}(\mathbf{m})$, a message authentication code (MAC) function can verify the integrity and authenticity of the message by using the symmetric key \mathbf{k} (for more details, please refer to Section 4.2.2).
SNIFF()	The voltage $\mathbf{V}t_i$ established by the power harvester within a fixed time t from boot-up at token i , is measured by SNIFF() described by $\mathbf{V}t_i \leftarrow \text{SNIFF}(t)$ (for more details, please refer to Appendix D.4).
PAM()	Family of functions employed by the Power Aware Execution mode of operation proposed for the token to mitigate power-loss (brownout) events (for more details, please refer to Figure 4.3.2).

Consequently, the study in this chapter fulfils the *urgent* and *unmet* security needs in the existing state-of-the-art multiple CRFID wireless dissemination protocol—Stork [22]. The contributions made in this Chapter are summarise below:

- **Develop a first secure and simultaneous (*fast*) wireless firmware update protocol.** Wisecr is the first secure and simultaneous (*fast*) firmware dissemination scheme to multiple batteryless CRFID devices. Wisecr provides three security functions for secure updates: i) preventing malicious code injection attacks; ii) IP theft; and iii)

4.1 Motivation and Contribution

attestation of code installation. Wisecr achieves *fast* updates by supporting simultaneous update to multiple CRFID devices through a secure broadcasting of firmware over a *standard non-secure unicast air interface protocol* (**Problem 1** and **Problem 2**).

- **A holistic design trajectory.** Wisecr starts from a formal secure scheme design to an end-to-end implementation requiring only limited on-device resources. Ultra-low power conditions and on-device resource limitations demand both a secure and an efficient scheme. First, we built an efficient broadcast session key exchange exploiting commonly available hardware acceleration for crypto on microcontroller units (MCUs). Second, to avoid power loss and thus achieve uninterrupted execution of a firmware update session, we propose *new* methods: i) adaptive control of the execution model of devices using RF powering channel state information collected and reported by field deployed devices; ii) reducing disruptions to broadcast data synchronisation across multiple devices by introducing the concept of a *pilot tag* selection from participating devices in the update scheme to drive the protocol. These methods avoid the need for costly, secure checkpointing methods and leaving a device in a vulnerable state during power loss. Third, in the absence of an operating system, we develop an immutable bootloader to: i) supervise the control flow of the secure firmware update process; ii) minimise the occurrence of power loss during an update session whilst abandoning a session in case an unpreventable power loss still occurs; and iii) manage the secure storage of secrets by exploiting commonly available on-chip memory protection units (MPUs) to realise an immutable, bootloader-only accessible, secrets (**Problem 1**).
- **A standards compliant method and an end-to-end realisation.** We develop Wisecr from specification, component design to architecture on the device, and implementation. We evaluate Wisecr extensively, including comparisons with current non-secure methods, and validate our scheme by an end-to-end implementation with an open source code release on GitHub (the link is available in Section 1.4). Wisecr is a standard compliant secure firmware broadcast mechanism with a demonstrable implementation using the widely adopted air interface protocol—ISO-18000-63 protocol albeit the protocol’s lack of support for broadcasting or multi-casting—and using commodity devices from vendors. Hence, the Wisecr scheme can be adopted in currently deployed systems (**Problem 1**). We demonstrate the firmware dissemination process here:

<https://www.youtube.com/watch?v=GgDHPJi3A5U>



scan to watch

4.1.1 Chapter Overview

Section 4.2 presents the threat model, security requirements and Wisecr design; Section 4.3 details how could we achieve the desired security requirements under challenging settings; Section 4.4 discusses our end-to-end implementation, followed by performance and security evaluations; Section 4.5 discusses related work with which Wisecr is compared. Section 4.6 is dedicated for our collaborator who made remarkable contribution to this chapter. Section 4.7 concludes this chapter.

4.1.2 Notations and Concepts

Adding to the general notations and conventions defined in Section 2.1 in Chapter 2, Table 4.1 summarises some key concepts introduced and referred to in this chapter.

4.2 Wisecr Design

In this section, we describe the threat model followed by details of Wisecr design, and then proceed to identify the minimal hardware security requirements needed to implement the scheme.

4.2.1 Threat Model

Our analysis and design are based on the networked RFID system illustrated in Figure 2.1 in Chapter 2. The RFID reader resides at the edge of the network and is typically connected to multiple antennas to power and communicate with RFID or CRFID devices energised by the antennas. The reader network interface is accessed by using a Low-Level Reader Protocol (LLRP) from a host machine. Communication with an RFID device operating in the UHF range is through the *EPC Gen2* protocol.

Our focus is on the insecure communication channel between the RFID reader connected antenna and the CRFID transponder or token $T_i \in \mathcal{T}$. Hence, we assume that the communication between a host and a reader is secured using standard cryptographic mechanisms [89]. Therefore, a host computer and a reader are considered as a single entity, the Server, denoted as S (a detailed execution of an *EPC Gen2* protocol session to *singulate* a single CRFID device from all visible devices in the field can be found in [22], [29]).

4.2 Wisecr Design

We assume a CRFID device can meet the pragmatic hardware security requirements, detailed in Section 4.2.3. Further, after device provisioning, the wired interface for programming is disabled—using a common technique adopted to secure resource-constrained microcontroller based devices [30], [38]. Subsequently, both the trusted party and adversary \mathcal{A} must use the wireless interface for installing new firmware on a token.

Building upon relevant adversary models related to wireless firmware update for low-end embedded devices [142], [177], we assume an adversary \mathcal{A} has full control over the communication channel between the Server S and the tokens \mathcal{T} . Hence, the adversary \mathcal{A} can eavesdrop, manipulate, record and replay all messages sent between the Server S and the tokens \mathcal{T} . This type of attacker is referred to as an input/output attacker [178].

We assume the firmware (application), potentially provided by a third party in the form of a stripped binary, may contain vulnerabilities or software bugs that can cause the program to deviate from the specified behavior with potential consequences being corruption of the bootloader and/or the non-volatile memory (NVM) contents. Such an occurrence is possible when firmware is frequently written in unsafe languages such as C or C++ [179], [180]. Hence, the firmware (application) cannot be trusted. In this context, similar to [142], [177], we also assume that the adversary cannot bypass any of the memory hardware protections (detailed in Section 4.2.3) and an adversary cannot mount invasive physical attacks to extract the on-chip non-volatile memory contents. Such an assumption is practical, especially in deeply embedded applications such as pacemaker control [29] where wireless update is the only practical mechanism by which to alter the firmware and physical access is extremely difficult.

As in [62], we also assume the adversary \mathcal{A} cannot mount implementation attacks against the CRFID, or gain internal variables in registers, for example, using invasive attacks and side-channel analysis. We do not consider Denial of Service (DoS) attacks because it appears to be impossible to defend against such an attacker, for example, that disrupts or jams the wireless communication medium in practice [142].

4.2.2 Wisecr Update Scheme

Wisecr enables the ability to securely distribute and update the firmware of multiple CRFID tokens, simultaneously. Given that a Server S must communicate with an RFID device \mathcal{T} using *EPC Gen2*, Wisecr is compatible with *EPC Gen2* by design. Generally, our scheme can be implemented after the execution of the anti-collision algorithm in the media access control layer of the *EPC Gen2* protocol, where a reader must first singulate a CRFID device and obtain

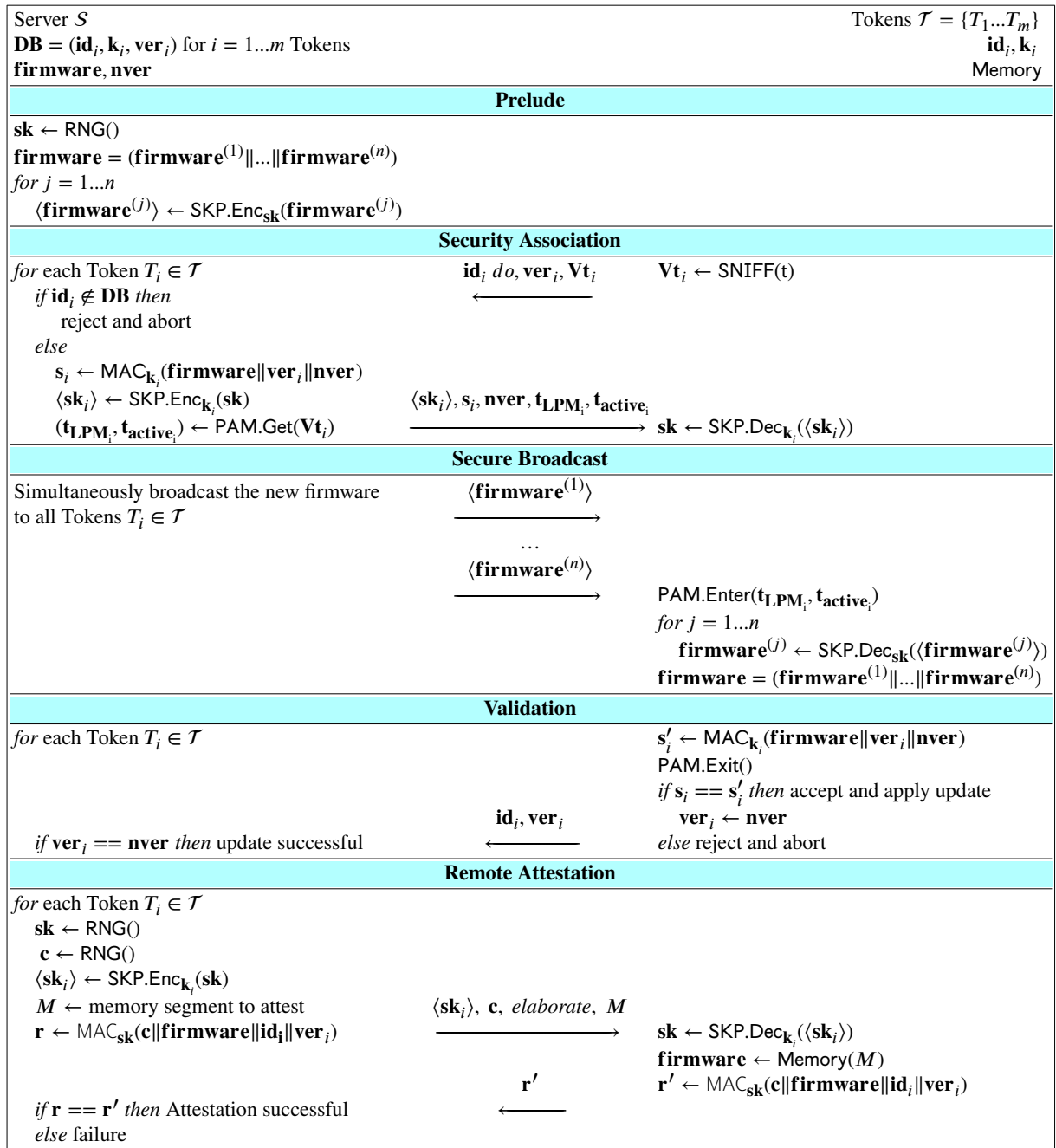


Figure 4.1: Wisecr: The proposed wireless, secure and simultaneous code dissemination scheme to multiple tokens. Notably, the functions SNIFF() and PAM() facilitate the secure and uninterrupted execution of an update session. A single symmetric key cipher SKP can be exploited in practice to realise all of the primitives needed, including the MAC() function, to address the demands of a resource-limited setting.

a handle, *RN16*, to address and communicate with each specific device. After singulating a device, the server can employ commands such as: BlockWrite, Authenticate, SecureComm and TagPrivilege specified in the *EPC Gen2* access command set [181], to implement the Wisecr scheme. Firmware update will take place once a device enters the *Access* state and by implementing Wisecr over *Access* command specifications such as

Authenticate. We describe the update scheme in Figure 4.1 and defer the details of our scheme implemented over the *EPC Gen2* protocol to the Appendix D.2.

As described in Figure 4.1, the Server \mathcal{S} maintains a database \mathbf{DB} of provisioned tokens, issues the new **firmware** and the corresponding version number (**nver**).

Each token T_i has a secure storage area provisioned with: i) an \mathbf{id}_i , the immutable identification number; ii) \mathbf{k}_i , the device specific secret key stored in NVM that is read-only accessible by the immutable bootloader. The \mathbf{k}_i assigned to different devices are assumed to be *independent and identically distributed* (i.i.d.); iii) \mathbf{ver}_i , the token's current firmware version number. The secure storage area is only accessible by the trusted and immutable bootloader provisioned on the device; this region is inaccessible to the firmware (application), and therefore, cannot be modified by it.

We describe a dissemination session in four stages: i) Prelude; ii) Security Association; iii) Secure Broadcast; iv) Validation. An update can be extended with an optional; and v) Remote Attestation to verify the firmware installation.

STAGE 1: Prelude (Offline). In this stage, the Server \mathcal{S} undertakes setup tasks. The Server uses $\text{RNG}()$ to generate a broadcast session key (\mathbf{sk}). The new **firmware** is divided into segments—or block; each block j is encrypted as $\langle \mathbf{firmware}^{(j)} \rangle \leftarrow \text{SKP.Enc}_{\mathbf{sk}}(\mathbf{firmware}^{(j)})$, where $\langle \mathbf{firmware}^{(j)} \rangle$ denotes the encrypted firmware block j . The division of firmware is necessary as the narrow band communication channel and the *EPC Gen2* protocol does not allow *arbitrary* size payloads to be transmitted to a token.

STAGE 2: Security Association. In this stage, the Server \mathcal{S} distributes the broadcast session key (\mathbf{sk}) to all tokens \mathcal{T} and builds a secure broadcast channel over which to simultaneously distribute the firmware to multiple tokens.

More specifically, each token in the energizing field of the Server responds with \mathbf{id}_i , \mathbf{Vt}_i and \mathbf{ver}_i . The token will not be included in the following update session if: i) the \mathbf{id}_i of the responding token is not in Server's \mathbf{DB} ; or ii) the token is not scheduled for an update ($\mathbf{valid}_i == \text{False}$). For tokens selected for an update, the Server computes a MAC tag $\mathbf{s}_i \leftarrow \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware} || \mathbf{ver}_i || \mathbf{nver})$. In practice, we cannot assume that each token is executing the same version of the firmware, therefore, a token specific MAC tag is generated over the device specific key whilst the firmware is encrypted with the broadcast session key.

The Server establishes a shared session key with each token T_i by sending $\langle \mathbf{sk}_i \rangle$, where $\langle \mathbf{sk}_i \rangle \leftarrow \text{SKP.Enc}_{\mathbf{k}_i}(\mathbf{sk})$ and \mathbf{k}_i is specific to the i_{th} token, and \mathbf{nver} . The $\langle \mathbf{sk}_i \rangle$ and \mathbf{s}_i are transmitted to

each token T_i . Each token decrypts the broadcast session key $\mathbf{sk} \leftarrow \text{SKP.Dec}_{\mathbf{k}_i}(\langle\langle\mathbf{sk}_i\rangle\rangle)$ —thus, all tokens selected for an update now possess the session key.

Notably, as detailed in Section 4.3.2, each token measures its *powering channel state* or its ability to harvest energy by measuring the voltage \mathbf{Vt}_i established by the power harvester within a fixed time t from boot-up, as $\mathbf{Vt}_i \leftarrow \text{SNIFF}(t)$ to transmit to the Server. There are two important reasons for measuring \mathbf{Vt}_i . First, to facilitate the power aware execution model (PAM) employed to mitigate power-loss at a given token. The Server uses the reported \mathbf{Vt}_i to control the execution model of the i_{th} token. Specifically, the reported \mathbf{Vt}_i is used by the Server to determine the length of time that a token dwells in low power mode (LPM) t_{LPM_i} and active mode t_{active_i} when executing computationally intensive tasks in the Secure Broadcast and Validation stages; here, the server determines $(t_{\text{LPM}_i}, t_{\text{active}_i}) \leftarrow \text{PAM.Get}(\mathbf{Vt}_i)$. Second, the Server uses the reported \mathbf{Vt}_i to realise the *Pilot-Observer Mode* of operation where one token is *elected* based on its \mathbf{Vt}_i , termed the *Pilot*, to control the flow in the Secure Broadcast stage by responding to server commands as detailed in Stage 3.

STAGE 3: Secure Broadcast. In this stage, the encrypted firmware blocks $\langle\langle\mathbf{firmware}^{(1..n)}\rangle\rangle$ are broadcasted; and each token stores the new encrypted firmware blocks in its application memory region (Segment M defined in Section 4.3.1). Once the broadcast is completed, each token starts firmware decryption and validation. The $\langle\langle\mathbf{firmware}\rangle\rangle$ is decrypted using the session key \mathbf{sk} as $\mathbf{firmware}^{(j)} \leftarrow \text{SKP.Dec}_{\mathbf{sk}}(\langle\langle\mathbf{firmware}^{(j)}\rangle\rangle)$.

To realise a secure and power efficient logical broadcast channel under severely energy-constrained settings, we use the *Pilot-Observer mode*. Herein, all tokens, except the *Pilot* token elected by the Server, enters into an *observer mode*. The tokens in the *observer mode* silently listen and store encrypted data disseminated by the server; the *Pilot* token performs the same operation whilst responding to the server commands. We employ two techniques within the Pilot-Observer Mode to mitigate power-loss and to achieve a secure broadcast to tokens: i) disabling energy consuming communication command reply from observers; and ii) the concept of electing a *Pilot* CRFID device to drive the update session as detailed in Section 4.3.2.

Notably, the techniques described in Stage 2 and 3 form the foundation for the uninterrupted execution of the bootloader to ensure security and enhance the performance of the firmware dissemination under potential power-loss events.

STAGE 4: Validation. In this stage, firmware is validated before installation. More precisely, a token specific MAC tag $s'_i \leftarrow \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware}||\mathbf{ver}_i||\mathbf{nver})$ is computed by each token

T_i . If the received MAC tag s_i matches the device computed s'_i , the integrity of the firmware established and the issuing Server is authenticated by the token. Subsequently, the new firmware is updated and the new version number \mathbf{nver} is stored as \mathbf{ver}_i . Otherwise, the firmware is discarded and the session is aborted. Notably, the *EPC Gen2* protocol provides a reliable transfer feature. Each broadcast payload is protected by a 16-bit Cyclic Redundancy Check (CRC-16) error detection method. Hence, the notification of a CRC failure to the Server results in the automatic re-transmission of the packet by the Server. Therefore, a MAC tag mismatch is more likely to be adversarial and discarding the firmware is a prudent action. At stage completion, each token switches from the *observer* or *Pilot* to the normal mode of operation after a software reset (reboot). Subsequently, all the temporary information such as session key \mathbf{sk} and the token specific MAC tag s'_i in volatile memory will be erased.

Once the firmware is installed, the Server is acknowledged with the status of each participating token in the session. This is achieved by performing an *EPC Gen2* handshake after a reboot of the tokens, and comparing the version number \mathbf{ver}_i reported from each token specified by \mathbf{id}_i to the new firmware version number \mathbf{nver} expected from the token. If the \mathbf{ver}_i is up-to-date, the Server is acknowledged that the token T_i has been successfully updated.

Remark. It is theoretically possible to include a MAC tag in the acknowledgement message at the end of the Validation stage to authenticate the acknowledgement. But the implementation of this in practice is difficult under constrained protocols and limited resources typical of intermittently powered devices. This is indeed the case with the *EPC Gen2* air interface protocol and CRFID devices we employed. We discuss specific reasons in Appendix D.2 where we detail implementation of the Wisecr Update Scheme over the *EPC Gen2* air interface protocol.

STAGE 5 (Optional): Remote Attestation.

The Server can elect to verify the firmware installation on a token by performing a remote attestation; a mechanism for the Server to verify the complete and correct software installation on a token. Considering the highly resource-limited tokens, we propose a lightweight challenge response based mechanism re-using the $\text{MAC}()$ function developed for Wisecr. The server sends a randomly generated challenge $c \leftarrow \text{RNG}()$ and evaluates the corresponding response \mathbf{r}' to validate the installation. We provision a new session key \mathbf{sk} to enable the remote attestation to proceed independent of the previous stages, whilst avoiding the derivation of a key on device to reduce the overhead of the attestation routine.

We propose two modes of attestation; a *fast mode* and an *elaborate mode* to trade-off veracity of the verification against computational and power costs. The *fast mode* only examines the token

serial number (\mathbf{id}_i) and the version number (\mathbf{ver}_i). While the *elaborate* mode traverses over an entire memory segment. The *elaborate* mode is relatively more time consuming but allows the direct verification of the code installed on the target token T_i .

We illustrate (in Figure 4.1) and demonstrate (in Section 4.4.5) the *elaborate* mode (more veracious and computationally intensive) where response $\mathbf{r}' \leftarrow \text{MAC}_{\mathbf{sk}}(\mathbf{c} \parallel \mathbf{firmware} \parallel \mathbf{id}_i \parallel \mathbf{ver}_i)$ attests the application memory segment containing the installed **firmware**. In contrast, the *fast* method computes the response as $\mathbf{r}' \leftarrow \text{MAC}_{\mathbf{sk}}(\mathbf{c} \parallel \mathbf{id}_i \parallel \mathbf{ver}_i)$.

4.2.3 Token Security Requirements and Functional Blocks

Our design is intentionally minimal and requires the following security blocks.

Immutable Bootloader (Section 4.3.1) We require a static NVM sector M_{rx} that is write-protected to store the executable bootloader image to ensure the bootloader can be trusted post deployment where, for example, firmware (application) code vulnerabilities or software bugs do not lead to the corruption of the bootloader and the integrity of the bootloader can be maintained.

Secure Storage (Section 4.3.1) To store a device specific secret, e.g., \mathbf{k}_i , we require an NVM sector M read-*only* accessible by the immutable bootloader in sector M_{rx} . This ensures the integrity and security of non-volatile secrets post deployment since the firmware (application) code cannot be trusted, for example, due to potential vulnerabilities or software bugs that can lead to the corruption of non-volatile memory contents).

Uninterruptible Bootloader Execution (Section 4.3.2) During the execution of the bootloader stored in sector M_{rx} , execution cannot be interrupted until the control flow is intentionally released by the bootloader.

Efficient Security Primitives (Section 4.3.3) The update scheme requires: i) a symmetric key primitive; and ii) a keyed hash primitive for the message authentication code that are both computationally and power efficient.

In Section 4.3 we discuss how the associated functional blocks are engineered on typical RF-powered devices built with ultra-low power commodity MCUs.

4.3 On-device Security Function Engineering

To provide comprehensive evaluations and demonstrations, we selected the WISP5.1LRG [9] CRFID device with an open-hardware and software implementation for our token \mathcal{T} . This CRFID device uses the ultra-low power MCU MSP430FR5969 from Texas Instruments. Consequently, for a more concrete discussion, we will refer to the WISP5.1LRG CRFID and the MSP430FR5969 MCU in the following.

4.3.1 Immutable Bootloader and Secure Storage

For resource-limited MCUs, several mechanisms—detailed in the Appendix D.1—exists for implementing secure storage: i) Isolated segments; ii) Volatile keys; iii) Execute only memory; and iv) Runtime access protections. We opt for achieving secure storage and bootloader immutability using Runtime Access Protection by exploiting the MCU’s memory protection unit (MPU), which offers flexibility to the bootloader. In particular, the MPU allows read/write/execute permissions to be defined individually for memory segments at power-up—prior to any firmware (application) code execution. Wisecr requires the following segment permissions to be defined by the bootloader to prevent their subsequent modifications through application code by locking the MPU:

Segment M is used as the secure storage area. During application execution, any access (reading/writing) to this segment results in an access violation, causing the device to restart in the bootloader.

Segment M_{rx} contains the bootloader, device interrupt vector table (IVT), shared code (e.g., *EPC Gen2* implementation). During application execution, writing to this segment results in an access violation.

Segment M_{rwx} covers the remaining memory, and is used for application IVT, code (**firmware**) and data.

4.3.2 Uninterruptible Bootloader Execution

The execution or control flow of the bootloader on the token must be uninterruptible by application code and power-loss events to meet our security objectives but dealing with brownout induced power-loss events is more challenging. Power loss leaves devices in vulnerable states for

attackers to exploit; therefore, we focus on innovative, pragmatic and low-overhead power-loss prevention methods. Our approach deliberately mitigates the chances of power-loss. In case a rare power-loss still occurs, the token will discard all state—including security parameters such as the broadcast session key; subsequently, the Server will re-attempt to update the firmware by re-commencing a fresh update session with this token. We detail our solution below.

Managing Application Layer Interruptions

The bootloader must be uninterruptible (by application code) for security considerations. For instance, the application code—due to an unintentional software bug or otherwise—could interrupt the bootloader while the device key is in a CPU register, so that the application code (exploited by an attacker) can copy the device key to a location under its control, or completely subvert access protections by overriding the MPU register before it is locked.

Recall, the memory segment M_{rx} (see Section 4.3.1) includes the memory region containing the IVT. This ensures that only the bootloader can modify the IVT. Since the IVT is under the bootloader’s control, we can ensure that any non-maskable interrupt is unable to be directly configured by the application code, whereas all other interrupts are disabled during bootloader execution. Consequently, the interrupt configuration cannot be mutated by application code.

Managing Power-Loss Interruptions

Frequent and inevitable power loss during the bootloader execution will not only interrupt the execution, degrading code dissemination performance but also compromise security. Although intermittent computing techniques relying on saving and retrieving state at check-points from NVM—such as Flash or EPPROM—is possible, these methods impose additional energy consumption and introduce security vulnerabilities revealed recently [34].

Confronted with the complexity of designing and implementing an end-to-end scheme under extreme resource limitations, we propose an on-device Power Aware execution Model (PAM) to: i) avoid the overhead of intermittent computing techniques; and ii) enhance security without saving check-points to insecure NVM.

We observe that only a limited number of clock cycles are available for computations per charge and discharge cycle (Intermittent Power Cycle or IPC) of a power harvester, as illustrated via comprehensive measurements in Figure 1.4. Further, the rate of energy consumption/depletion is faster than energy harvesting. We recognise that there are three main sources of power-loss: i) (CPU) energy required for function computation exceeding the energy supply capability from

4.3 On-device Security Function Engineering

the harvester; ii) (FRAM.R/W) memory read/write access such as in executing `Blockwrite` commands; and iii) (RFID) power harvesting disruptions from communications—especially for backscattering data in response to *EPC Gen2* commands.

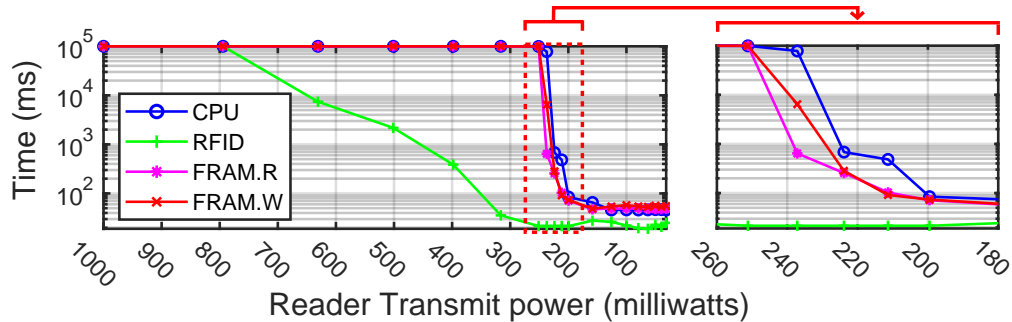


Figure 4.2: Impact of four key device tasks on power-loss: CPU (computations); FRAM.R/W (memory read/write accesses); and RFID (communications). The data above are collected by using special firmware. The power-loss event is captured by monitoring through a GPIO pin. We conduct 10 repeated measurements and report the mean time before power-loss. The plot provides a lateral comparison among four operations. The data is measured from a single CRFID device with oscilloscope probes attached to measure the device’s internal state.

To understand the severity of these four causes, we measure the maximum time duration before brownout/power-loss versus the harvested power level for each task—CPU, FRAM.R, FRAM.W and RFID. In the absence of a controlled RF environment (i.e., anechoic chamber), it is extremely difficult to maintain the same multipath reflection pattern. Especially when changing the distance between the radiating reader antenna and an instrumented CRFID device considering the multipath interference created by the probes, cables, and the nearby oscilloscope and researcher to monitor the device’s internal state. To minimise the difficulty of conducting the experiments, we place the CRFID device at a fixed distance (20 cm) whilst keeping all of the equipment at fixed positions, and adjust the transmit power of the RFID reader through the software interface. According to the free-space path loss equation [97], adjusting the transmit power of the RFID reader or changing the distance can be used to vary the available power at the CRFID device. We describe the detailed experimental settings in Appendix D.8. For experiments without the requirement for monitoring the device’s internal state, we still employ distance-based measurements as in previous studies [22], [32], [38].

The results are detailed in Figure 4.2. For a transmit power greater than 800 mW, the CRFID transponder continuously operates without power failure within 100 seconds. If the reader transmit power is below 800 mW, the average operating time of the RFID task drops as the power level decreases. This is because the RFID communication process invokes shorting the antenna, during which the energy harvesting is interrupted. Notably, different from Flash memory, reading data from FRAM (FRAM.R) consumes more power than writing to it (FRAM.W) as

a consequence of the destructive read and the compulsory write-back [182]. Consequently, we developed:

- The *Pilot-Observer Mode* to reduce the occurrence of RFID tasks by enabling observing devices to listen to broadcast packets in silence whilst *electing* a single *Pilot* token to respond to the Server.
- The *Power Aware Execution Model (PAM)* to ensure memory access (FRAM tasks) and intensive computational blocks of the security protocol (CPU tasks) do not exceed the powering capability of the device.

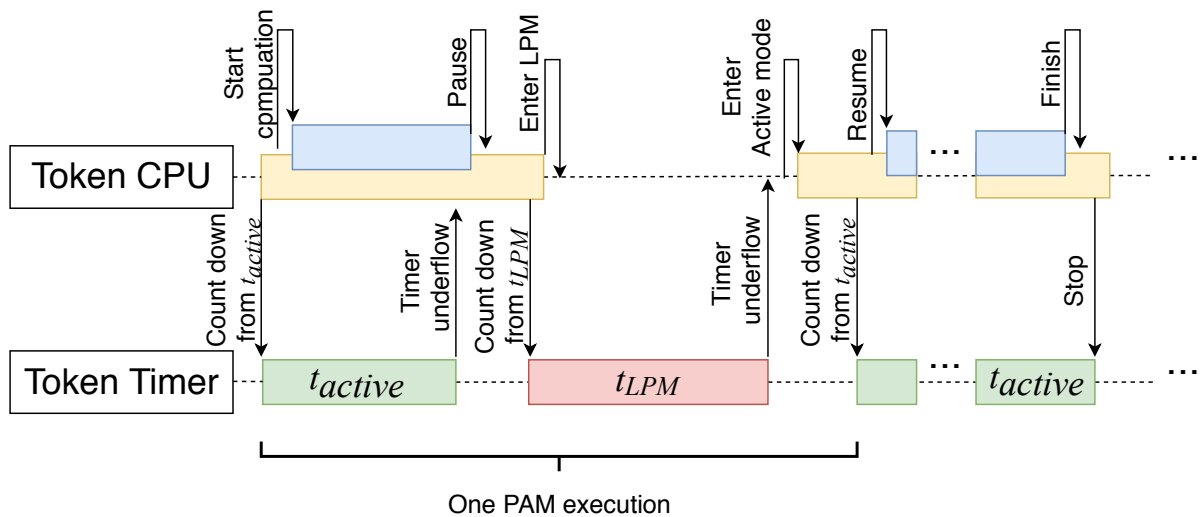


Figure 4.3: A sequence diagram describing our proposed PAM and illustrating the interaction between the Token’s CPU execution and the hardware timer. One PAM execution cycle consists of an active mode followed by a low power mode (LPM), one complete task may involve multiple PAM cycles. Other operations, such as RFID communications and FRAM access, are coordinated by the token CPU. If the CPU is in the LPM state, the entire system will be halted to allow energy to accumulate.

Power Aware Execution Model (PAM). The execution mode enables a token to dynamically switch between *active* power mode and lower power mode (LPM)—LPM preserves (SRAM) state and avoids power-loss while executing a task. PAM is illustrated in Figure 4.3. In active power mode, the token executes computations, and switches to LPM before power-loss to accumulate energy; subsequently, the token is awoken to active power mode to continue the previous computation after a period of t_{LPM} .

Our PAM model builds upon [38], [162] in that, these are designed for execution scheduling to *prevent power-loss from brownouts*. Compared to [38], we consider dynamic scheduling of tasks and in contrast to [162] sampling of the harvester voltage (*only possible in specific devices*) within the application code, we consider dynamic scheduling determined by the

4.3 On-device Security Function Engineering

more resourceful Server \mathcal{S} using a single voltage measurement reported by a token \mathcal{T} (see Appendix D.3 for detailed comparison). We outline the means of achieving our PAM model on a token below.

During the Security Association stage shown in Figure 4.1, a token measures and reports the voltage $\mathbf{Vt}_i \leftarrow \text{SNIFF}(t)$ to the Server. The \mathbf{Vt}_i measurement indicates energy that can be harvested by token T_i under the settings of the current firmware update session. According to \mathbf{Vt}_i , the Server determines the active time period t_{active} and LPM time period t_{LPM} for each CRFID device (detailed development of a model to estimate t_{active} and t_{LPM} from \mathbf{Vt} is in Appendix D.4). *Consequently, each CRFID device's execution model is configured by the Server with device specific LPM and active periods at run-time. Hence, an adaptive execution model customised to the available power that could be harvested by each CRFID device is realised. Notably, this execution scheduling task is outsourced to the resourceful Server.*

In this context, we assume the distances between the antenna and target CRFID devices are relatively constant during the *short* duration of a firmware update. Firmware updates are generally a maintenance activity where CRFID integrated components are less likely to be mobile to ease maintenance, such as during the scheduled maintenance of an automated production line [183]; night time updates in smart buildings when people are less likely to be present and facilities are inoperative [184]; or the pre-flight maintenance or inspection of aircraft parts while parked on an apron [24]. Further, it is desirable to maintain a stable powering channel in practice by ensuring a consistent distance during the maintenance or patching of devices for a short period. This is a more reasonable proposition than the wired programming of each device. Hence, the relatively fixed distance is a reasonable assumption in practice. Notably, given the challenging nature of the problem, previous non-secure firmware update methods, such as R^3 [32] and Stork [22], were evaluated under the same assumption.

PAM Experimental Validation. To understand the effectiveness of our PAM method to reduce the impact of brownout, we execute a computation intensive module, $\text{MAC}()$ using PAM, at power-up on a CRFID. We employ a few lines of code to toggle GPIO pins to indicate the successful completion of a routine. Notably, it is difficult to track a device's internal state without a debug tool attached to the device; however, if the debug interface is in use, it will either interfere with powering, or affect the timing by involving additional Joint Test Action Group (JTAG) service code. We measure the time taken to complete the $\text{MAC}()$ execution (latency) and success rate (success over 10 repeated attempts under wireless powering conditions) with digital storage oscilloscope connected to GPIO pins indicating a successful execution before power loss. We compute a MAC over a 1,536-Byte randomly generated message to test the effectiveness of PAM

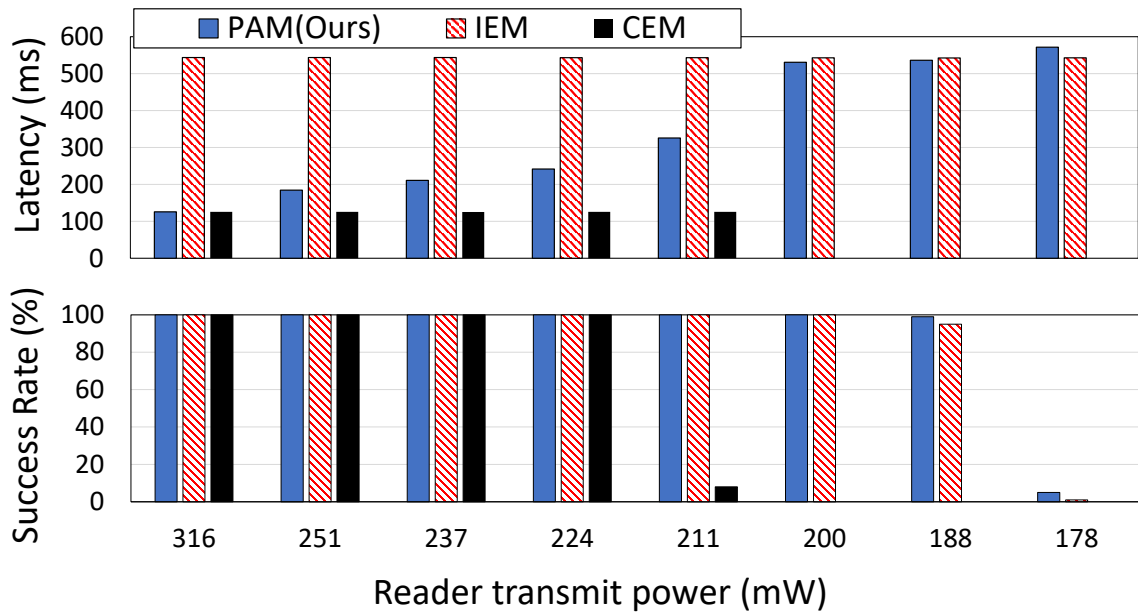


Figure 4.4: Experimental evaluation of the latency and success rate of our proposed power aware execution model (PAM) compared to the Intermittent Execution Mode (IEM) method in [38] and a typical continuous execution model (CEM). The long-run task used in this evaluation is a MAC computation over a 1,536-Byte random message typically requiring 125.5 ms to complete in CEM. Notably, CEM fails when the reader transmit power is below 200 mW. We conduct 100 repeated measurements and report the mean.

in preventing a power loss event due to brownout and compare performance with the execution method—denoted IEM—of a fixed LPM state (30 ms) *programmatically* encoded during the device provisioning phase as in [38].

The results in Figure 4.4 show the effectiveness of PAM to mitigate the interruptions from power loss. This is evident when the success rate results without PAM—using the continuous execution model (CEM)—is compared with those of PAM at decreasing operating power levels. However, as expected, we can also observe that the dynamically adjusted execution model parameters (t_{LPM} and t_{active}) of PAM at decreasing power levels to prevent power loss events and increase latency or the time to complete the routine. When PAM is compared with IEM, the dynamically adjusted execution model parameters allow PAM to demonstrate an improved capability to manage interruptions from power-loss at poor powering conditions; this is demonstrated by the higher success rates when the reader transmit power is at 188 mW and 178 mW. Since IEM encodes operating settings *programmatically* during the device provisioning phase [38], we can observe increased latency at better powering conditions when compared to PAM which allows a completion time similar to that obtained from CEM when power is ample. Thus, PAM provides a suitable compromise between latency and successful completion of a task at different powering conditions.

4.3 On-device Security Function Engineering

Notably, the RFID media access control (MAC) layer on a CRFID device is implemented in software as assembly code and executed at specific clock speeds to ensure strict signal timing requirements in the *EPC Gen2* air interface protocol. Additionally, protocol message timing requirements places strict limits on waiting periods for devices responses. Hence, we do not consider the direct control of the execution mode during communication sessions for managing power consumption and instead rely on the Pilot-Observer mode method we investigate next.

Table 4.2: Execution overhead of pilot and observer tokens when receiving broadcast packets.

	Clock Cycles ¹	Memory Usage (Bytes)		Operation
		ROM	RAM	
Pilot token receiving	23,082	12	2	apply decoding, prepare reply (e.g., compute CRC)
Pilot token replying	1,131	0	0	communication (backscattering)
Observer token receiving	22,002	12	2	apply decoding

¹Numbers are collected using a JTAG debugger where the reported values are averages over 100 repeated measurements. We provide a detailed discussion of the experiments and analysis process in Appendix D.7.

Pilot-Observer Mode.

We observed and also confirmed in Figure 4.2 that the task of responding to communication commands will likely cause power loss. During the Secure Broadcast stage shown in Figure 4.1, communication is dominated by repeated `SecureComm` commands with payloads of encrypted **firmware** and the data flow is uni-directional. Intuitively, we can disable responses from all the tokens to save energy. However, the `SecureComm` command under *EPC Gen2* requires a reply (ACK) from the token to serve as an acknowledgement [181]. An absent ACK within 20 ms will cause a protocol timeout and execution failure.

To address the issue, we propose the pilot-observer mode inspired by the method in Stork [22]. A critical and distinguishing feature of our approach is *the intelligent election of a pilot token from all tokens to be updated*—we defer to Appendix D.5 for a more detailed discussion on the differences. As illustrated in Figure 4.5(a), our approach places all in-field tokens to be updated into an *observer* mode except one token *elected by the Server* to drive the *EPC Gen2* protocol—this device is termed the *Pilot* token. By doing so, observer tokens process all commands such as `SecureComm`—ignoring the handle designating the target device for the command—whilst remaining silent or muting replies to *all* commands whilst in the *observer* mode. Muting replies significantly reduce energy consumption and disruption to power harvesting of the tokens.

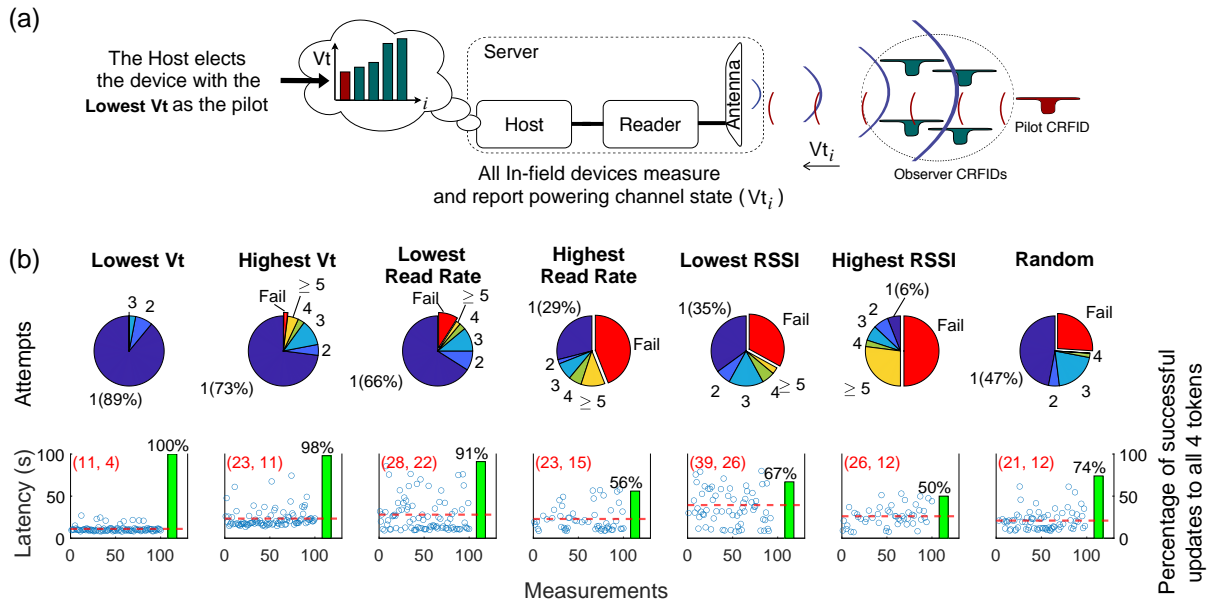


Figure 4.5: (a) Our proposed Pilot-Observer method. Only the elected *Pilot* using the V_{t_i} based election method responds while *observers* listen silently; (b) Evaluation of Pilot Token selection strategies. Measured Tokens are placed 40 cm above the reader antenna, where the powering condition is critical; we defer the reader to Appendix D.6 for further results from other operational distances. The number of attempts to have all 4 CRFIDs (Tokens) updated (pie chart) and the corresponding latency (scatter plot) over 100 repeated measurements. The mean number and the standard division for successful updates for all four tokens are also labeled in the scatter plot with (mean, std) given in seconds.

We propose *electing* the token with the *lowest* reported V_t as the pilot based on the observation: measuring powering channel state from the token, obtained from $V_{t_i} \leftarrow \text{SNIFF}(t)$, is the most reliable measure of power available to a given device (see the discussion in Appendix D.4).

We recognise that it is very difficult to implement an explicit synchronisation method to ensure the observer tokens stay synchronised with the pilot token in the secure broadcast update session. This is due to the level of complexity and overhead that an explicit method would bring and the consequence of such an overhead would be the increased occurrence of failures of a secure broadcast update session due to the additional task demands on tokens—we discuss the problem further in Section 4.7. Although synchronisation between the pilot and observer tokens are not explicit, the Pilot-Observer method implicitly enforces a degree of synchronicity. The selected pilot token—tasked with responding to the Host commands during the broadcast—has to spend more time than observer tokens to prepare the uplink packet and send a reply (ACK) to the Host (RFID reader) to meet the *EPC Gen2* specification requirements as summarised in Table 4.2; hence the update process is controlled by the *slowest* token. Further, since we elect the token with the lowest powering condition as the pilot, the update is controlled by the most energy-starved

4.3 On-device Security Function Engineering

token. This strategy leads to other tokens having higher levels of available power and extra time to successfully process the broadcasted firmware and remain synchronised with the update process. Hence, *if the pilot succeeds, observers are expected to succeed*.

Validation of Pilot Election Method. To demonstrate the effectiveness of our pilot token election method, we considered *seven* different pilot token selection methods based on token based estimations of available power and indirect methods of estimating the power available to a token by the server: 1) Lowest V_t : this implies the pilot is selected based on the most conservative energy availability at token; 2) Highest V_t : the pilot has to reply to commands, hence we may expect higher power availability to prevent the failure of a broadcast session due to brownout at the pilot; 3) Lowest Read Rate: a slow pilot allows observers to gather more energy during the broadcast and remain synchronised with the protocol to prevent failure of the observer tokens; 4) Highest Read Rate: a faster pilot could reduce latency; 5) Lowest Received Signal Strength Indicator (RSSI): successful communication over the poorest channel may ensure all observers are on a better channel (RSSI acts as an indirect measure of the powering channel at a token); 6) Highest RSSI: prevent pilot failure due to a potentially poor powering channel at the pilot token; and 7) Random selection: *monkey beats man on stock picks*.

We have extensively evaluated and compared all of the above seven pilot token selection methods. The experiment is conducted by placing 4 CRFID tokens at 20 cm, 30 cm, 40 cm and 50 cm above an reader antenna; each CRFID is placed in alignment with the four edges of the antenna (See Figure 4.6). We repeated the code dissemination process *100 times* at each distance test, for each of the 7 different pilot selection methods. We recorded two measures: i) latency (seconds); and ii) number of times the broadcast attempt updated all of the 4 in-field devices.

As illustrated in Figure 4.5.(b), at 40 cm where the power conditions are more critical at a token, the selected pilot token with the lowest token voltage (V_t) shows an enormous advantage, in terms of both latency and attempt number. Overall, our proposed approach (electing the lowest V_t) ensures that most observer tokens are able to correctly obtain and validate the firmware in a given broadcast session on the first attempt in the critical powering regions of operation (we provide a theoretical justification in Appendix D.4).

We also evaluated the reduction in power consumption achieved from computational tasks, in addition to eliminating the communication task. While we provide a detailed discussion of the experiments and analysis process in Appendix D.7, we summarise our experimental results in Table 4.2. The measurements show that the observer tokens, compared to the pilot token,

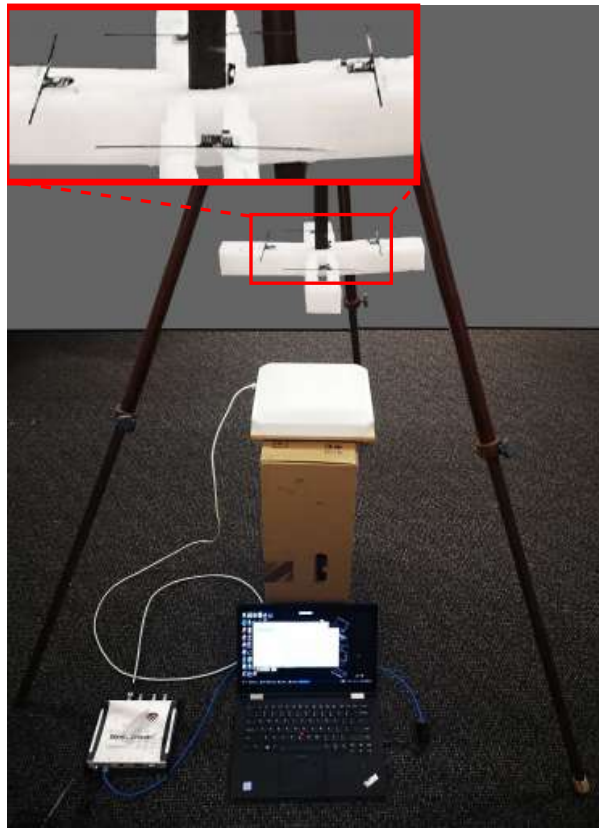


Figure 4.6: Experiment setup showing: ① a host PC, ② a RFID reader, ③ a reader antenna and ④ four CRFID devices mounted on a foam frame, supported by a wooden tripod.

consumes 2,211 less clock cycles *per* firmware packet from the Host. This is a reduction of 9.13% to process each firmware packet sent from the Host.

Summary. *Our pilot-election based method effectively reduces the chance of power failure as well as de-synchronisation of the observer tokens during a firmware broadcast session. Our approach is able to ensure that more of the observer tokens are able to correctly obtain and validate the firmware during a single broadcast sessions.*

4.3.3 Efficient Security Primitives

Wisecr requires two cryptographic primitives: i) symmetric key primitive SKP(); and ii) a keyed hash primitive for the message authentication code MAC(). When selecting corresponding primitives in the following, two key factors are necessary to consider:

1. *Computational cost:* The CPU clock cycles required for a primitives quantifies both the computation cost and energy consumption. Therefore, clock cycles required for executing

4.3 On-device Security Function Engineering

the primitives need to be within energy and computational limits of the energy harvesting device.

2. *Memory needs*: On-chip memory is limited—only 2 KiB of SRAM on the target MCU—and must be shared with: i) the RFID protocol implementation; and ii) user code.

Symmetric Key Primitives. We first compare block ciphers via software implementation benchmarks—clock cycle counts and memory usage—on our target microcontroller [185], [186]. We also considered the increasingly available cryptographic co-processors in microcontrollers: our targeted device uses an MSP430FR5969 microcontroller and embeds an on-chip *hardware* Advanced Encryption Standard (AES) accelerator (we refer to as HW-AES) [187]. Based on the above selection considerations, HW-AES is confirmed to outperform others. Specifically, HW-AES to encrypt/decrypt a 128-bit block consumes 167/214 clock cycles with a power overhead of 21 $\mu\text{A}/\text{MHz}$; therefore, we opted for HW-AES for SKP.Dec function implementation on our target device. We configured AES to employ the Cipher-Block Chaining (CBC) mode¹¹ similar hardware AES resources can be found in a variety of microcontrollers, ASIC, FPGA IP core and smart cards. In the absence of HW-AES, a software AES implementation, such as tiny-AES-c can be an alternative.

Message Authentication Code. MACs built upon BLAKE2s-256, BLAKE2s-128, HWAES-GMAC and HWAES-CMAC on our targeted MSP430FR5969 MCU were taken into consideration [38]. We selected the 128-bit *HWAES-CMAC*—Cipher-based Message Authentication Code¹²—based on AES since it yielded the lowest clock cycles per Byte by exploiting the MCU’s AES accelerator (HW-AES).

4.3.4 Bootloader Control Flow

We can now realise an uninterruptible control flow for an immutable bootloader built upon on the security properties identified and engineered to achieve our Wisecr scheme described in Section 4.2.2. We describe the bootloader control flow in Figure 4.7.

At power-up, ① the token performs MCU initialisation routines, ② setup MPU protections for bootloader execution and carries out a self-test to determine whether a firmware update

¹¹We are aware of CBC padding oracle attacks; however, in our implementation, the response an attacker can obtain is to the CMAC failure or success and not the success or failure of the decryption routine. Alternatively, the routines can be changed to employ authenticated encryption in the future, with an increase in overhead.

¹²The implementation in NIST Special Publication 800-38B, *Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication* is used here.

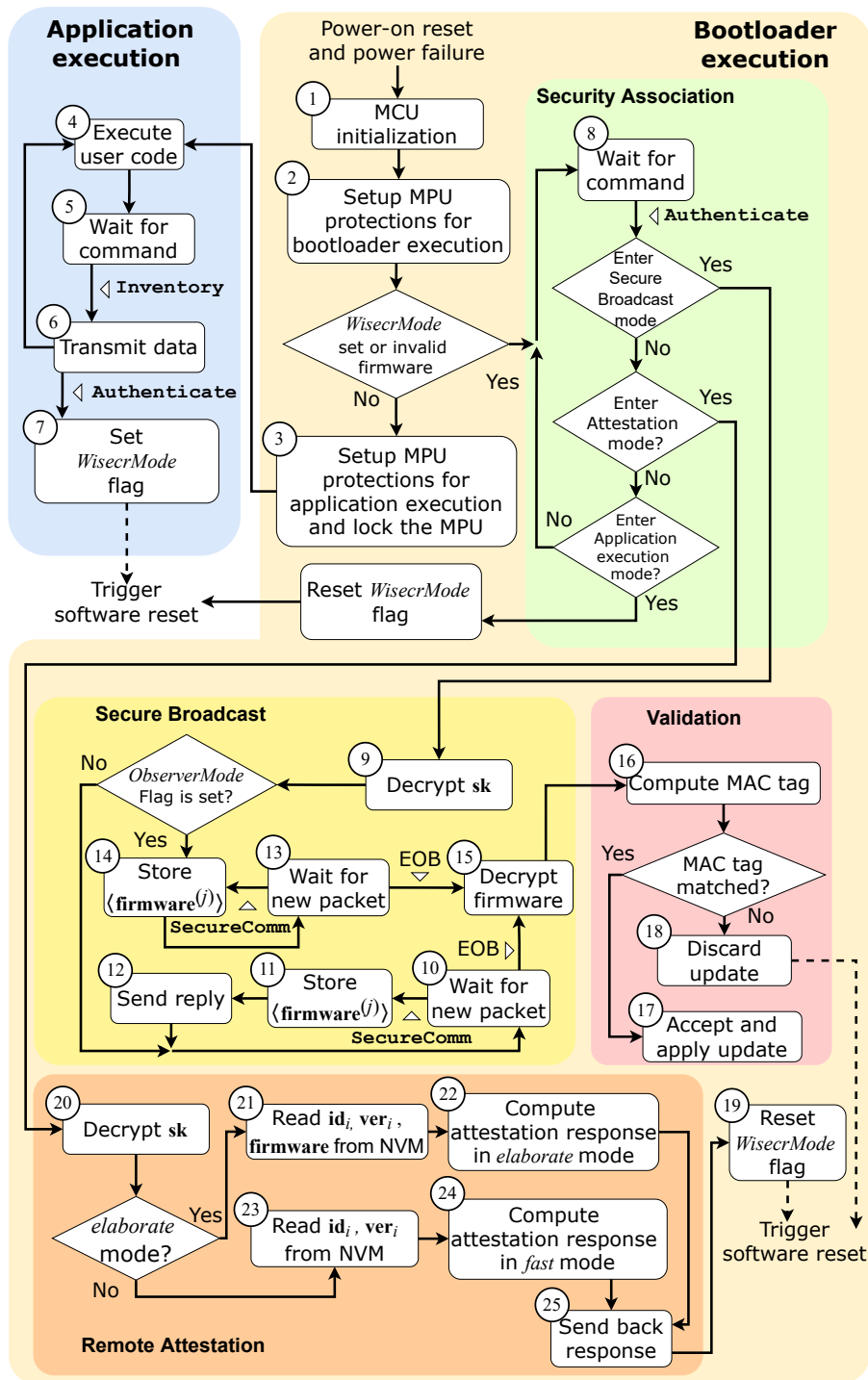


Figure 4.7: Wisecr Bootloader control flow. The Bootloader manages Security Association, Secure Broadcast, Validation and Remote Attestation stages.

is required, or if there is a valid application installed. If no update is required and the user application is valid, (3) the bootloader configures MPU protections for application execution, before handing over control to the application. During the application execution, (4) the user code is executed, (5) then the token T waits for command from the Server S , the Server S may

4.3 On-device Security Function Engineering

send an `Inventory` commands to instruct the token T to (6) send back e.g., sensed data, or in (7) setup the `WisecrMode` flag and trigger a software reset in preparation for an update.

Security Association. Upon a software reset, a set `WisecrMode` flag directs the token T to enter the Security Association stage. (8) Subsequently, the token waits for further instructions from the Server. On reception of an `Authenticate` command, carrying an encrypted broadcast session key $\langle \mathbf{sk} \rangle_i$ and the MAC tag s_i of the new firmware, (9) the token T decrypts $\langle \mathbf{sk} \rangle_i$ with its device key \mathbf{k}_i and acquires the session key \mathbf{sk} .

Secure Broadcast. Recall, at this stage, all tokens selected for update will be switched into the observer mode except the pilot token that is set to respond to the Server. The pilot token in state (10) receives a new chunk of encrypted firmware $\langle \mathbf{firmware}^{(j)} \rangle$, and (11) stores it into the specified memory location. (12) The pilot token sends reply to the Server. In (13), for tokens in observer mode, they silently listen to the communication traffic between the Server and the Pilot token without replying. In other words, tokens under observer mode receives the encrypted firmware chunks, (14) store chunks in memory but ignores unicast handle identifying the target device; this significantly saves observers' energy from replying as detailed in Section 4.3.2. Once an End of Broadcast (`EOB`) message is received, all tokens stop waiting for new packets and start firmware decryption (15).

Validation. The token computes a local MAC tag s'_i , including the decrypted firmware, and compares it with the received MAC tag s_i (16). The firmware is accepted, applied to update (17) the token, if s'_i and s_i are matched, and (19) the `WisecrMode` flag is reset; otherwise, (18) the firmware is discarded and the update is aborted. Subsequently, each token performs a software reset to execute the new firmware or reinitialise in bootloader mode if the firmware update is unsuccessful.

Remote Attestation. On reception of an `Authenticate` command with an instruction to perform remote attestation, (20) the token first decrypts the session key \mathbf{sk} and reads (21) \mathbf{id}_i , \mathbf{ver}_i and $\mathbf{firmware}$ from the NVM according to the specified memory address and size (firmware is only used in the *elaborate* mode, in the *fast* mode (23) only \mathbf{id}_i and \mathbf{ver}_i are involved). The attestation response is then computed as $\mathbf{r} \leftarrow \text{MAC}_{\mathbf{sk}}(\mathbf{c} \parallel \mathbf{firmware} \parallel \mathbf{id}_i \parallel \mathbf{ver}_i)$ if the *elaborate* mode is selected (22) or $\mathbf{r} \leftarrow \text{MAC}_{\mathbf{sk}}(\mathbf{c} \parallel \mathbf{id}_i \parallel \mathbf{ver}_i)$ if using the *fast* mode (24). Subsequently, the response \mathbf{r} is sent back to the Server S , (25) the `WisecrMode` flag is cleared and (19) the device is reset.

Notably, a power-loss event during the control flow will result in a reset and rebooting of the token T and a transition out from the firmware update state. Most significantly, in such an

occurrence, the immutable bootloader functionality is preserved. Although the loss of state implies that a Server S must reattempt the secure update, we prevent the token from being left in a vulnerable state to inherently mitigate security threats posed in the power *off* state [34].

4.4 End-to-end Implementation and Experiments

This section elaborates on software tools, the end-to-end implementation of Wisecr and extensive experiments conducted to systematically evaluate the performance of Wisecr.

4.4.1 End-to-end Implementation

We implement a bootloader and a *Server Toolkit* to wirelessly and simultaneously update multiple CRFID transponders using *commodity* networked RFID hardware and *standard* protocols. We realise the Wisecr scheme in Figure 4.1 and the identified security property requirements. Specifically, we employ MSP430FR5969 MCU based WIPS5.1LRG CRFID devices. The MSP430FR5969 microcontroller has a 2 KiB SRAM and a 64 KiB FRAM internal memory. *The implementation is a significant undertaking* and includes software development for the CRFID as well as the RFID reader and backend services for the server update mechanisms. Given that we have open sourced the project, we describe: i) the *bootloader* in Section 4.4.2; and ii) the Sever Toolkit and the protocols for Host-to-RFID-reader and Reader-to-CRFID-device communications for implementing Wisecr over *EPC Gen2* in Appendix D.2.

4.4.2 Bootloader and Memory Management

Wisecr Bootloader. The immutable bootloader supervises firmware execution while needing to be compatible with standard protocols, we developed our bootloader based on the Texas Instrument’s recent framework for wireless firmware updates—*MSP430FRBoot* [141] —and the code base from recent work [38] employing the framework. Such construction ensures a standard tool chain for compilation and update of new firmware whilst the usage of industry standards is likely to increase adoption in the future. We compile the firmware code into ELF (Executable and Linkable Format) made up of binary machine code and linker map specifying the target memory allocation—the memory arrangement is illustrated Figure 4.8.

Wisecr Memory Management. For implementing the Secure Storage component, several different mechanisms are available (as summarised in Appendix D.1). In Wisecr, we select

4.4 End-to-end Implementation and Experiments

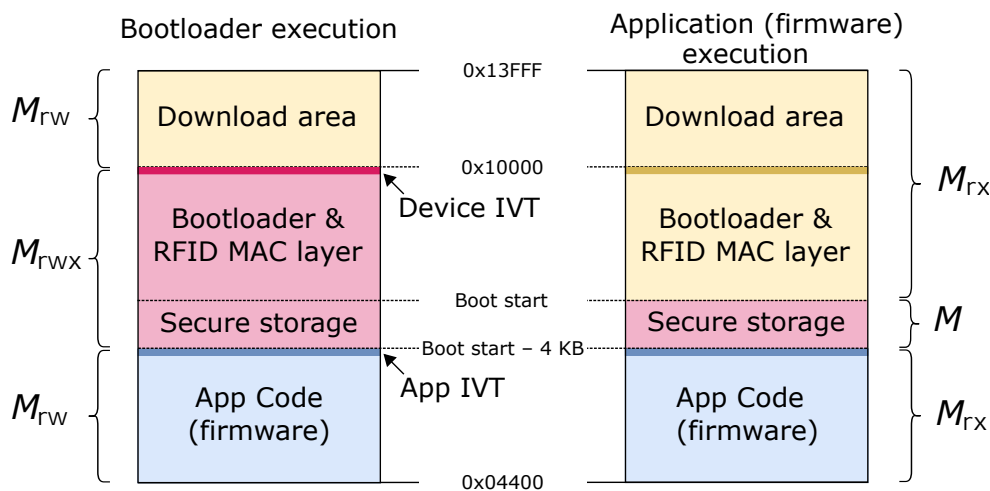


Figure 4.8: Memory Protection Unit (MPU) segmentation diagram and memory arrangement during bootloader execution (left) and application code execution (right). Due to hardware limitations, only 3 segments can be defined, and the secure storage area must be at least 1 KiB in size due to segment boundary alignment requirements [188]

to use the *Runtime access protection* (e.g. using MPU segments at run-time) In this scheme, secure storage is available on device boot-up, but is locked (until next boot-up) by the bootloader before any application code is executed. We employed this method and the implemented memory arrangement is described in Figure 4.8. Switching to privilege mode is done by the bootloader on boot up, and the applications do not need to make any specific modifications. If an application attempts to access a privileged resource, e.g. overwriting the bootloader segment, a power-up clear (PUC) will be triggered by the MPU to cause reboot, before any risky operation.

Importantly, any power-on reset, regardless of the cause, will result in a reboot and the execution of the bootloader before transferring control to application code—see Application execution stage in Figure 4.7. Prior to entering the application execution stage, the bootloader enforces the MPU policy for application execution (see ③). This step is necessary with the target MCU used in our implementation as it only allows defining three memory segments for protection. Thus, we are able to rely on the very limited protections provided by the MCU to achieve the security objectives of a trusted bootloader; recall, in our threat model, the bootloader is trusted, and we realise this in practice by ensuring that the bootloader provisioned is immutable where the secrets stored on device must remain inaccessible and immutable to the application firmware. Now, it is infeasible to revoke the policy enforced during the application execution stage, unless the device is power-cycled. But, the device will enter the bootloader stage after a reboot and MPU protections will be re-enabled prior to executing any application code. We summarise the memory protections developed using the limited MPU configurations to realise Wisecr security requirements in bootloader and application execution modes in Section 4.3.2; therein, we also

discuss alternative methods to realise such protections to ensure the generalisation of Wisecr to other MCUs.

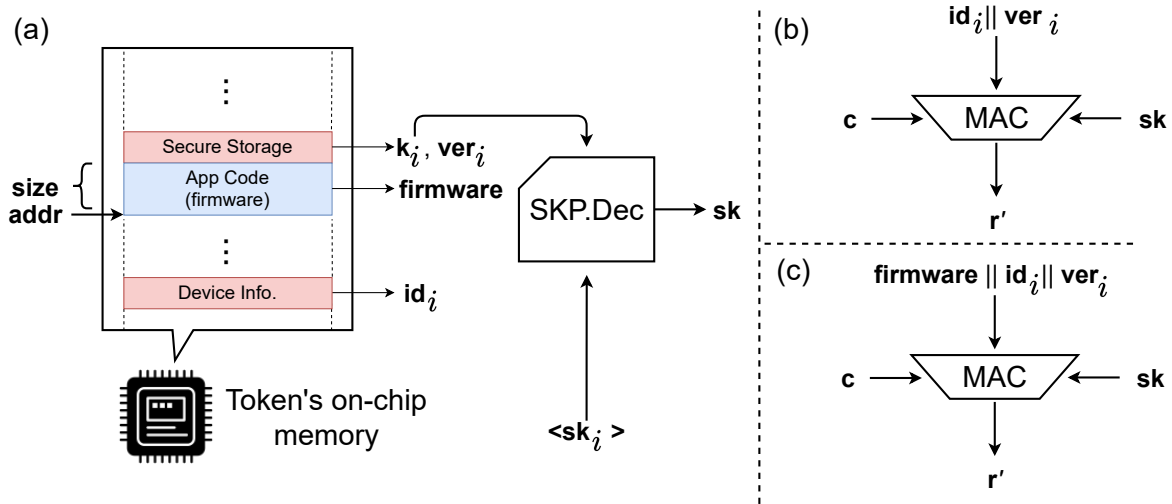


Figure 4.9: (a) Token memory and allocations; (b) *fast* mode; and (c) *elaborate* mode.

Table 4.3: Wisecr Implementation and Execution Overheads. Tested under the CCS 9.0.1.0004 development environment, with compiler TI v18.12.1.LTS Optimisation settings: $-O = 3$; $-opt_for_speed = 5$.

	Clock Cycles	Memory Usage (Bytes)		HW Requirement
		ROM	RAM	
Functional blocks				
$sk \leftarrow SKP.Dec_{k_i}(\langle sk_i \rangle)$ (For a 16-Byte block)	772	706 ²	62	HW-AES
$firmware^{(j)} \leftarrow SKP.Dec_{sk}(\langle firmware^{(j)} \rangle)$ (For a 16-Byte block)	772	690	62	HW-AES
$s_i \leftarrow MAC_{sk}(firmware ver_i nver)$	26,698 ¹	2,374	72	HW-AES
$Vt_i \leftarrow SNIFF(t)$	1,530	276	12	ADC
PAM.Enter(t_{LPM_i} , t_{active_i}) and PAM.Exit() ³	51	86	6	Timer
Setup MPU for bootloader execution	76	58	0	MPU
Setup MPU for application execution	91	60	0	MPU
Secure Storage (used for k_i , id_i and ver_i)	0	19	0	MPU
Stages				
Security Association	2,302	982	74	ADC, HW-AES
Secure Broadcast	11,604 ¹	760	68	HW-AES, Timer
Validation	26,724 ¹	2,390	77	HW-AES, Timer
Remote Attestation (<i>elaborate</i> mode)	18,360 ⁴	3,172	80	HW-AES
Remote Attestation (<i>fast</i> mode)	5,574	3,172	80	HW-AES
Total				
Wisecr (excluding Remote Attestation, for the pilot Token)	40,630 ¹	3442	154	All of the above
SecuCode (using fixed key, updates a single token)	27,092 ¹	2442	78	HW-AES, Timer

¹For a typical 240-Byte firmware

²16 Bytes of this value incorporates the device key k_i in secure storage

³A single PAM execution, as defined in Figure 4.3

⁴4,397 clock cycles for the setup routine—a constant overhead for any block size, 1,060 clock cycles for each 16-Byte block and $n + 2$ clock cycles for an n -Byte padding to form a 16-Byte block, if required.

Remote Attestation. As shown in Figure 4.9 (a), Remote Attestation routine is an integral part of the immutable bootloader, it has access to all memory segments: Token device key k_i , version number ver_i , tag serial number id_i and the installed user application code **firmware**. A challenge c is a one-time use random string that is generated by the Server S . The r' is the report to be transferred back to the Server. On receiving an encrypted session key $\langle sk_i \rangle$, the Token decrypts

4.4 End-to-end Implementation and Experiments

it with SKP.Dec and obtains the session key \mathbf{sk} . In the *fast* mode illustrated in Figure 4.9(b), the response \mathbf{r}' is calculated based on \mathbf{id}_i and \mathbf{ver}_i . In contrast, the response \mathbf{r}' is computed over an entire memory segment, such as the **firmware**, in the *elaborate* mode as depicted in Figure 4.9 (c).

4.4.3 Wisecr Implementation Overheads

We first evaluate the execution and implementation overhead of each Wisecr *security functional block*; results¹³ are summarised in Table 4.3. The execution overhead is measured in terms of clock cycles for installing a firmware of 240 Bytes. The implementation overhead is measured in memory usage, consisting of ROM for code and constants and RAM for run-time state. Secondly, we assess the necessary hardware modules such as hardware AES accelerator (HW-AES), analog-to-digital converter (ADC) and Timer. All functional blocks are implemented in platform independent C source files. We summarise the implementation costs for each stage; the most significant implementation and execution overhead is from the Validation stage because of the computationally heavy MAC() function—recall Remote Attestation is an optional stage.

In Table 4.3, we also compare with SecuCode¹⁴; presently, the only CRFID firmware update scheme considering security—notably, the scheme only prevents malicious code injection attacks and is designed to update a single device at a time as highlighted in the method comparison Table 4.4. In summary, *Wisecr* consumes 49.97% more clock cycles, 40.95% more ROM and 97.43% more RAM space than the SecuCode. However, *Wisecr* can update multiple devices simultaneously to attain a performance advantage as we demonstrate in Section 4.4.3 because SecuCode only updates one device at a time. In addition, Wisecr provides IP protection (encryption of the firmware transmitted over the insecure wireless channel) and validates firmware installation. As expected, the performance improvements and additional security features are achieved at an increased implementation overhead.

¹³Tested under the CCS 9.0.1.0004 development environment, with compiler TI v18.12.1.LTS Optimisation settings: `-O = 3; -opt_for_speed = 5`.

¹⁴In [38], the device key is derived from SRAM PUF responses. In our comparison, we opt for a device key in the protected NVM instead to make a fair overhead comparison. Further, the key derivation module only adds a fixed overhead at start-up.

4.4.4 Experiment Designs and Evaluations

We employed four WIPS5.1LRG CRFID devices and an Impinj R420 RFID reader with a 9 dBic gain antenna to communicate with and energise the CRFID devices. All of the experiments are conducted with the RFID reader feeding the antenna with 30 dBm output power. Given that, the recent *EPC Gen2 V2.0* [181] security commands such as the `SecureComm` are not yet widely supported by commercial RFID readers, like the Impinj Speedway R420, we map the unsupported *EPC Gen2* command to the `BlockWrite` command as in [38]. We summarise our evaluations below¹⁵:

- We evaluate performance—latency and throughput in Section 4.4.5—and compare with: i) `SecuCode` [38], the scheme only prevents malicious code injection attacks and is designed to update a single device at a time as shown in the method comparison Table 4.4; ii) `Wisecr` intentionally set to sequential mode—broadcasting/updating one token at a time—termed `Wisecr (Seq)`, to assess the gains from `Wisecr` broadcasting firmware to multiple devices simultaneously; and iii) non-secure multi-CRFID dissemination scheme, `Stork` [22] (see Section 4.5.1)
- We evaluate the efficacy of our Power Aware Execution Model (PAM) and Pilot Token selection method (results in the Section. 4.3.2).
- We demonstrate `Wisecr` in an end-to-end implementation, see Section 4.4.6 and demonstration video at:

<https://www.youtube.com/watch?v=GgDHPJi3A5U>



scan to watch

4.4.5 Performance Evaluation and Results

We use two performance metrics: i) end-to-end *latency*; and ii) *throughput* (bits per second). Latency is the time elapsed from the Server Toolkit transmitting the firmware to an RFID reader using LLRP commands to the time the Server Toolkit confirms the acknowledgement of successfully updating *all chosen tokens*. Throughput is the total data transferred over the latency, where $Throughput (bps) = (|firmware| \times Num. of Tokens) / Latency$. Our evaluation is carried out under three different controlled variables: i) distance; ii) number of tokens; and

¹⁵Notably, our extensive experiments to evaluate techniques and modules we developed are detailed in the Appendices with references in the main article

4.4 End-to-end Implementation and Experiments

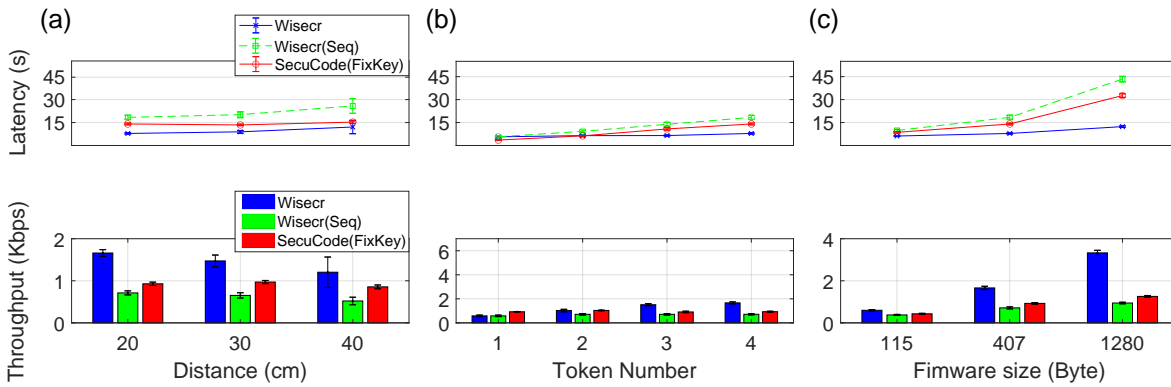


Figure 4.10: Comparing Wisecr, Wisecr (Seq)—Wisecr operating in sequential mode, SecuCode [38] *with no secure broadcast* support under increasing: (a) Distance (4 Tokens with a 407 Byte firmware); (b) Number of Tokens (at 20 cm with 407 Byte firmware); and (c) Firmware size (at 20 cm with 4 Tokens).

iii) firmware size. In each experiment, only one variable is changed and our results are collected over 100 repeated measurements, with outliers outside of 1.5 times upper and lower quarterlies removed.

First, we transfer the same firmware code (a 407 Byte accelerometer service) to four target devices located at distances from 20 cm to 40 cm, covering good to poor powering channel states, to evaluate the *stability* of the Wisecr scheme. As expected, we can observe in Figure 4.10(a) that the performance of all three schemes downgrades with increasing distance. Notably, Wisecr and SecuCode outperforms the Wisecr (Seq). Because, Wisecr (Seq) incurs a larger overhead through the need for executing a Security Association stage to setup the broadcast security parameters, for each device. Further, SecuCode (without IP protection) does not need to execute the power intensive firmware decryption operations, and thus is less likely to encounter power-loss and update session failure or require the extra time necessary by Wisecr to decrypt the firmware. As expected, Wisecr outperforms albeit the added security functions provided in comparison to SecuCode.

Second, following the method and metrics in Stork [22], we tested the performance of the schemes by increasing the number of CRFID devices to be updated. We can see in Figure 4.10(b) that the latency of both Wisecr (Seq) and SecuCode increases linearly while Wisecr remains relatively steady regardless of the number of devices to update. Consequently, Wisecr exhibits significantly improved throughput as the number of device increases. Further, we examine performance under increasing firmware size: 115 Bytes; 407 Bytes (a sensor service code); and 1280 Bytes (a computation code firmware). Efficacy of broadcasting to multiple devices simultaneously is validated by latency and throughput performance detailed in Figure 4.10(c).

Third, all three candidates show improved efficiency as larger firmware sizes are transmitted; Wisecr’s significantly better efficiency can be attributed to the broadcast method.

We also conducted an evaluation of the execution overhead (latency) introduced by the remote attestation routines. Since remote attestation is performed on one singulated token at an instance, we only examine its impact on one individual device. The latency can be aggregated for multiple devices. As illustrated in Figure 4.11, the remote attestation in *fast* mode always completes in 1.1 ms, regardless of the firmware size. While the time required to complete the *elaborate* mode scales from 1.3 ms to 6.5 ms with respect to firmware sizes from 115 Bytes to 1280 Bytes.

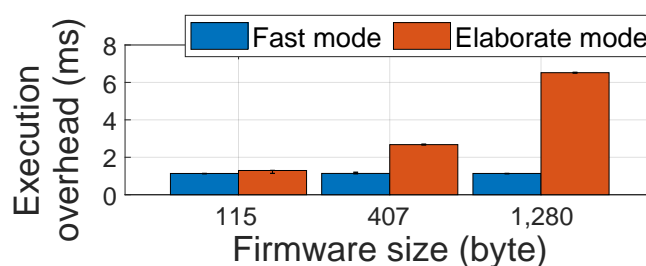


Figure 4.11: The execution overhead of the attestation function with respect to different firmware sizes in an end-to-end experiment setting. Mean values over 100 repeated measurements are plotted.

4.4.6 Case Study

As shown in Figure 4.12, we employ four CRFID devices embedded in a 3D printed unmanned aerial vehicle (UAV) rotor prototype. *The CRFID devices are factory programmed with firmware for monitoring the centrifugal load and the wired programming interface is disabled prior to embedding and deployment. All four CRFID devices are required to be updated wirelessly and securely with code for monitor the blade flapping vibration in a wind tunnel test.*

We employed Wisecr to simultaneously, securely and wirelessly update the four CRFID sensor devices with the 391-Byte firmware code. The experimental results summarised in Figure 4.12 show successful updates for 100 repeated attempts in two settings: i) when the rotor is attached to the UAV Wisecr updated all of the devices in the first attempt in 76% of the time with an average latency of 28.12 seconds; and ii) when the rotor assembly is free standing, Wisecr updated all of the devices in the first attempt in 89% of the time with an average latency of 14.27 seconds¹⁶.

¹⁶Although previous studies have performed experiments with static tags, we also attempted to update the tokens while the rotor blades are mobile by rotating them at approximately 1 RPM. We were able to update tokens over 50% of the time. As expected, the changing powering conditions dramatically reduced the update rate (see details in Appendix D.9)

4.4 End-to-end Implementation and Experiments

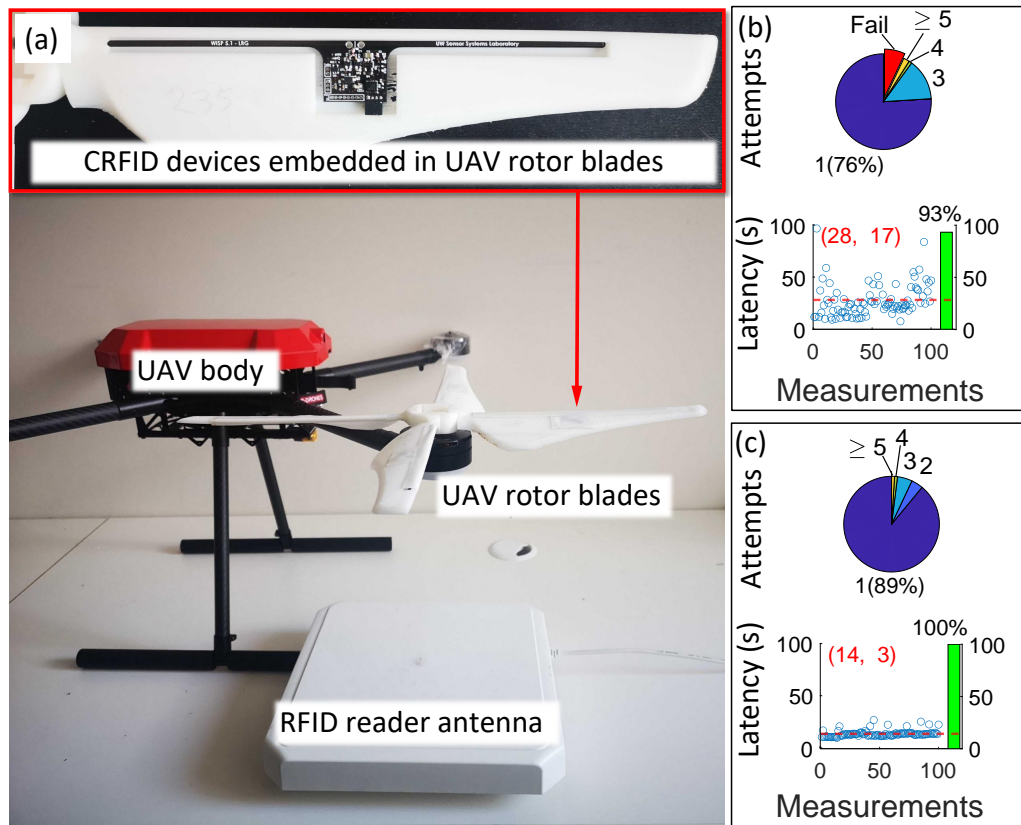


Figure 4.12: (a) Application scenario: wirelessly updating the firmware of four CRFID sensor devices embedded in an unmanned aerial vehicle (UAV) rotor prototype. The UAV rotor aerodynamic model is adopted and modified from: <https://grabcad.com/library/4-bladed-propeller-experimental-1>. With Wisecr, we simultaneously, securely and wirelessly updated the four devices with a 391-Byte code. Experimental results when: (b) the rotor assembly is attached to the UAV and (c) when the rotor assembly is free standing over the antenna. The green bar graph denotes the percentage of successful updates to all 4 devices.

A demonstration of the secure firmware update process in using our end-to-end implementation illustrated in Figure 4.12 is available at:

<https://www.youtube.com/watch?v=GgDHPJi3A5U>



scan to watch

We open source the complete source code for Wisecr and related tools at:

<https://github.com/AdelaideAuto-IDLab/Wisecr>.

4.4.7 Security Analysis

Under the threat model in Section 4.2.1, we analyse Wisecr security against: i) IP theft (code reverse engineering); ii) malicious code injection attacks under code alteration, loading unauthorised code, loading code onto an unauthorised device, and code downgrading; and iii) incomplete code installation.

IP Theft or Code Reverse Engineering. The wirelessly broadcasted firmware is encrypted using a one-time 128-bit broadcast session key \mathbf{sk}_i . Without the knowledge of the broadcast session key, the adversary \mathcal{A} is unable to obtain the plaintext **firmware** through passive eavesdropping. Once the firmware is decrypted on the token, the binaries cannot be accessed outside the immutable bootloader. Therefore, the probability of a successful IP theft is now determined by the security provided by the size of the selected keys—we employ 128-bit keys in the Wisecr implementation.

Loading Unauthorised Code. The security property of a MAC tag $\mathbf{s}_i \leftarrow \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware}||\mathbf{ver}_i||\mathbf{nver})$ ensures that the adversary \mathcal{A} cannot commit a malicious firmware injection attack without the knowledge of the session key \mathbf{sk} —recall that the sharing of \mathbf{sk} is protected through a secure channel established with a token specific private key \mathbf{k}_i . Hence, only the Server with the session key \mathbf{sk} is able to produce a valid MAC tag to a token T . Hence, the probability of launching a successful attack by the adversary \mathcal{A} is limited to the probability of successfully obtaining the device key \mathbf{k}_i and/or the session key \mathbf{sk} in a man-in-the-middle attack to successfully construct a MAC tag and disseminate a malicious firmware that will be correctly validated by the target tokens. Therefore, without knowledge of \mathbf{k}_i or \mathbf{sk} , the probability of fooling a token T to inject a malicious firmware by \mathcal{A} is no more than the brute-force attack probability 2^{-128} determined by the key size $|\mathbf{k}_i| = 128$.

Code Alteration. For each firmware update, a MAC tag $\mathbf{s}_i \leftarrow \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware}||\mathbf{ver}_i||\mathbf{nver})$ is used to verify code integrity. Therefore, without knowledge of the session key \mathbf{sk} , the adversary \mathcal{A} can not generate a valid MAC tag for an altered firmware. Any attempted changes to the firmware during the code dissemination will be detected and, thus, firmware discarded by the token.

Loading Code onto an Unauthorised Device. Each authorised device i maintains a unique and secret device specific key \mathbf{k}_i only known to the Server \mathcal{S} . Therefore, an unauthorised device cannot decrypt the session key \mathbf{sk}_i to install a recorded firmware encrypted with \mathbf{sk}_i of an authorised device.

Code Downgrading. The adversary \mathcal{A} can attempt to downgrade the firmware to an older version to facilitate exploitation of potential vulnerabilities in a previous distribution. This can be attempted through a replay attack by impersonating the Server. However, for each firmware update, a specific MAC tag: $\mathbf{s}_i \leftarrow \text{MAC}_{\mathbf{k}_i}(\mathbf{firmware}||\mathbf{ver}_i||\mathbf{nver})$ is generated based on two monotonically increasing version numbers: \mathbf{ver}_i and \mathbf{nver} . Without direct access to modify the Secure storage contents in region M to re-write the current version number of a device to an

4.5 Related Work and Discussion

older version, the attacker cannot replay a recorded MAC tag from a previous session to force a token to accept an older firmware. By virtue of the monotonically increasing version numbers, and the secure MAC primitive protected by the session key \mathbf{sk} , a successful software downgrade using a replay attack cannot be mounted by the adversary \mathcal{A} .

Incomplete Firmware Installation. In a man-in-the-middle attack, the adversary \mathcal{A} may attempt to prevent a new firmware installation and spoof a device response with an updated version number during an interrogation—the version number is public information and sent in plaintext as shown in Figure 4.1. Hence, a token may be left executing an older firmware version with potential firmware vulnerabilities. The Server can verify that a specific disseminated firmware deployed by the token is the same as the one issued by the Server by performing a remote attestation. In our Wisecr scheme, we have considered an optional remote attestation stage (detailed in Section 4.2.2) to allow the Server to verify the firmware installation on a given token. In general, both code installation and attestation are bounded to the device key \mathbf{k}_i , thus, it is infeasible to fool the Server without knowledge of the device key.

Related-key Differential Cryptanalysis Attack. In our security association stage, the session key \mathbf{sk} is encrypted multiple times with different device keys \mathbf{k}_i . A related-key differential cryptanalysis of round-reduced AES-128 is demonstrated in [189]. The prerequisite to mount such an attack is that the attacker knows or is able to choose a relation between several secret keys and gain access to both plaintext and ciphertext [190]. In our approach, as mentioned in Section 4.2.1, \mathbf{k}_i chosen by the server are *i.i.d.*, and only the ciphertext is publicly available where the attacker is not able to force the server to encrypt a plaintext of their choosing. Therefore, related-key attacks cannot be mounted under our threat model. Even if such an attack is possible, to the best of our knowledge, there is no successful full-round related-key differential cryptanalysis attack against AES-128, which serve as the SKP function in our implementation.

4.5 Related Work and Discussion

Here, we discuss related works in the area of *wireless code dissemination to CRFID devices*. Notably, the topic of wireless code updating has been intensively studied for battery powered Wireless Sensor Network (WSN) nodes. For example, a recent study by Florian *et al.* [142] proposed updating the firmware and peer-to-peer attestation of multiple mesh networked IoT devices. Physical unclonable functions [38] and blockchains [191] have also been utilised in code dissemination. But, such schemes [142], [191] are designed for *battery powered* WSN nodes featuring *device-to-device communication capability*, and *supervisory control*

of operating systems (e.g. TinyOS), absent in batteryless CRFID devices (see challenges in Section 4.1). Further, batteryless devices operate under extreme energy and computational capability limitations. In particular, harvested energy is intermittent and limited and thus devices face difficulties supporting security functions such as key exchange using Elliptic Curve Diffie-Hellman used in [142]. Therefore, we focus on research into wireless dissemination schemes for CRFID like devices in our discussions. Further, we also discuss and compare our work with existing wireless methods developed for CRFID to highlight the aim of our scheme to fulfil unmet security objectives for simultaneous wireless firmware update to multiple CRFIDs.

Wireless Code Dissemination to CRFID. Given the recent emergence of the technology, there exists only a few studies on code dissemination to CRFID devices. *Wisent* from Tan *et al.* [29], and R^3 from Wu *et al.* [32] demonstrate a wireless firmware update method for CRFIDs. Subsequently, Aantjes *et al.* [22] proposed Stork, a fast multi-CRFID wireless firmware transfer protocol. Stork forces devices to ignore the RN16 handle in down-link packets—a form of promiscuous listening—to realise a logical broadcast channel in the absence of *EPC Gen2* support for a broadcast capability; RN16 specifies the designated packet receiver. This technique enables Stork to simultaneously program multiple CRFIDs to achieve *fast* firmware dissemination. Our Pilot-Observer mode (Section 4.3.2) is based on a similar concept, but instead of always selecting the first seen device, as in Stork, we strategically elect the token with the lowest V_t as the *Pilot* to achieve higher broadcast success. Brown and Pier from Texas Instrument (TI) developed *MSPBoot* [175] in late 2016. They demonstrate an update using a UART or SPI bus to interconnect a MSP430 16-bit RISC microcontroller and a CC1101 sub-1GHz RF transceiver.

However, none of the schemes, *Wisent*, R^3 , Stork and the work in [175], considers security when wirelessly updating firmware. SecuCode [38] took the first step to prevent malicious code injection attacks where a single CRFID device is updated. But SecuCode lacks scalability and performs device by device updates, cannot protect the firmware IP, and provides no validation of firmware installation.

Remote Attestation. Attestation enables the Server to establish trust with the token’s hardware and software configuration. Existing mainstream remote attestation methods are unsuitable for a practicable realisation in the context of intermittently powered, energy harvesting platforms we focus on. For example, peer-to-peer attestation adopted in [142] is impractical in the CRFID context due to the lack of a device-to-device communication channel. In boot attestation [192], a public-key based scheme is proposed to fit ownership and third party attestation, however public-key schemes are too computationally intensive for the batteryless devices we consider.

4.5 Related Work and Discussion

Recently, a publish/subscribe mechanism based asynchronous attestation for large scale WSN is presented in SARA [193], however, such a publish/subscribe paradigm is yet to be supported over a standard *EPC Gen2* protocol. In contrast, our lightweight-remote-attestation mechanisms (see Section 4.2.2) are inspired from the recent study in [30] where methods are designed for resource-limited platforms and require no additional building blocks besides those necessary for Wisecr.

Table 4.4: Comparison with Related Studies.

Work	Multiple devices	Power-loss mitigation		Security			Public source code release
		Communication Energy Reduction	Adaptive IEM	Prevent Malicious Code Injection	Prevent IP Theft	Code installation attestation	
Wisent [29]	✗	✗	✗	✗	✗	✗	✓
R ³ [32]	✗	✗	✗	✗	✗	✗	✗
Stork [22]	✓	✓	✗	✗	✗	✗	✓
SecuCode [38]	✗	✗	✗	✓	✗	✗	✓
Wisecr (Ours)	✓	✓	✓	✓	✓	✓	✓

Comparisons. Table 4.4 summarises a comparison of Wisecr with wireless code dissemination studies *specific to passive CRFID devices*. Wisecr is the only *secure* scheme (prevent IP theft and malicious code injection, and provide attestation of firmware installation) for *simultaneous* update of passively powered devices. We have also extensively compared the performance of Wisecr with the *non-secure* code update method of Stork to provide an appreciation for the security overhead in an end-to-end implementation below.

4.5.1 Comparison with Stork (Insecure Method)

We extensively compared the performance of our secure Wisecr scheme with Stork [22]. Both methods aim to offer multiple CRFID wireless firmware updates, but security objectives are not considered by Stork. Comparisons are carried out under three different test settings:

- Four different operating ranges from 20 to 50 cm.
- Updating 1 to 4 tags concurrently in-field to evaluate reduced latency to update (or improvements to scalability) (as in Stork, tags are at 20 cm from the antenna).
- Consider three different firmware sizes (as in Stork, tags are at 20 cm from the antenna).

We measured two performance metrics: i) latency; and ii) throughput as defined in Section 4.4.5 for each test setting. For both Stork and Wisecr, we used the same binary files generated from the

same compilation. The settings in the Wisecr Server Toolkit are: 10 attempts per broadcast run; lowest voltage V_{t_i} pilot token selection method; Send Mode is set to Broadcast. Meanwhile, the settings in Stork are: BWPayload is set to throttle; Update Only is set to True; Reprogramming Mode is set to Broadcast; Compression is Disabled (as compression simply reduces the size of the firmware, we did not enable this option). Results are mean values from 100 repeated protocol update runs. Comparison results from our experiments are detailed in Figure 4.13:

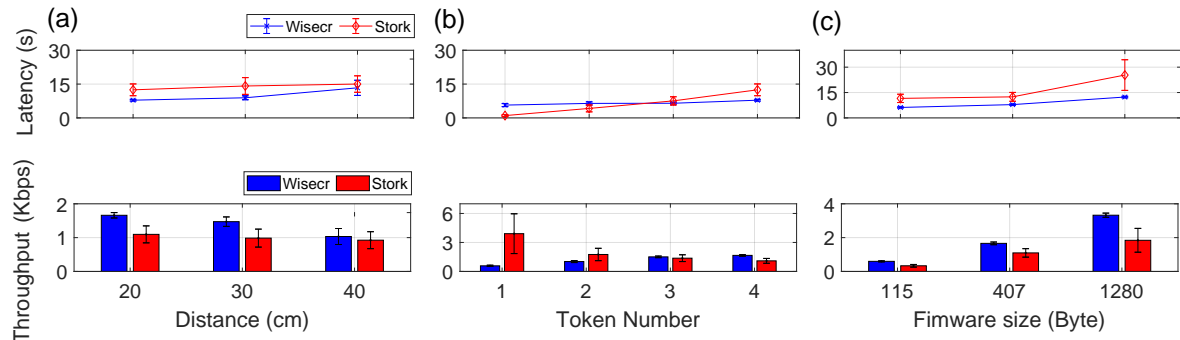


Figure 4.13: We repeated the firmware update process for Wisecr (ours) and Stork to compare between Stork (providing *no security properties*) and our Wisecr scheme under three different test settings: (a) increasing operational range; (b) increasing number of tokens; and (c) payload sizes.

- From Figure 4.13 (a): when powering channel state is reasonable, that is from 20 cm to 40 cm, Wisecr outperforms Stork, as Wisecr does not need per-block checking and relies on the pilot token selection method to improve broadcast performance. However, Stork performs better than Wisecr at 50 cm when the powering channel condition is very poor because Stork is able to continue to transmit the firmware across power failures since the stork scheme does not need to meet any security requirements, and therefore, is able to send firmware payload in plaintext from the last correctly received payload. In contrast Wisecr gracefully fails when a power interruption cannot be prevented and all states (such as the session key) are lost and a new broadcast attempt must be made by the Server Toolkit.
- From Figure 4.13 (b): Wisecr exhibits significantly better latency and throughput as the number of tokens increases.
- From Figure 4.13 (c): both protocols show improved efficiency as larger payload/firmware sizes are transmitted. Notably, Wisecr exhibits much better efficiency attributed to the significantly reduced latency experienced during the broadcast method.

The gains in performance and faster updates to many devices achieved by Wisecr can be attributed to: i) our pilot token selection method to drive the protocol where a significantly large

4.6 Acknowledgement

proportion of in-field tokens are updated in a single attempt; and ii) our validation method of complexity $O(n \cdot 1)$ where n is the number of tokens—although computationally intensive—is more lightweight than the read back mechanism of Stork, $O(n \cdot k)$ where k is the code size, relying on the narrow band communication channel employed by CRFID devices.

4.6 Acknowledgement

Michael Chesser contributed to the development of the Wisecr App (the desktop application) and interfacing with the RFID reader, the immutable on-device SecuCode bootloader implementation and implementing the broadcast method over the RFID air interface through the Wisecr App.

4.7 Chapter Summary

In this chapter, we have proposed and implemented the first secure and simultaneous wireless firmware update to many RF-powered devices with remote attestation of code installation. We explored highly limited resources and innovated to resolve security engineering challenges to implement Wisecr. The scheme prevents malicious code injection, IP theft, and incomplete code installation threats whilst being compliant with standard hardware and protocols. Wisecr performance and comparisons with state-of-the-art through an extensive experiment regime have validated the efficacy and practicality of the design, whilst the end-to-end implementation source code is released to facilitate further improvements by practitioners and the academic community.

In this chapter, the lightweight physically obfuscated key derivation using on-chip SRAM fingerprints were not employed since the implementation overheads of even the lightweight on-device reverse fuzzy extractor encoder in Chapter 3 was found to be too high for Wisecr. Even though in Section 3.3.3 in Chapter 3, we have applied a number of techniques in SecuCode, including a narrow legal temperature range, we still need to sacrifice the key reliability to reduce the time complexity of the reverse fuzzy extractor encoder operate under the resource—energy and computational capability—limitations imposed by the devices. Hence, the following chapters will investigate the problem of implementing an RFE-based key derivation method on resource-limited devices.

Meanwhile, the PAM proposed in this chapter requires on-device power measurements by sampling harvested voltage level using ADC integrated inside the MCU, which may not be available on some devices and sampling the ADC comes at the expense of consuming some of

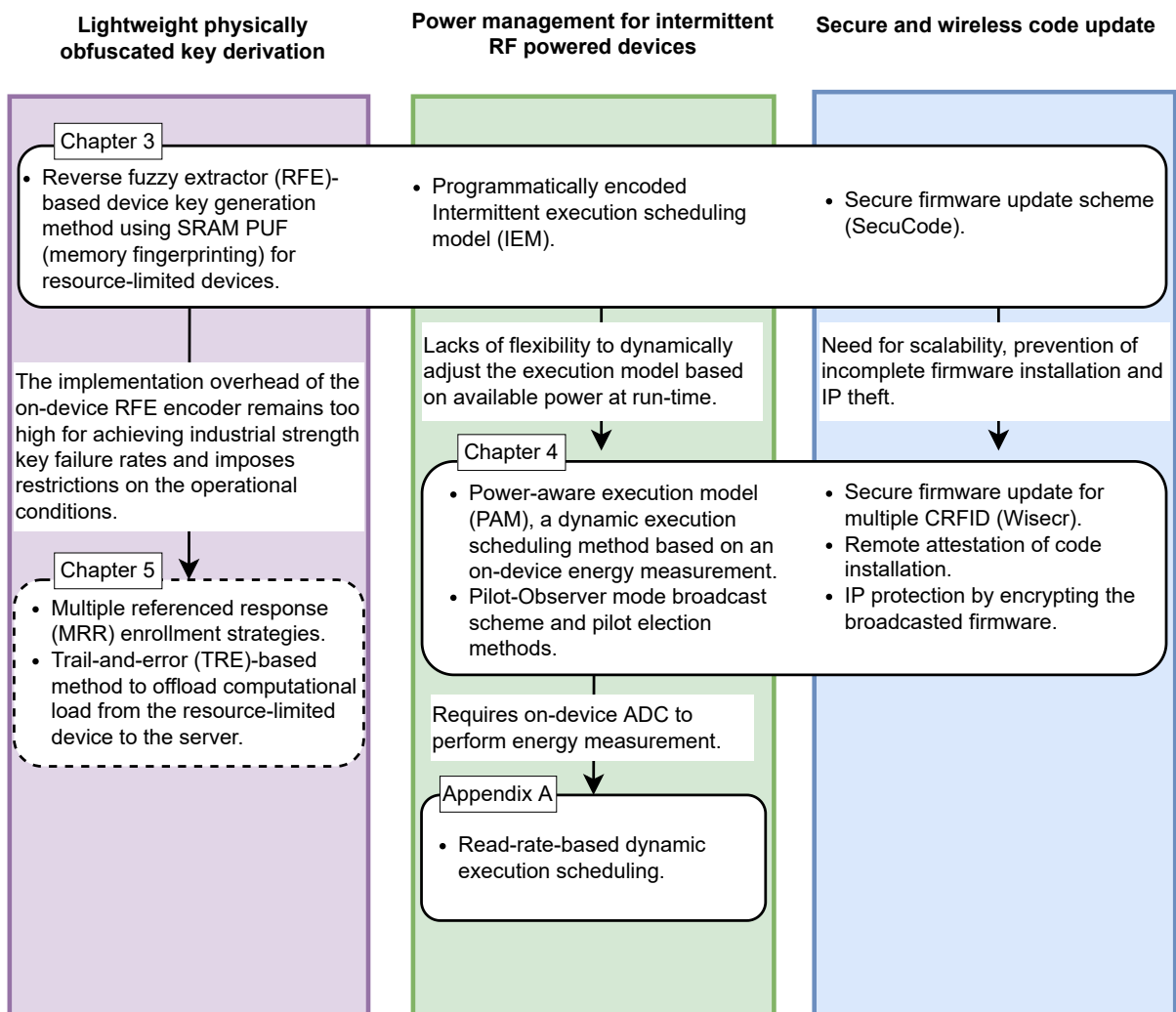


Figure 4.14: Upcoming chapter sneak peek.

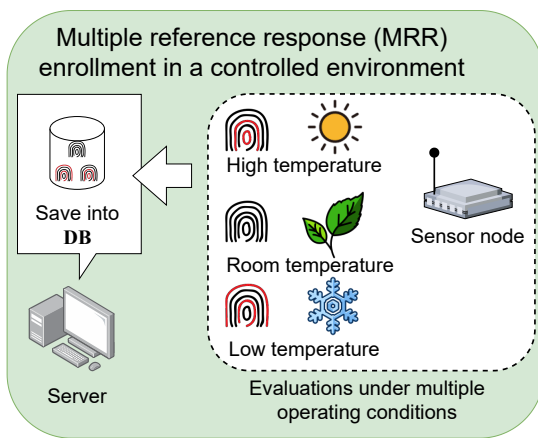
the harvested energy. Therefore, we introduce a Read-rate based dynamic execution scheduling mechanism in Appendix A based on the read-rate reported from the reader without any implementation overhead on-device. However, due to the throughput limitation of an ALOHA channel and multi-CRFID interference, read-rate based method becomes inaccurate under the multi-device environment. It requires further investigations to be adopted to the Wisecr.

In Figure 4.14 we illustrate the problems addressed in this chapter and explore the problems to be examined in the upcoming chapters. Notably, Chapter 5 will consider the problem of reducing the computational complexity of the on-device RFE generator function by investigating a novel enrolment strategy.

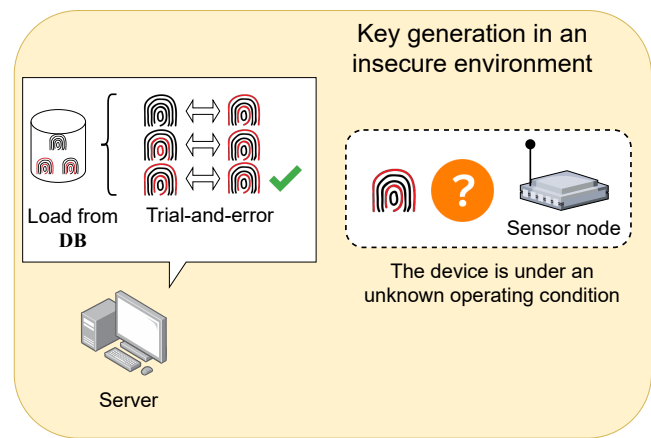
Chapter 5

Lightweight Key Generation with Multiple Reference Response Strategies

Enrollment



Deployment



Chapter 3 demonstrates the computationally expensive and challenging nature of implementing a reverse fuzzy extractor-based PUF key generation method in the context of resource-limited devices operating under intermittent powering settings. In this chapter, we propose the adoption of multiple referenced responses (MRR), subject to the same PUF challenge, produced under multiple discrete operating conditions at the resourceful server during the secure response enrolment stage. The MRR methodology reduces the burden on the on-device reverse fuzzy extractor encoder function, as used in Chapter 3, by allowing the server to reconcile a greater number of errors with a minimal compromise in the level of security afforded by the key generator. The reduction in computational cost achieved by the proposed MRR method is demonstrated from software implementations on a batteryless, resource-limited CRFID device, where the PUF data (memory fingerprints) are collected from intrinsic SRAM within the CRFID devices.

5.1 Motivation and Contribution

Resource-limited devices provide challenging environments for building security mechanisms, where traditional secure key-storage methods are hardly applicable [194]. As an alternative, reverse fuzzy extractor (RFE)-based key generators have been widely accepted to derive reliable cryptographic keys from device memory fingerprints, which are inherently noisy [66], [73]. Although an RFE-based key generator—introduced in Chapter 2—assures a high-level of security, as shown in Chapter 3, they are challenged by expensive implementation overheads resulting from the error correction process responsible for stabilising a noisy response. The prohibitive resource demands for error correction poses a major roadblock for adopting memory PUF key generators on resource-constraint embedded computing platforms such as CRFID devices with limited computational capability, memory and power.

The lightweight physically obfuscated key derivation method in SecuCode, designed in Section 3.3.3 in Chapter 3, applied a number of complex techniques to enhance the reliability of key material (i.e., the SRAM PUF responses) and reduce the computational complexity of a reverse fuzzy extractor-based physically obfuscated key derivation method, including imposing a narrow legal temperature range for device key generation. Consequently, the legal temperature range limits the application scenario of SecuCode and introduces more implementation overhead to the resource-limited device by sampling the chip temperature using an internal thermometer. Therefore, this Chapter considers the following problem:

- How can we *reduce the overheads* of the (reverse) fuzzy extractor-based physically obfuscated key derivation without imposing a constraint on the operating conditions?

In addressing the problem, this chapter takes an important step to investigate a novel enrolment methodology to substantially optimise the overhead of implementing a PUF key generator on resource-limited devices such as CRFID devices and wireless sensor nodes.

One of the key observations is that all previous PUF key generators solely enrol a single reference response that is evaluated under the so-called nominal operating condition, e.g., room temperature. This is ineffective for reducing the unreliability caused by the fact that the operating condition of a PUF in-the-field can vary significantly from the *nominal* operating condition used in the enrolment process¹⁷. In contrast, we propose multiple reference response (MRR)

¹⁷We recognise that the study in [195] conducted Ring Oscillator frequency measurements under two discrete operating conditions with the objective of maximising the number of independent response bits enrolled from an RO-PUF whilst facilitating the selection of highly reliable bits at a given selection threshold [195]; however, only the derived single reference is enrolled.

enrolment under discrete operating conditions. The crucial observation is that one of the operating conditions of an enrolled reference response will be closer to the operating condition of the PUF in the field. Alternatively, though the reproduced response is fixed and is based on the operating condition of the PUF that is out of the control of the server, the reference response can indeed be flexibly selected by the server. The overall result is a significant reduction in the expected unreliability when compared with the conventional single reference enrolment methodology.

As an immediate application, we combine MRR with a RFE to realise a MRR-based RFE (MR³FE) that suits lightweight key derivation; attributing to the greatly decreased implementation overhead. To examine the MRR method's generalisation, it is adopted to a FE, termed MR²FE. Performance efficiency of both MR³FE and MR²FE are evaluated by software implementations on a CRFID device that is batteryless and resource-limited. For instance, when a key restoration failure rate of less than 10^{-6} is required and pre-selection-based MRR enrolment using only three references at $\{-15\text{ }^{\circ}\text{C}, 25\text{ }^{\circ}\text{C}, 80\text{ }^{\circ}\text{C}\}$ is utilised, MR³FE can reduce the clock cycle overhead by 45% in comparison with a conventional RFE, while MR²FE can reduce the clock cycle overhead by 42% in comparison with a conventional FE.

The main contributions made in this chapter are summarised as below:

- **A novel multiple reference responses (MRR) enrolment strategy.** For the first time, we leverage MRR enrolled under discrete operating conditions for PUF key generation.
- **A RFE-based key derivation method with MRR strategy.** As an immediate application, a lightweight key derivation method is proposed, by putting the RFE and MRR strategy together, dubbed MR³FE. This chapter also analyse the key failure rate of MR³FE.
- **Practical use cases targeting resource constraint devices.** To demonstrate the significant reduction in implementation overhead of MR³FE, this chapter performed extensive experiments using software implementations targeting a resource constraint device—a batteryless CRFID device—with an embedded SRAM PUF. To examine the generalisation of MRR, this chapter experimentally showcase its applicability to a fuzzy extractor and also validate the substantially reduced implementation overhead.

5.1.1 Chapter Overview

Section 5.2 introduces MRR enabled RFE-based key derivation (MR³FE), which is experimentally validated in Section 5.3. Section 5.4 demonstrates the generalisation of the MRR

5.2 Multiple Reference Response-based Reverse Fuzzy Extractor (MR³FE)

by adopting it into FE (MR²FE). We discuss related work and analysis the security of the MRR when it is employed for FE and RFE as well as limitations of current investigations in Section 5.5. Section 5.6 concludes this chapter.

5.1.2 Notations and Concepts

Adding to the general notations and conventions defined in Section 2.1 in Chapter 2, Table 5.1 summarises some key concepts introduced and referred to in this chapter.

Table 5.1: Table of notations in this chapter.

S	A resourceful server.
\mathcal{T}	A resource-limited device.
\mathcal{A}	The adversary \mathcal{A} is an attacker, for the adversary \mathcal{A} 's goal ability please refer to Section 5.5.
DB	Server's database, where each element is a two-tuple describing each token \mathcal{T} : i) the unique and immutable identification number \mathbf{id}_i ; and ii) enrolled responses $\mathbf{r}_1, \dots, \mathbf{r}_j, \dots, \mathbf{r}_J$.
\mathbf{r}	A PUF response.
\mathbf{k}	A secret key for secure communications.
\mathbf{p}	Helper data, for more details please refer to Section 2.3.3.
\mathbf{n}	A randomly generated nonce.
\mathbf{u}	A hash digest.
\square'	An apostrophe denotes a quantity evaluated at different time, e.g., \mathbf{r}' denotes the reproduced response.
\square_i	A subscript $i \in [1, 2, \dots]$ denotes a specific item out of a collection, e.g., \mathbf{r}_2 is the PUF response measured under the 2 nd operating condition.
$\square_{\mathcal{A}}$	A subscript \mathcal{A} denotes a value forged by the adversary \mathcal{A} , e.g., $\mathbf{p}_{\mathcal{A}}$ is a helper data manipulated by an adversary.
OC_i	The i^{th} operating condition (OC), e.g., temperature, (for more details, please refer to Section 5.2.2).
RNG()	A random number generator RNG() outputs a random number when invoked.
Hash()	A cryptographic hash function (for more details, please refer to Section 5.3.2).
PUF	A function denoting an SRAM PUF that outputs response \mathbf{r} (SRAM start-up value) when invoked. (for more details, please refer to Section 2.3 in Chapter 2).
FE. \square ()	A fuzzy extractor FE is a noise compensation or error correction utility defined by two functions: key generation algorithm FE.Gen() and key reconstruction algorithm FE.Rep(). For more details please refer to Section 2.3.3.

5.2 Multiple Reference Response-based Reverse Fuzzy Extractor (MR³FE)

This section explains our intuition for developing the multiple reference response (MRR) approach, and then focus on the application of the approach in its most interesting context, the reverse fuzzy extractor (RFE). We explain our rationale by developing an understanding of response unreliability.

5.2.1 Our Observation

The commonly used PUF reliability model, e.g., in [124], [125], assumes a fixed error rate, specifically, each response reevaluation is assigned the same error rate. This is also referred to as the homogeneous response error rate. In practice, PUF responses are experimentally demonstrated to exhibit a bit-specific reliability—heterogeneous error rate [114], [126].

In this chapter we use the expected value of BER concept introduced in Section 2.3.2 in Chapter 2 to provide not only a convenient but also a valid method to analyse the key failure rate in relation to a (reverse) fuzzy extractor.

Commonly, \mathbf{r} is a reference response evaluated under a given operating condition and \mathbf{r}' is the reproduced response evaluated, most likely, under a differing operating condition. BER is influenced by factors such as thermal noise as well as environmental parameters such as supply voltage and temperature.

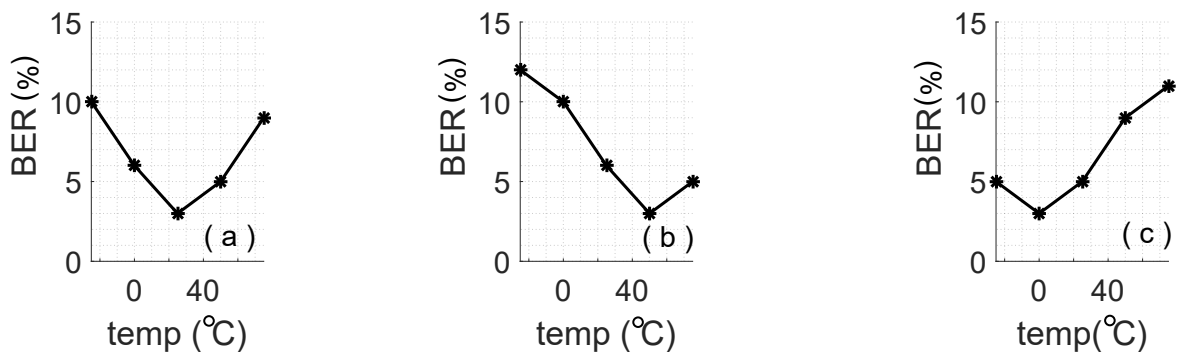


Figure 5.1: Referenced response under three operating conditions: (a) the referenced response is enrolled under nominal operating condition of 25 °C. To the best of our knowledge, actually all current PUF applications enrol only a single nominal response evaluated under a temperature around 27 °C that is the room temperature; (b) the referenced response is enrolled under 50 °C; and (c) the referenced response is enrolled under 0 °C.

An example is used to explain our observations and rationale. Figure 5.1 (a)¹⁸ illustrates a *single* reference response enrolled under 25 °C¹⁹; the nominal reference operating condition. We can see that the BER increases when the operating temperature deviates away from the reference operating condition of 25 °C. The maximum BER is around 10%, which occurs at –25 °C. The minimum BER is under the reference temperature of 25 °C. Here, we reason the minimum BER to be solely caused by thermal noise. In Figure 5.1 (b), the *reference response* is enrolled under 50 °C. We can see that the minimum BER appears at 50 °C; the nominal reference operating condition in this case. The maximum BER is approximately 12% when the re-generated response

¹⁸The BER value in this figure is not obtained from experimental evaluations, it is used only for illustrative purposes.

¹⁹Herein, for simplicity, supply voltage is assumed to be constant.

5.2 Multiple Reference Response-based Reverse Fuzzy Extractor (MR³FE)

is evaluated under $-25\text{ }^{\circ}\text{C}$, that is $75\text{ }^{\circ}\text{C}$ lower the reference operating condition. Similarly, in Figure 5.1 (c), when the reference response is enrolled under $0\text{ }^{\circ}\text{C}$, the minimum BER occurs at $0\text{ }^{\circ}\text{C}$ and the maximum BER around 12% occurs when the operating condition increases by $75\text{ }^{\circ}\text{C}$.

In summary, no matter which specific nominal reference operating condition is selected, for example, $-25\text{ }^{\circ}\text{C}$, $25\text{ }^{\circ}\text{C}$ or $50\text{ }^{\circ}\text{C}$, the minimum BER is always at the reference operating condition. BER increases as the difference between the reference operating condition and the operating condition under which response \mathbf{r}' is reproduced increases.

A deviation of the operating conditions of the PUF in the field from that under which a response is enrolled will always lead to a deterioration in the expected BER. Although we cannot change the operating condition under which the PUF operates in the field, we recognise that we can choose a suitable *reference* operating condition during response reconciliation to reduce the maximum number of erroneous bits we expect in a re-generated response. Next, we utilise this important observation to reason about the MRR-based RFE key derivation (MR³FE) mechanism.

5.2.2 RFE-based Key Derivation with MRR Strategy

Figure 5.2 depicts the proposed MR³FE key derivation protocol. In comparison with conventional RFE-based key derivation (as depicted in Figure 5.7), there are two distinct differences:

- In the enrolment phase, instead of enrolling a *single* reference response, the server S enrolled multiple reference responses; each reference response is evaluated at a different operating condition. This is highlighted in ①.
- In the key derivation phase, the server S recovers the re-generated response \mathbf{r}' of the token \mathcal{T} based on the enrolled multiple reference responses. This is highlighted in ②.

Next we elaborate on the MR³FE key derivation by taking two reference responses as an example.

An Example with Two Referenced Responses

In Figure 5.1, during the enrolment phase, we assume that the server S enrolls two reference responses, \mathbf{r}_1 and \mathbf{r}_2 , evaluated under $50\text{ }^{\circ}\text{C}$ and $0\text{ }^{\circ}\text{C}$, respectively. It is worth reminding that \mathbf{r}_1 and \mathbf{r}_2 are subject to the same challenge applied to the same PUF.

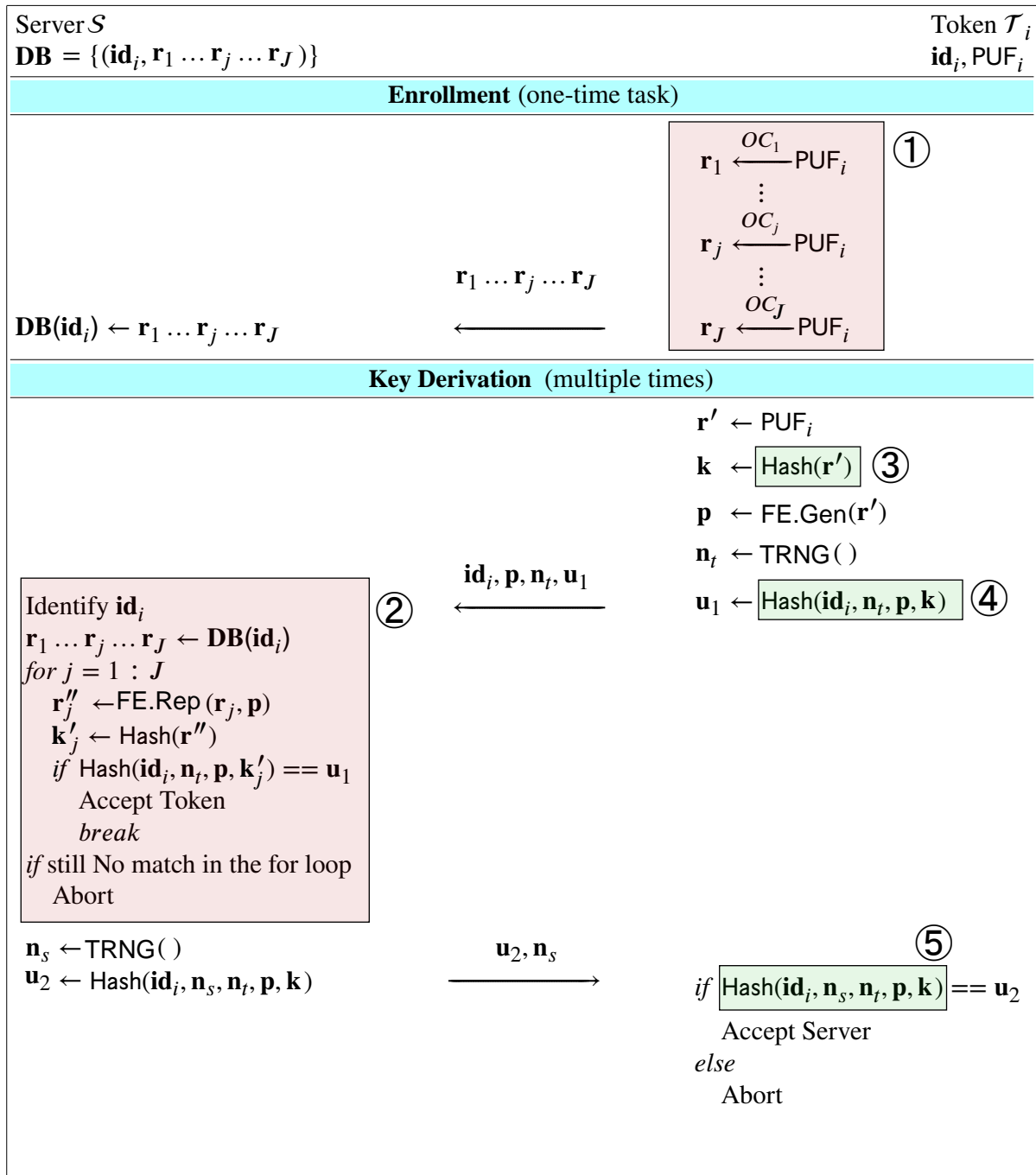


Figure 5.2: RFE-based key derivation with MRR enrolment strategy. OC stands for operating condition.

In the key derivation phase, the token \mathcal{T} measures the response \mathbf{r}' and then computes helper data $\mathbf{p} \leftarrow \text{FE.Gen}(\mathbf{r}')$. In addition, verification data $\mathbf{u}_1 \leftarrow \text{Hash}(\mathbf{id}_i, \mathbf{n}_t, \mathbf{p}, \mathbf{k})$ is computed, where \mathbf{u}_1 is a keyed hash value with \mathbf{k} as the key. The \mathbf{id}_i is the identification number of current token \mathcal{T} , \mathbf{n}_t is a nonce generated by the token \mathcal{T} . $\mathbf{id}_i, \mathbf{n}_t, \mathbf{p}$ along with \mathbf{u}_1 are publicly sent to the server S . The server S now attempts to reconstruct the response \mathbf{r}' based on its enrolled responses: \mathbf{r}_1 and \mathbf{r}_2 . This can be handled in an iterative way. The server S first uses \mathbf{r}_1 to generate $\mathbf{r}'' \leftarrow$

5.2 Multiple Reference Response-based Reverse Fuzzy Extractor (MR³FE)

FE.Rep(\mathbf{r}_1, \mathbf{p}). Once response \mathbf{r}'' is obtained, the server S verifies whether Hash($\mathbf{id}_i, \mathbf{n}_t, \mathbf{p}, \mathbf{k}'$) equals the verification value \mathbf{u}_1 sent by the token \mathcal{T} using the recovered secret key $\mathbf{k}' \leftarrow \text{Hash}(\mathbf{r}'')$. If the verification is successful, then $\mathbf{k} = \mathbf{k}'$, for this reason, response \mathbf{r}' is deemed to be successfully restored. The key derivation can now proceed based on the shared secret session key \mathbf{k} . If Hash($\mathbf{id}_i, \mathbf{n}_t, \mathbf{p}, \mathbf{k}'$) is not equal to \mathbf{u}_1 and the verification fails, the server S continues to use \mathbf{r}_2 for reconstructing \mathbf{r}' to determine whether \mathbf{r}' can be successfully recovered.

Notably, it is only after *both* \mathbf{r}_1 and \mathbf{r}_2 are exhausted in the recovery of the response \mathbf{r}' that MR³FE-based key derivation fails. This occurs on the condition that the verification of \mathbf{u}_1 has *failed* and implies that the recovery of \mathbf{r}' has failed.

Advantages

The example above illustrates the advantages of a MR³FE. Let us first assume that the computed helper data \mathbf{p} by the token \mathcal{T} is only supposed to guarantee a successful secret key \mathbf{k} reconstruction by the server S with $\mathbf{k} \leftarrow \text{Hash}(\mathbf{r}')$ when the BER is no more than 5%—in other words, less than 5% of response bits being erroneous under a reevaluation. Assume a *single* reference response \mathbf{r} under 25 °C is utilised for key reconstruction by the server S , as in the conventional RFE case, and the response \mathbf{r}' is reproduced under an operating condition of −25 °C. We can observe from Figure 5.1 (a) that the \mathbf{r}' is highly *unlikely* to be correctly recovered by the server S because the BER based on the reference response \mathbf{r} evaluated under 25 °C is much higher than 5% at −25 °C.

Let's now assume employing two reference responses, \mathbf{r}_1 and \mathbf{r}_2 , as in the MR³FE setting, and still assume that \mathbf{r}' is from the PUF operating under −25 °C. We can see that reference response \mathbf{r}_2 has a high chance to *successfully* recover response \mathbf{r}' relying on the fact that the BER based on \mathbf{r}_2 evaluated under 0 °C as a reference response is now less than 5%—see Figure 5.1 (c). Similarly, if \mathbf{r}' is measured from a PUF operating under 75 °C, then using \mathbf{r}_1 evaluated under 50 °C as a reference response will lead to a BER of less than 5%, see Figure 5.1 (c), and consequently, a successful response recovery.

Overall, we can see that in practice, the server S is unable to change the operating condition under which the re-generated response \mathbf{r}' is evaluated. Consequently, an approach using a single reference response is forced to account for a significantly higher BER than at the nominal operating condition. In contrast, we can observe that the proposed MRR approach facilitates the server S to employ an appropriate reference response to minimise the expected difference between a reference response \mathbf{r} and the re-generated response \mathbf{r}' to meet a given error correcting capability threshold d .

Next, we analyse the key reconstruction failure rate of MR³FE key derivation; this is also the false rejection rate of the key derivation mechanism.

5.2.3 Key Failure Rate

To validate the efficiency of the proposed MR²FE and MR³FE, we focus on the *average* failure rate of the PUF key generator. In [126], it is demonstrated that the expected value of key failure rate based on a bit-specific reliability model is equivalent to the key failure rate predicted under the commonly used reliability model with a fixed response error rate—as in equation (2.2) in Section 2.3.2 in Chapter 2. In other words, the homogeneous reliability model correctly captures the average key failure rate of a PUF key generator [73], [126]. Therefore, we will use BER defined in equation (2.2) to derive key failure rate.

This chapter studies the family of BCH(n, k, t) linear codes with a syndrome-based decoding strategy to realise a reverse fuzzy extractor considering its popularity [73], [124] and its security property [73], [74]—we discuss security of (reverse) fuzzy extractors in Section 5.5. Here, n is the code word length, k is the code size, t is the number of errors that can be corrected within this n -bit block. Assuming response bit are i.i.d., we can express the average key failure rate of recovering an n -bit response \mathbf{r}' based on a selected reference response \mathbf{r}_j , termed as P_{1j} , where the $j \in \{1, \dots, J\}$ with J as the number of multiple references employed by the server S , as:

$$P_{1j} = 1 - \text{binocdf}(t; n, \text{BER}_j) \quad (5.1)$$

where BER_j is the BER using \mathbf{r}_j as the reference response. Here, $\text{binocdf}()$ is a cumulative density function of a binomial distribution with t successes in n Bernoulli trials, with each trial having success probability of p , expressed as:

$$\text{binocdf}(t; n, p) = \sum_{i=0}^t \binom{n}{i} p^i (1-p)^{(n-i)} \quad (5.2)$$

A BCH(n, k, t) encoding produces $(n - k)$ -bit helper data assumed to be publicly known while k bits form the secret key material. For a single BCH(n, k, t) block, the complexity of finding the k -bit response from \mathbf{r}' is 2^k . It is not common to use a single large BCH(n, k, t) block; typically a large block is split into small processing blocks to reduce implementation complexity (overhead) [196]. For k bits of key material, response \mathbf{r}' can be divided into multiple non-overlapping blocks of a BCH(n_1, k_1, t_1) code where $n_1 < n$ and $k_1 < k$ for a parallel implementation. Now the complexity of finding the k bit secret is $2^{k_1 \cdot L}$ where L is the number

5.3 Experimental Validations

of parallel BCH(n_1, k_1, t_1) code blocks used to realise k bits of secret key material. Given a BCH(n_1, k_1, t_1) code employed to gain a security level of k bits with $L = \lceil k/k_1 \rceil$ blocks, the key recovery failure rate under the assumption of i.i.d code blocks is:

$$P_{2j} = 1 - (1 - P_{1j})^L. \quad (5.3)$$

When all J reference responses $\{\mathbf{r}_1, \dots, \mathbf{r}_j, \dots, \mathbf{r}_J\}$ are employed, \mathbf{r}' reconstruction fails only when *all* reference responses cannot restore the response \mathbf{r}' . Therefore, the key failure rate $P_{\mathbf{r}}^{\text{fail}}$ for J reference responses can be expressed as a joint probability distribution:

$$P_{\mathbf{r}}^{\text{fail}} = P(\mathbf{r}_1, \cap \dots \cap, \mathbf{r}_j, \cap \dots \cap, \mathbf{r}_J) \quad (5.4)$$

However, due to the complexity of PUF response properties, e.g., correlations, formally deriving a joint distribution without assuming that $\{\mathbf{r}_1, \dots, \mathbf{r}_j, \dots, \mathbf{r}_J\}$ are independently drawn under distinct operating conditions is a non-trivial task²⁰. Here, we take a very conservative evaluation of the key failure rate $P_{\mathbf{r}}^{\text{fail}}$ without a prior notion of independence implied on the reference responses $\{\mathbf{r}_1, \dots, \mathbf{r}_j, \dots, \mathbf{r}_J\}$ ²¹. We recognise that we can express an upper bound for the key failure rate $P_{\mathbf{r}}^{\text{fail}}$ as:

$$P_{\mathbf{r}}^{\text{fail}} = P(\mathbf{r}_1, \cap \dots \cap, \mathbf{r}_j, \cap \dots \cap, \mathbf{r}_J) \leq \min\{P_{2j}\}, j \in \{1, \dots, J\} \quad (5.5)$$

Now we adopt the very conservative estimate:

$$P_{\mathbf{r}}^{\text{fail}} = \min\{P_{2j}\}, j \in \{1, \dots, J\} \quad (5.6)$$

in our analysis.

5.3 Experimental Validations

We employ the ultra low power MCU used in CRFID transponders (WISP 5.1 LRG) to evaluate the overhead of the proposed MR³FE key derivation mechanism as illustrated in Figure 5.3. As discussed in Chapter 1 and Chapter 3, a CRFID device is battery-less device representative of a low-end resource-limited computing platform. Since a CRFID device has SRAM memory, it

²⁰Under an assumption of independence, the key failure rate $P_{\mathbf{r}}^{\text{fail}} = \prod_{j=1}^J P_{2j}$.

²¹Our conservative evaluation of the key failure rate $P_{\mathbf{r}}^{\text{fail}}$ implies that the overhead reduction benefits reported by our MRR approach is also conservatively assessed.

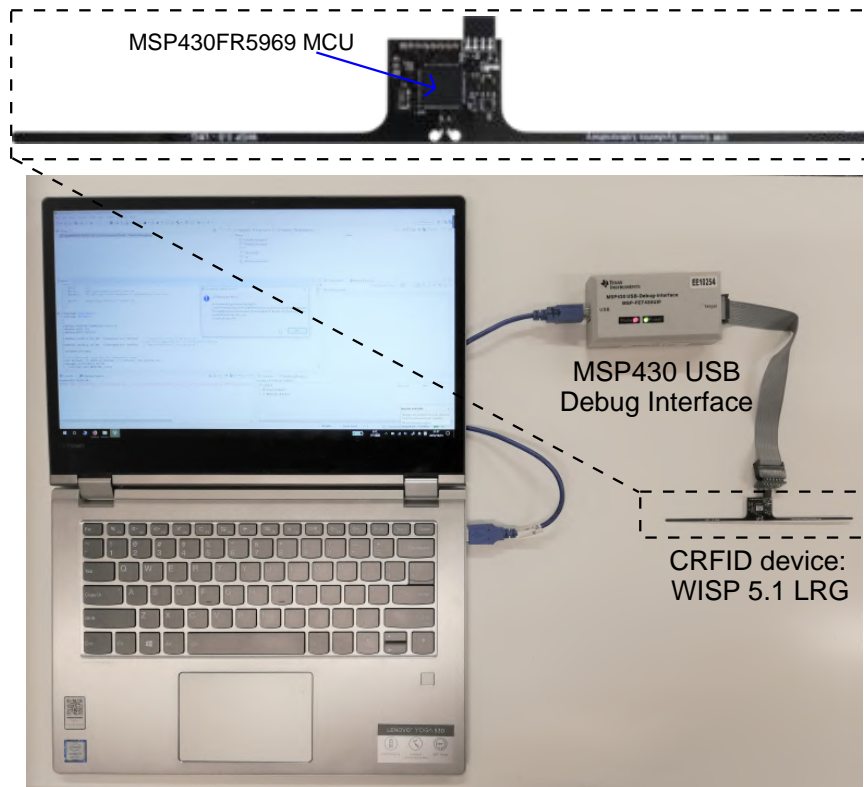


Figure 5.3: Experiment setup. The CRFID transponder is enlarged.

has the potential to use an intrinsic SRAM PUF as a trust anchor without requiring additional hardware [38].

5.3.1 SRAM PUF Dataset

The SRAM PUF dataset used is from 23 MSP430FR5969 MCUs, three of them are collected from CRFID transponders (as shown in Figure 5.3) and the rest 20 are from the MSP20 dataset contributed in Chapter 3. From each MCU, the power-up states of 16,384 KiB SRAM cells are read as SRAM PUF responses. The dataset consists of 100 repeated response measurements under the temperature conditions: $-15\text{ }^{\circ}\text{C}$, $0\text{ }^{\circ}\text{C}$, $25\text{ }^{\circ}\text{C}$, $40\text{ }^{\circ}\text{C}$ and $80\text{ }^{\circ}\text{C}$. It has been experimentally shown that the SRAM PUF reliability is much less sensitive to voltage variations compared with temperature fluctuations attributing to a SRAM cell's symmetric structure [67], [197], [198]. Hence, we focus on reliability under varying temperature conditions in our study.

5.3.2 Overhead Evaluations

5.3 Experimental Validations

Test Setup

The test environment used is Texas Instruments' (TI) Code Composer Studio (CCS) 7.2.0, the C code used is downloaded to a MSP430FR5969 LaunchPad Evaluation Kit via USB. TI CCS has a built-in GCC toolchain for our hardware kit. This includes the `msp430-gcc-6.4.0.32` win32 compiler. Considering that our main goal is to demonstrate the enhanced efficacy of MR³FE compared to the conventional RFE in a *relative* manner, dedicated optimisation of the C code was deemed out of scope. optimisation [199] of the fuzzy extractor software implementation can be carried out in the future to further minimise the *absolute* implementation overhead of the MR³FE.

The software instructions are executed sequentially as advanced out-of-order execution is unavailable for typical resource constraint MCUs. The overhead measured in terms of clock cycles to complete the algorithm is our primary concern. We measured clock cycles using the Profile Clock tool supported in the CCS environment. In addition, we also measured memory usage. Besides the 2 KiB SRAM memory embedded in the MSP430FR5969 microcontroller, it is configured with a 63 KiB FRAM. Here **FRAM usage** (overhead) is reflective of code size, while the **SRAM usage** represents size of the internal state used by the algorithm. The code size is assessed by the `.text` block in FRAM using the Memory Allocation tool in CCS where the internal state is manually counted for any local variables declared inside the algorithm routine.

Hash function and BCH code encoding are two pivotal components for realising the MR³FE and dominates the overhead of the MR³FE implementation. We comprehensively evaluate these building blocks by testing the following.

- **Hash Functions.** Six different hash functions are tested. The results are listed in Table E.1 in the Appendix. E We evaluate clock cycles and memory overhead. The input message size we selected is 240 bytes for these tests. Among all six software-based hash implementations, the BLAKE2s-128 with a 128-bit hash presents the best performance. Therefore, BLAKE2s-128 is selected for our evaluations.
- **BCH Code Encoding.** BCH(n_1, k_1, t_1) code encoding overhead under different n_1, k_1, t_1 settings are tested. Results are detailed in Table. E.2 in the Appendix. E

5.3.3 Comparisons

We first evaluate BER under three different response enrolment strategies: i) single readout; ii) majority voting ; and iii) pre-selection.

- In the single readout response enrolment, all the enrolled responses under a distinct temperature is evaluated only *once*.
- In the majority voting response enrolment, all the responses under a distinct temperature are evaluated 9 times and then the majority vote is applied for enrolment.
- In the pre-selection response enrolment, first, each response under 25 °C is repeatedly measured 10 times, only the response bits exhibiting 100% reliable re-generations (all ‘1’s/‘0’s’) are selected—12% of bits are discarded during this process. Then the reference responses under other temperatures, –15 °C, 0 °C, 40 °C, 80 °C are obtained by applying majority voting to the *pre-selected responses under 25 °C* using 9 repeated measurements.

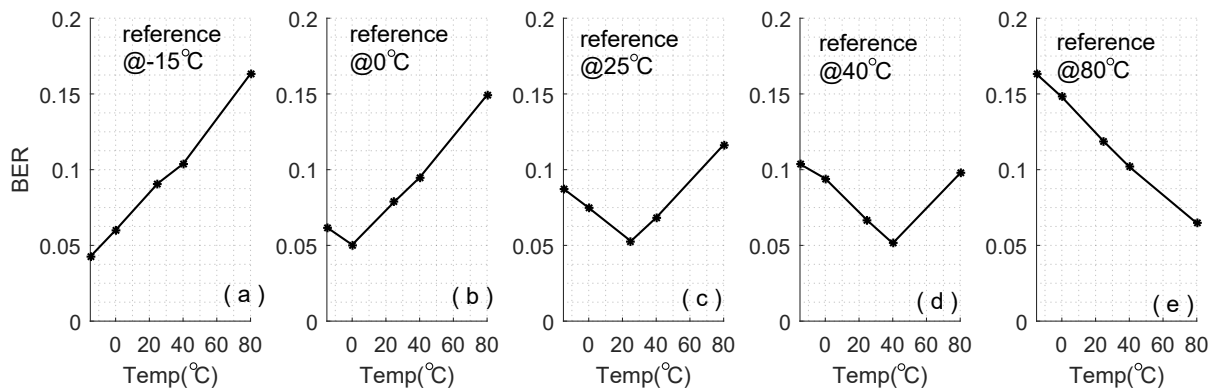


Figure 5.4: BER when single readout response enrolment strategy is utilised: (a) reference response is enrolled at –15 °C; (b) reference response is enrolled at 0 °C; (c) reference response is enrolled at 25 °C; (d) reference response is enrolled at 40 °C; and (e) reference response is enrolled at 80 °C.

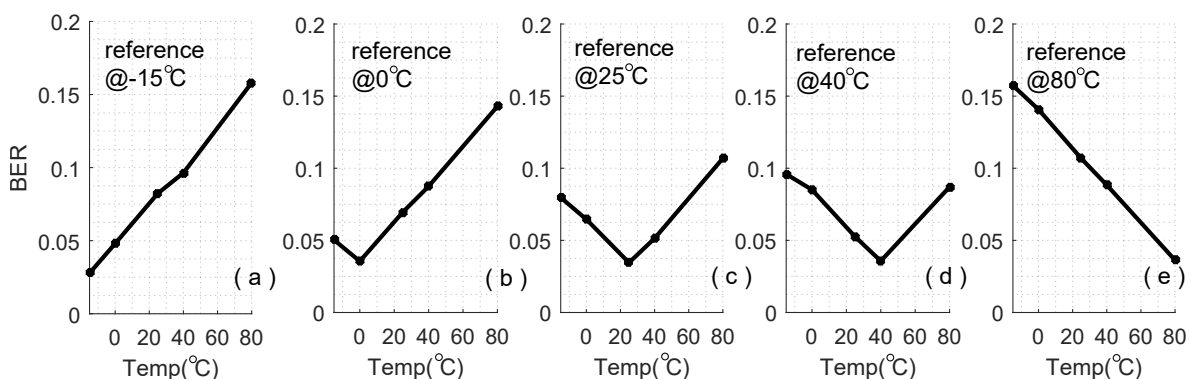


Figure 5.5: BER when majority voting response enrolment strategy is utilised: (a) reference response is enrolled at –15 °C; (b) reference response is enrolled at 0 °C; (c) reference response is enrolled at 25 °C; (d) reference response is enrolled at 40 °C; (e) reference response is enrolled at 80 °C.

BER evaluations based on the three different response enrolment approaches—single readout, majority voting and pre-selection—are illustrated in Figure 5.4, Figure 5.5 and Figure 5.6, respectively.

5.3 Experimental Validations

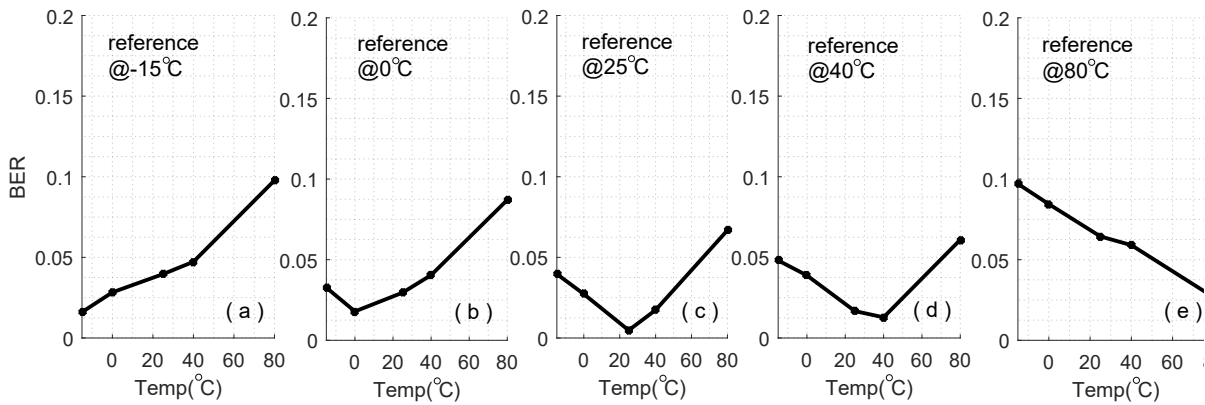


Figure 5.6: BER when pre-selection response enrolment strategy is utilised: (a) reference response is enrolled at -15°C ; (b) reference response is enrolled at 0°C . (c) reference response is enrolled at 25°C ; (d) reference response is enrolled at 40°C ; (e) reference response is enrolled at 80°C .

We observe the following:

- Regardless of response enrolment strategy, it is empirically verified that the BER increases as a function of the temperature difference between the response re-generation temperature and the reference temperature.
- As expected, both majority voting and pre-selection approaches reduce BER; the pre-selection strategy is more efficient.

Key Failure Rate. Based on BER values obtained from the three different response enrolment strategies, we are able to evaluate the key failure rate. We use parallel $\text{BCH}(n_1, k_1, t_1)$ blocks as discussed in Section 5.2.3. We consider an evaluation under the preference of deriving a 128-bit secret. Therefore, we determine the number of $\text{BCH}(n_1, k_1, t_1)$ blocks required by using $\left\lceil \frac{128}{k_1} \right\rceil$. The key failure rates we have determined is detailed in Table. 5.3. We observe the following:

- Before applying MRR, majority voting and pre-selection reduces the BER and thus decreases the key failure rate.
- Regardless of response enrolment approaches, our MRR approach further suppresses the key failure rate. In other words, the MRR approach complements response reliability enhancement approaches such as majority voting and pre-selection performed in the *enrolment phase*.

Overhead. We are now able to compare the overhead of the MR^3FE (RFE with MRR) with the conventional RFE (only using a SRR) when they are implemented on a CRFID token

Table 5.2: Key failure rate achieved for single readout, majority voting and pre-selection response enrolment strategies to realise a 128-bit key.

(n_1, k_1, t_1)	block num.	Single readout			Majority voting			pre-selection		
		SRR	2MRR	3MRR	SRR	2MRR	3MRR	SRR	2MRR	3MRR
(63,18,10)	8	0.6074	0.2821	2.67×10^{-2}	0.4355	0.1446	2.7×10^{-3}	2.26×10^{-2}	1.10×10^{-2}	8.21×10^{-6}
(63,16,11)	8	0.3789	0.1342	8.2×10^{-3}	0.2366	5.9×10^{-2}	6.22×10^{-4}	6.8×10^{-3}	3.0×10^{-3}	9.85×10^{-7}
(127,29,21)	5	0.1712	2.86×10^{-2}	2.49×10^{-4}	7.55×10^{-2}	7.1×10^{-3}	2.82×10^{-6}	1.79×10^{-4}	4.36×10^{-5}	2.97×10^{-11}
(127,22,23)	6	6.62×10^{-2}	7.4×10^{-3}	3.04×10^{-5}	2.39×10^{-2}	1.4×10^{-3}	1.91×10^{-7}	2.08×10^{-5}	4.19×10^{-6}	5.47×10^{-13}
(127,15,27)	9	5.7×10^{-3}	2.95×10^{-4}	2.66×10^{-7}	1.4×10^{-3}	3.51×10^{-5}	5.09×10^{-10}	1.66×10^{-7}	2.67×10^{-8}	$< 10^{-21}$
(255,47,42)	3	2.48×10^{-2}	9.0×10^{-4}	1.25×10^{-7}	5.4×10^{-3}	6.8×10^{-5}	2.47×10^{-11}	6.69×10^{-8}	4.58×10^{-9}	$< 10^{-21}$
(255,29,47)	5	2.8×10^{-3}	3.96×10^{-5}	8.27×10^{-10}	3.83×10^{-4}	1.62×10^{-6}	3.66×10^{-14}	3.92×10^{-10}	1.65×10^{-11}	$< 10^{-21}$
(255,21,55)	7	1.52×10^{-5}	4.72×10^{-8}	4.59×10^{-14}	9.90×10^{-7}	7.17×10^{-10}	$< 10^{-21}$	1.79×10^{-14}	$< 10^{-21}$	$< 10^{-21}$
(255,13,59)	10	7.97×10^{-7}	1.45×10^{-9}	$< 10^{-21}$	3.56×10^{-8}	1.59×10^{-11}	$< 10^{-21}$	$< 10^{-21}$	$< 10^{-21}$	$< 10^{-21}$

Table 5.3: Overhead of RFE and FE when SRR, 2MRR and 3MRR are used.

(n_1, k_1, t_1)	block num.	Reverse Fuzzy Extractor			Fuzzy Extractor				
		CPU cycles	Memory Usage		CPU cycles			Memory Usage	
			FRAM	SRAM	SRR	2MRR	3MRR	FRAM	SRAM
(63,18,10)	8	745,721	5,819 bytes	352 bytes	3,360,134	6,720,268	10,080,402	6,843 bytes	1,464 bytes
(63,16,11)	8	722,193	5,699 bytes	355 bytes	3,689,662	7,379,324	11,068,986	6,983 bytes	1,406 bytes
(127,29,21)	5	1,221,359	6,019 bytes	470 bytes	8,081,576	16,163,152	24,244,728	11,563 bytes	1,466 bytes
(127,22,23)	6	1,319,223	6,033 bytes	477 bytes	10,663,102	21,326,204	31,989,306	12,095 bytes	1,464 bytes
(127,15,27)	9	1,316,184	6,015 bytes	484 bytes	19,129,444	38,258,888	57,388,332	13,159 bytes	1,466 bytes
(255,47,42)	3	2,063,241	6,407 bytes	708 bytes	18,515,476	37,030,952	55,546,428	28,925 bytes	1,466 bytes
(255,29,47)	5	2,200,269	6,467 bytes	728 bytes	35,091,151	70,182,302	105,273,453	31,535 bytes	1,466 bytes
(255,21,55)	7	2,377,650	6,481 bytes	734 bytes	58,631,390	117,262,780	175,894,170	31,535 bytes	1,466 bytes
(255,13,59)	10	2,329,519	6,379 bytes	742 bytes	85,493,076	170,986,152	256,479,228	39,527 bytes	1,536 bytes

The FRAM and SRAM memory can be reused when multiple BCH blocks and hash are sequentially computed.

\mathcal{T} . Considering performance advantages, BLAKE2s-128 is chosen for the hash function (see Table. E.1 in the Appendix. E). It is worth reminding here that RFE-based key derivation requires a hash operation for three times as highlighted in ③, ④ and ⑤ (see Figure 5.2). In Table. 5.3, the overhead of RFE-based key derivation is detailed when SRR, 2MRR, 3MRR are deployed. We observe the following from our experiments:

- **Single Readout Response (SRR) enrolment.** To achieve $P_r^{\text{fail}} < 10^{-6}$, ten BCH(255,13,59) blocks are required when the conventional single reference response under 25 °C is used, whereas nine smaller BCH(127,15,27) blocks are adequate when 3MRR under -15 °C, 25 °C, 80 °C is deployed. In this context, the MR³FE with 3MRR reduces clock cycle overhead by 43.50% in comparison with the conventional RFE.
- **Majority Voting Response enrolment.** To achieve $P_r^{\text{fail}} < 10^{-6}$, seven BCH(255,21,55) blocks are needed when the conventional single reference response under 25 °C is used.

5.4 Multiple Reference Response-based Fuzzy Extractor (MR²FE)

In contrast, six smaller BCH(127,22,23) blocks are adequate when 3MRR under $-15\text{ }^{\circ}\text{C}$, $25\text{ }^{\circ}\text{C}$, $80\text{ }^{\circ}\text{C}$ is used. In this context, the MR³FE with 3MRR reduces clock cycle overhead by 44.52% in comparison with the conventional RFE.

- **Pre-selection Response enrolment.** To achieve $P_r^{\text{fail}} < 10^{-6}$, nine BCH(127,15,27) blocks must be applied when the conventional single reference response under $25\text{ }^{\circ}\text{C}$ is used. Conversely, eight smaller BCH(63,16,11) blocks are adequate when 3MRR under $-15\text{ }^{\circ}\text{C}$, $25\text{ }^{\circ}\text{C}$, $80\text{ }^{\circ}\text{C}$ are used. In this context, the MR³FE with 3MRR reduces clock cycle overhead by 45.13% in comparison with the conventional RFE.

Although memory usage reduction is not significant, our MRR approach leads to usage of a smaller BCH code size (less FRAM usage) and less internal state (less SRAM usage) at run time. In terms of the most concerned performance measure—MCU clock cycles—we can observe that MRR always greatly outperforms SRR. The MRR approach significantly reduces the clock cycle overhead (nearly over 43% reduction) to achieve the same key failure rate as the conventional SRR method. Alternatively, given the same overhead, the key failure rate is reduced by several orders of magnitude by employing our MRR method.

5.4 Multiple Reference Response-based Fuzzy Extractor (MR²FE)

This section examines the generalisation of the developed MRR methodology. We investigate the practicality of MRR when it is adopted into a FE scenario. In this context, the target device for implementation is not necessarily a highly resource constraint device like the CRFID transponder. *We assume that the employment of a FE is mainly to derive a secure cryptographic key, while minimising the FE implementation overhead is always a desirable goal.* In the case of MRR enabled FE—termed MR²FE, recall that it is the PUF device that iteratively carries out decoding and checking to identify whether the key is correctly recovered.

5.4.1 MR²FE

The MR²FE key generator operates as follows:

1. During the key enrolment phase, the server \mathcal{S} registers $\{\mathbf{r}_1, \dots, \mathbf{r}_j, \dots, \mathbf{r}_J\}$, which are responses subject to the same challenge but generated under differing operating

conditions: $\{OC_1, \dots, OC_j, \dots, OC_J\}$. Response \mathbf{r}_j is hashed to gain a cryptographic key $\mathbf{k}_j \leftarrow \text{Hash}(\mathbf{r}_j)$. The server S computes helper data $\mathbf{p}_j \leftarrow \text{FE.Gen}(\mathbf{r}_j)$, where $j \in \{1, \dots, J\}$.

2. During the key reconstruction phase, the PUF device re-generates response \mathbf{r}' . Then the PUF device loads helper data \mathbf{p}_j , assumed to be public information and sent from the server S at run time or loaded from an insecure off-chip/on-chip NVM. Simultaneously, the server S sends a verification value $\mathbf{u}_j \leftarrow \text{Hash}(\mathbf{id}, \mathbf{n}_s, \mathbf{p}_j, \mathbf{k}_j)$ along with \mathbf{id}_i and \mathbf{n}_s —the nonce generated by the RNG on the server S —to the PUF device.
3. The PUF device performs $\mathbf{r}'_j \leftarrow \text{FE.Rep}(\mathbf{p}_j, \mathbf{r}')$, where $\mathbf{r}'_j = \mathbf{r}_j$ only on the condition that $\mathbf{u}_j = \mathbf{u}'_j \leftarrow \text{Hash}(\mathbf{id}_i, \mathbf{n}_s, \mathbf{p}_j, \mathbf{k}'_j)$, with $\mathbf{k}'_j = \text{Hash}(\mathbf{r}'_j)$. Once this occurs, the PUF device is deemed to have successfully restored the response \mathbf{r}_j . Thus the secret key \mathbf{k}_j is recovered and *the following steps are skipped*.
4. Otherwise, if $\mathbf{u}'_j \neq \mathbf{u}_j$, then the secret key \mathbf{k}_j reconstruction fails. Step 2 and 3 must be repeated for reconstructing another key \mathbf{k}_i using helper data \mathbf{p}_i , where $i \neq j$.
5. If none of the $\mathbf{k}_j, \forall j \in \{1, \dots, J\}$, reconstructions are successful, the key reconstruction fails.

As we can observe here that the server S actually enrolls J helper data; each corresponding to one reference response generated under a varying operating condition. The PUF device iteratively conducts key reconstruction attempts to recover one enrolled cryptographic key, \mathbf{k}_j . If one of them is successfully reconstructed, the key recovery succeeds. Otherwise, if none of them succeeds, key reconstruction failure occurs.

To study the overhead of the MR²FE key generator, we comprehensively evaluate the BCH code decoding overhead—this corresponds to the FE.Rep() implementation overhead.

5.4.2 BCH Code Decoding

BCH(n_1, k_1, t_1) code is chosen again for consistency, its decoding overhead under different n_1, k_1, t_1 settings are tested, results are detailed in Table. E.2 in the Appendix. E The experimental setup is same as that described in Section 5.3.2.

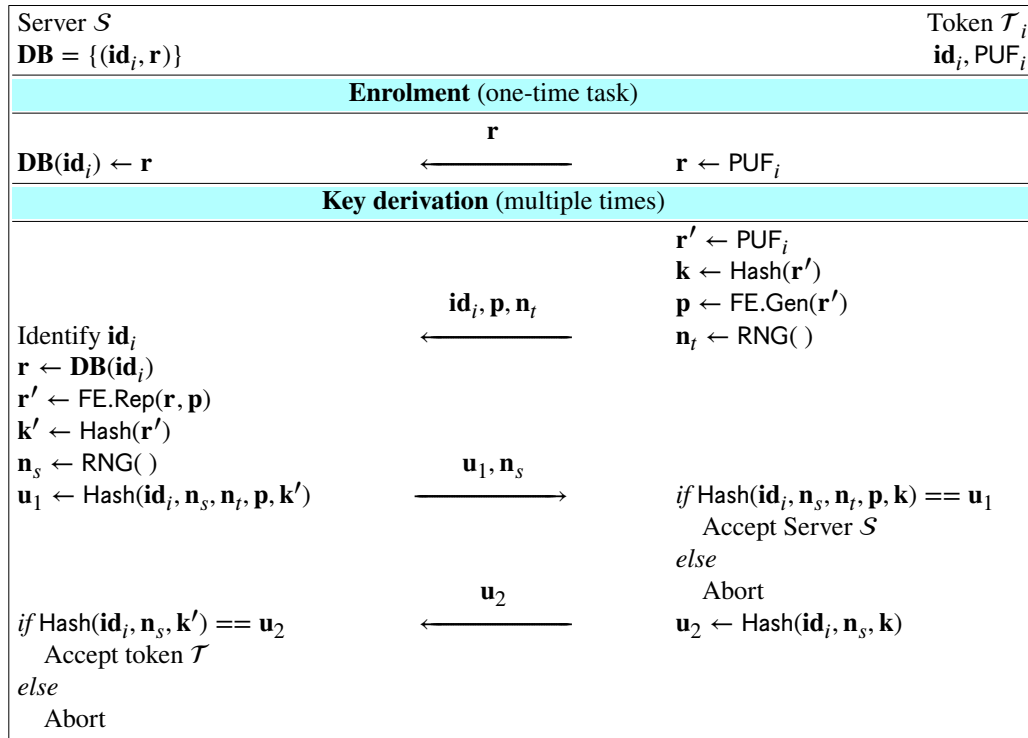
5.4.3 Comparison

Following the operational steps of the MR²FE key generator (see Section 5.4.1), we are able to quantitatively compare the MR²FE implementation overhead imposed on the token \mathcal{T} given SRR, 2MRR and 3MRR. Similar to Section 5.3.3, we still select BLAKE2s-128 hash function. MR²FE needs to execute the FE.Rep() function *at most* J times and the Hash() function $2 \times J$ times. This is significantly different from the MR³FE where increasing J , the number of reference responses, brings *no extra* computational overhead to the token \mathcal{T} .

In practice, the recovery of secret key \mathbf{k}_j that is from a nominal operating condition, e.g., room temperature, is *recommended* to be tried first. Because, the response \mathbf{r}' will be more likely reproduced under an operating condition that is close to the nominal operating condition. On the condition that such a trial succeeds, the remaining trials are no long needed—step 4) and 5) are skipped—and the imposed overhead is further reduced. Nevertheless, we analyse the worst-case scenario, that is assuming that all J trials have to be performed by the PUF device before a successful key recovery. Overhead comparisons for a FE with SRR, 2MRR, 3MRR are detailed in Table 5.3. We can make the following observations.

- **Single Readout Response enrolment.** To achieve a $P_r^{\text{fail}} < 10^{-6}$, ten BCH(255,13,59) blocks are required when the conventional single reference response under 25 °C is used. In contrast, nine smaller BCH(127,15,27) blocks are adequate when 3MRR under –15 °C, 25 °C, 80 °C is used. In this context, the MR²FE with 3MRR reduces clock overhead by 32.87% in comparison with the conventional FE.
- **Majority Voting Response enrolment.** majority voting To achieve a $P_r^{\text{fail}} < 10^{-6}$, seven BCH(255,21,55) blocks are required when the conventional single reference response under 25 °C is used, whereas six smaller BCH(127,22,23) blocks are adequate when 3MRR under –15 °C, 25 °C, 80 °C is used. In this context, the MR²FE with 3MRR reduces clock overhead by 45.44% in comparison with the conventional FE.
- **Pre-selection Response enrolment.** To achieve a $P_r^{\text{fail}} < 10^{-6}$, nine BCH(127,15,27) blocks are required when the conventional single reference response under 25 °C is used. Conversely, eight smaller BCH(63,16,11) blocks are adequate when 3MRR under –15 °C, 25 °C, 80 °C are used. In this context, the MR²FE with 3MRR reduces clock overhead by 42.14% in comparison with the conventional FE.

MR²FE still greatly outperforms the conventional FE with SRR. The reason lies on the fact that MR²FE alleviates the demanding for a large BCH code block for decoding (BCH decoder

Figure 5.7: RFE-based key derivation scheme between the token \mathcal{T} and server S .

complexity grows approximately with the square of the block size [200]). Therefore, given same key failure rate and a moderate number J of enrolled models, for example $J = 3$, MR²FE still consumes significantly less overhead even when it is the PUF device that performs J recovery attempts. Overall, the overhead experimental results in Table 5.3 demonstrates that the efficiency gains from the MRR methodology also applies to a fuzzy extractor setting.

5.5 Related Work and Discussion

This section first describes the conventional RFE-based key derivation, and conducts security analysis, including helper data manipulation attacks, brute-force attacks and entropy leakage of (reverse) fuzzy extractors with MRR. Then shows that MRR is not specific to SRAM PUFs, but appears to be generic to other silicon PUF types as well. Last, through hash and secure sketch overhead comparisons, we highlight the significance of reducing the overhead of a secure sketch implementation for constructing a lightweight (reverse) fuzzy extractor.

5.5.1 RFE-based Key Derivation

The RFE is beneficial to implementation overhead of the token \mathcal{T} attributing to a cheaper FE.Rep(). The key derivation based on the RFE is thereby enabled and firstly proposed by Van

5.5 Related Work and Discussion

Herrewewe et al. [66], later improved by Maes [197]. In Figure 5.7, it depicts Maes’s RFE-based key derivation protocol [197]. Notably, the shaded $\mathbf{k} \leftarrow \text{Hash}(\mathbf{r}')$ in Figure 5.7 is not explicitly utilised in [197], whereas the \mathbf{r}' itself is treated as a shared key between the server \mathcal{S} and the token \mathcal{T} . Herein, instead of using \mathbf{r}' that might not be uniformly distributed—not having full bit-entropy, we adopt a hash to extract a key \mathbf{k} with full bit-entropy.

During the one-time enrolment phase, a response \mathbf{r} is registered by the server \mathcal{S} and saved in the database (DB). During the key derivation phase, the token \mathcal{T} computes a helper data $\mathbf{p} \leftarrow \text{FE.Gen}(\mathbf{r}')$, where \mathbf{r}' is the reproduced response. The server \mathcal{S} receives the public \mathbf{p} and uses the registered \mathbf{r} to restore the $\mathbf{r}'' \leftarrow \text{FE.Rep}(\mathbf{p}, \mathbf{r}')$. Only when the distance between \mathbf{r}' and \mathbf{r} is small—smaller than a threshold d , can $\mathbf{r}'' = \mathbf{r}'$. Only the token \mathcal{T} and the server \mathcal{S} have the knowledge of the \mathbf{r}' , thus, the $\mathbf{k} \leftarrow \text{Hash}(\mathbf{r}')$ with \mathbf{k} is a shared session key. The key derivation is assisted by nonces \mathbf{n}_t and \mathbf{n}_s that are generated by the token \mathcal{T} ’s and the server \mathcal{S} ’s RNG, respectively. Nonces are responsible for halting replaying attacks.

The RFE should hold two properties: correctness and security.

- Correctness means that the \mathbf{r}' will be successfully recovered based on the \mathbf{r} and the \mathbf{p} through $\mathbf{r}' \leftarrow \text{FE.Rep}(\mathbf{r}, \mathbf{p})$ only on the condition that $\text{FHD}(\mathbf{r}, \mathbf{r}') \leq \frac{d}{|\mathbf{r}|}$, $\text{FHD}()$ evaluates fractional Hamming distance (FHD) between two binary vectors.
- Security implies that given the exposed \mathbf{p} , there are enough entropy left in the \mathbf{r}' .

This chapter focuses on the correctness requirement as we are aiming to significantly reduce the $\text{FE.Gen}()$ implementation overhead on a token \mathcal{T} based on the MRR method. We are not intending to invent a methodology to enhance the security of RFEs or the RFE-based key derivation mechanism, we retain their security properties [201]–[203]. Nonetheless, for the sake of completeness, we discuss the security of (reverse) fuzzy extractors in Section 5.5.

5.5.2 Helper Data Manipulation Attack

Delvaux et al. [131], [204], first introduced HDM attacks, although not on helper data generated from fuzzy extractors. In [73], HDM attacks are applied on a soft-decision error correction decoding. Here, the adversary \mathcal{A} sends manipulated helper data to the PUF key generator and observes key recovery failures. Over multiple queries, the adversary \mathcal{A} learns information about the PUF response, which eventually allows the adversary \mathcal{A} to recover the response. To prevent such an attack, one potential countermeasure is to check the integrity of helper data [73]. During

the key enrolment phase, the helper data \mathbf{p} and the enrolled response \mathbf{r} are hashed together to produce the hash value \mathbf{u} . The hash value \mathbf{u} is validated during the key recovery phase. Consequently, a helper data manipulation attack will always fail because the adversary \mathcal{A} is unable to provide a valid hash \mathbf{u} since the adversary \mathcal{A} has no knowledge of the response \mathbf{r} .

Becker [74] recently revealed a new HDM attack strategy against robust fuzzy extractors. In general, instead of attempting to recover the secret PUF response, the HDM attack attempts to set the PUF response corrected by the key generator to a response $\mathbf{r}_{\mathcal{A}}$ predetermined by an adversary \mathcal{A} . Consequently, the adversary \mathcal{A} attempts to defeat the helper data integrity checks by crafting a hash value $\mathbf{u}_{\mathcal{A}}$ and helper data $\mathbf{p}_{\mathcal{A}}$ in an attempt to manipulate the PUF key generator with a high probability of producing the response $\mathbf{r}_{\mathcal{A}}$ crafted by the adversary \mathcal{A} during the response error correction process. Such an HDM attack now allows an adversary \mathcal{A} to impersonate the PUF integrated device. Further, Becker's extended HDM attack allows the adversary \mathcal{A} to recover the original response \mathbf{r} , and the secret key. Various error correction codes including Reed-Muller codes [205], [206] based on different decoding strategies, soft-decision codes [114], [206] and even-numbered repetition decodes [206] are examined and shown to be vulnerable [74].

A generic countermeasure against Becker's HDM attacks does not yet exist and remains an open challenge [74]. However, the ability to mount the attacks depends on: i) the error correction code employed; and ii) the method used for error correction. Becker [74] shows linear BCH code-based syndrome decoding is immune to HDM attacks; an adversary \mathcal{A} is unable to set a specific response, although helper data may be manipulated to cause the corrected response bits to flip²². The evaluations in this chapter on MR²FE and MR³FE is built upon BCH codes and employ syndrome decoding secure under HDM attacks. Nevertheless, the method we propose is *agnostic* to the fuzzy extractor employed, because we do not rely on any specific code or decoding strategy.

Further, recall that helper data is manipulated and sent to the PUF device, while the adversary \mathcal{A} tests the key failure to determine their success. This implies that the adversary \mathcal{A} is able to conduct an arbitrary number of queries albeit less than the complexity of a brute force attack or random guessing. In a reverse fuzzy extractor setting, it is the PUF device or token \mathcal{T} performing the FE.Gen() operation to generate helper data. Thus, whenever a HDM attack is orchestrated by an adversary \mathcal{A} , the manipulated helper data is sent to the server \mathcal{S} . Consequently, HDM attacks target the server \mathcal{S} in the context of a reverse fuzzy extractor. In this setting, the HDM attack is very likely to be detected by the resourceful server \mathcal{S} because of the abnormal key reconstruction

²²A detailed discussion and a proof that syndrome-based decoding is immune from the HDM attacks presented by Becker can be found in Section 6.1 of the article in [74]. Therein, Becker also derives a security criterion to validate the immunity of a decoding method against the HDM attacks presented in [74].

failure rates resulting from the tampered helper data. It is not unreasonable to conjecture that in a reverse fuzzy extractor setting, and when faced with a more intelligent server \mathcal{S} , an adversary \mathcal{A} will lose the assumed ability to apply an arbitrary number of queries.

5.5.3 Entropy Leakage

Given a secure sketch with BCH(n, k, t) code, entropy leakage is caused mainly from the public helper data. The well-known min-entropy loss is the $n - k$ bound given the exposure of helper data. This $n - k$ bound is conservative. Research studies have explored the derivation of a tight bound for min-entropy loss [73], [152], [201]. However, calculation of min-entropy loss using the tight bound in [152] requires undertaking exhaustive simulations as a straightforward analytical method is not available. The purpose of our work is to demonstrate that the MRR method significantly reduces a token \mathcal{T} 's (reverse) fuzzy extractor implementation overhead. Therefore, we consider the conservative ($n - k$) min-entropy loss bound where the public helper data generated by the BCH(n, k, t) code leaks ($n - k$)-bit entropy [152], [201]. Then, taking response bias b into account (more details please refer to Section 2.3.2), the residual min entropy H_∞ of the n -bit response \mathbf{r} conditioned on the public helper data \mathbf{p} can be expressed as:

$$H_\infty = n \cdot \log_2(\max(b, 1 - b)) - (n - k) \quad (5.7)$$

The reverse fuzzy extractor and a conventional fuzzy extractor might not provide identical security guarantees expressed by equation (5.7). This is because a reverse fuzzy extractor can result in unanticipated entropy loss under repeated helper data exposure associated with a given PUF response \mathbf{r}' ; unless, PUF responses demonstrate a symmetry property. In other words, the one-probability, the probability of a given bit attaining a binary one value, of PUF responses is a symmetric distribution [167]; alternatively, are unbiased. Generally, the extra entropy loss is a result of the leakage of bit-specific reliability information [152].

The extra entropy loss is important only when PUF response bias is considerably different from the ideal value of 50% as shown by the analysis in [152], [207]. We can see that for the SRAM PUFs tested in our work, the extra entropy loss is very small. For instance, at 25C the evaluated mean bias of SRAM PUFs we tested is 49.87%. Such a small bias aligns with expectations from modern silicon PUFs according to other studies [197]. In this context, employing a few more response bits in the reverse fuzzy extractor can compensate for the small extra loss in entropy. As observed by Delvaux [207], for a PUF with low bias within [0.42, 0.58], increasing the length of raw responses alone is an effective measure.

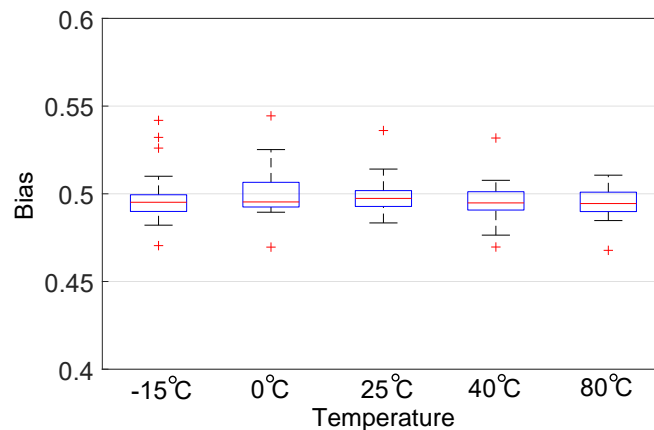


Figure 5.8: Bias of 23 tested SRAM PUFs under five different temperatures.

If the bias is severe, entropy compensation by solely increasing the length of raw responses becomes ineffective. As a result, de-biasing the biased raw responses [201] must be undertaken first, e.g., via classic von Neumann (CVN) de-biasing, pair-output von Neumann de-biasing with erasures (ϵ -2O-VN). Notably, not all de-biasing schemes offer re-usability—multiple use of the same PUF response—for a reverse fuzzy extractor [152]. Nevertheless, a reverse fuzzy extractor with MRR is naturally compatible with de-biasing schemes that offer re-usability.

We also tested the SRAM PUF bias in our dataset under the five different operating temperatures. We want to examine if there is a relationship between the bias—fraction of ‘1’s—and the temperature. If there are more ‘1’/‘0’ responses, indicative of severe bias, when the temperature is higher or lower, then the temperature might cause an unanticipated entropy loss. The bias of 23 SRAM PUFs under varying temperature is detailed in Figure 5.8. We can observe that the mean bias is almost invariant to temperature. Therefore, we can expect that a change in the operating temperature to not lead to additional entropy loss.

5.5.4 Brute Force Attack Complexity under MRR

We recognise that under a MRR model, whilst retaining the security properties of a given secure sketch, brute force attack complexity will reduce. Consider, using six parallel BCH(127,22,23) codes as outlined in Table. 5.2 and Table. 5.3. According to equation (5.7), we can obtain a key \mathbf{k} having a 129-bit entropy given a bias of 0.4987 obtained from SRAM PUF test data. Therefore, without knowledge of \mathbf{k} , the probability of an adversary \mathcal{A} succeeding in a brute-force attack to determine \mathbf{k} is $\frac{1}{2^{129}}$ when a conventional single reference response is employed in the reconstruction. When J multiple response references are employed, the server S or token \mathcal{T} can attempt to reconstruct the response, effectively, J times to obtain the key \mathbf{k} . Correspondingly,

5.5 Related Work and Discussion

the brute-force attack complexity decreases linearly as a function of the number of multiple reference response models J ; in our example, the probability of a brute force attack succeeding is $\frac{J}{2^{129}}$. In general, for J multiple reference responses and min-entropy bound as expressed in equation (5.7), we can express the probability P^{brute} of a successful brute force attack as

$$P^{\text{brute}} = 2^{-H_{\infty} + \log_2(J)}. \quad (5.8)$$

Now, given J MRR models, brute-force attack complexity reduces only slightly while significantly alleviating failure probability $P_{\mathcal{T}}^{\text{fail}}$ and the token \mathcal{T} 's FE.Gen() or FE.Rep() implementation overhead as shown in Table. 5.2 and Table. 5.3, respectively. For example, consider the probability of key failure $P_{\text{fail}} = 2.08 \times 10^{-5}$ for SRR and 5.47×10^{-13} for 3MRR for the BCH (127,22,23) code in our example given in Table. 5.2, when pre-selected response enrolment approached is utilised. Compared to the 3 fold increase in the success of a brute force attack, still extremely low, 3MRR has resulted in more than 10^7 fold decrease in the key failure rate. Further, consider achieving a key failure rate of less than 10^{-6} using SRR. We can see from Table. 5.2 that 10 parallel blocks of BCH(255,13,59) have to be used when a single readout response enrolment is adopted. Considering the implementation overhead of a reverse fuzzy extractor on a token \mathcal{T} to also achieve key failure rate less than 10^{-6} employing MRR, nine smaller BCH(127,15,27) are adequate. In this context, the overhead in terms of CPU clock cycles is reduced by 43% with 3MRR—2,329,519 vs 1,316,184 clock cycles. We can see that the gains in failure rate and implementation overhead more than compensate for the very small reduction in brute force attack complexity.

5.5.5 Generalisability of MRR

We have validated the MRR method with SRAM PUFs where operating conditions that deviate from the reference condition results in responses that exhibit a higher BER than that responses generated at the enrolled reference condition. We can observe that this behaviour agree with other experimentally validated silicon PUFs such as RO-PUFs [208], Latch PUFs, Data flip-flop PUFs, Buskeeper PUF and Arbiter PUFs [197] summarised in Table 5.4 and obtained from published literature. We can see that, in general, the BER increases when the difference between the reference operating condition and the condition under which the response is re-generated increases. We have shown in our study that in this context, our MRR approach provides the capability to select a reference operating condition that is potentially closer to the specific

operating condition of the in-the-field PUF. Therefore, in general, we can expect the MRR approach to provide an implementation efficiency for silicon PUFs.

Table 5.4: BER evaluated as the difference between the enrolled and working operating conditions. The data is obtained from [197].

	(−40 °C, 1.02 V)	(−40 °C, 1.32 V)	Reference condition (25 °C, 1.20 V)	(85 °C, 1.02 V)	(85 °C, 1.32 V)
Latch PUF	23.10%	23.38%	2.61%	10.62%	10.60%
Data Flip-flop PUF	12.79%	12.90%	3.54%	18.10%	17.89%
Buskeeper PUF	9.68%	9.77%	4.16%	17.71%	17.48%
Arbiter PUF	7.41%	5.41%	3.04%	5.23%	5.34
RO PUF	9.01%	7.81%	1.53%	7.11%	8.35%

5.5.6 MRR Enrolment

We observe that an improvement in the error correction efficiency is always achieved with trade-offs; for instance, increasing the enrolment overhead while reducing the key failure rate. We can see in a soft-decision decoding approach as in [61], the PUF-key generator efficiency is enhanced but requires repeated response measurements, in the order of 10 to 100, to collect individual bit’s reliability information as additional helper data [61]. A pre-processing method such as majority voting that can be used with hard decision decoding to reduce key failure rates also requires repeated measurements during the response bit enrolment phase [153].

In our MRR approach, we trade-off the overhead of enrolling multiple reference responses during enrolment with a significantly reduced implementation efficiency on a token \mathcal{T} . More specifically, we can see that our MRR approach requires a moderate increase in enrolment overhead—enrolling J reference responses given J operating conditions²³ while it significantly reduces the encoding/decoding implementation overhead on a PUF token \mathcal{T} .

We can see that MRR facilitates minimising the token \mathcal{T} overhead. We can also see that the corresponding implementation efficiency on a token \mathcal{T} increase the computation burden on a server S because multiple reference responses now need to be evaluated in parallel. Our results demonstrate a small number of reference models, for example $J = 3$ in our evaluation, already greatly improves (reverse) fuzzy extractor implementation efficiency on a token \mathcal{T} . For instance, in the reverse fuzzy extractor case, compared to a single readout response enrolment strategy used to achieve a key failure rate of $P_r^{\text{fail}} < 10^{-6}$, the reverse fuzzy extractor with 3MRR reduces

²³In practice, one can set up several temperature zones in one temperature oven, similar to the reflow oven, or employ an environmental chamber with rapid temperature transition times. Naturally, any such method will incur a time and cost overhead.

the clock cycle overhead by 43%. In addition, the MRR enrolment overhead is only incurred once during the enrolment phase but the benefits from the MRR approach extends to the life of the token \mathcal{T} . In the context of resource-limited devices such as CRFID tokens we have employed in our evaluations, minimising implementation overhead is highly desirable in practice. We can expect a resourceful server \mathcal{S} in such a context to easily manage the increase in computation overhead.

5.5.7 Hash and Encoding/Decoding Overhead

Based on Table E.1, Table E.2 and Table E.3 in Appendix E, we can see that it is paramount to minimise the absolute error correction overhead for not only the FE.Gen() but also FE.Rep() function because the overhead of a hash function is always much less than that of secure sketch. In other words, regardless of whether a RFE or a FE is employed, the FE.Gen() and FE.Rep() implementation overhead is always dominant.

According to the fully implemented PUF-based key generator on an FPGA platform by Maes et al. [124], where concatenated (7, 1, 3) repetition code and a BCH(318, 174, 17) code was used, the BCH(318, 174, 17) and (7, 1, 3) repetition code decoder costs were 112 and 37 FPGA slices, respectively, while the hash implementation of SPONGENT-128 only occupied up 22 slices. The hash function logic overhead was only 15% of the error correction logic. Our code encoding/decoding overhead evaluation agrees with this observation but from a different implementation. In particular, our results are from implementations in software rather than hardware, as in [124]. In fact, in the software implementation of the hash and BCH decoder, in comparison to the hardware implementation, the relative hash overhead is much less than the BCH decoder. We can conclude, based on our results, that a lightweight PUF key generator is very hard to achieve without optimising the error correction coding/decoding overhead regardless of a hardware or software implementation approach. The MRR method we present provides a new approach to substantially minimise the error correction overhead.

5.6 Chapter Summary

This chapter developed the MRR approach to significantly reduce the overhead of RFE and FE implementations and proposed MR³FE and MR²FE for lightweight key derivation and key generation. We validated our approach using a class of ultra low power MCUs employed by a CRFID transponder (WISP 5.1 LRG) as an exemplary resource-constrained device. The

extensive experimental results and analysis in this chapter included SRAM PUF data from the MSP20 dataset and demonstrated that, regardless of response enrolment approaches, (R)FE with MRR will always greatly outperform the conventional (R)FE with a single reference response. Enrolling more reference responses under fine-grained operating conditions can further reduce a token \mathcal{T} 's overhead, specifically, in MR³FE case, because its overhead is dependent on the number of enrolled MRR. The proposed MRR is not only applicable to the studied SRAM PUF but also to other silicon PUF types. Dedicated and specific implementation optimisations, e.g, C code optimisations, can be exploited to further decrease the absolute overhead we have reported in this chapter. We leave discussion for future research direction to Chapter 8.

In the physically obfuscated key derivation method in SecuCode developed in Chapter 3, a very narrow legal operation temperature range from 0 °C to 40 °C is required to achieve a key failure rate as low as 1.6×10^{-3} , as discussed in Section 3.4.1 in Chapter 3. On the one hand, this legal temperature range reduces the applicability of SecuCode. On the other hand, it also requires sampling the on-chip thermometer, i.e., the use of the TEMP() function in SecuCode, and introduces additional overhead. However, if we want to achieve an industrial standard key failure rate below 10^{-6} over the entire -15 °C to 80 °C temperature range (where raw BER = 11%), with the conventional SRR method, the use of a much more computationally intensive RFE encoder as shown in Table 5.5—see SRR—is needed. If we apply the MRR technique at three different temperatures: -15 °C, 25 °C and 80 °C (shown as 3MRR in Table 5.5), a key failure rate of 9.85×10^{-7} that meets an expected industrial standard is achievable. Compared to the SRR-based method, the clock cycles required for MRR-based method is reduced by 45%, while the FRAM (code memory) and SRAM (data memory) usage is also reduced by 5.25% and 26.6%, respectively.

Table 5.5: Comparing the implementation overhead of applying MRR to achieve an extended operational temperature range and an industry standard key reliability.

method	Parameters				Implementation overhead of RFE encoder		
	Temperature range	P_{fail}	(n1,t1,k1)	blocks	CPU	FRAM	SRAM
SecuCode (Chapter 3)	0 °C to 40 °C	1.6×10^{-3}	(31,16,3)	8	146,533	621 bytes	53 bytes
SRR	-15 °C to 80 °C	1.66×10^{-7}	(127,15,27)	9	1,316,184	6,015 bytes	484 bytes
3MRR	-15 °C to 80 °C	9.85×10^{-7}	(63,16,11)	8	722,193	5,699 bytes	355 bytes

Although we have seen the MRR-based methods can reduce the on-device implementation overhead for an RFE encoder function, compared to traditional SRR-based methods, the proposed MRR-based methods, such as the MR³FE, still rely on the underlying RFE to generate reliable keys. The implementation overhead is still too high for adoption into highly resource-limited devices, especially if a full operating temperature range and industrial standard

5.6 Chapter Summary

key reliability are desired. In Chapter 6 we will investigate an alternative to the traditional RFE-based method to further reduce the on-device implementation overhead of the lightweight physically obfuscated key generator by removing the need for an RFE, as previewed in Figure 5.9.

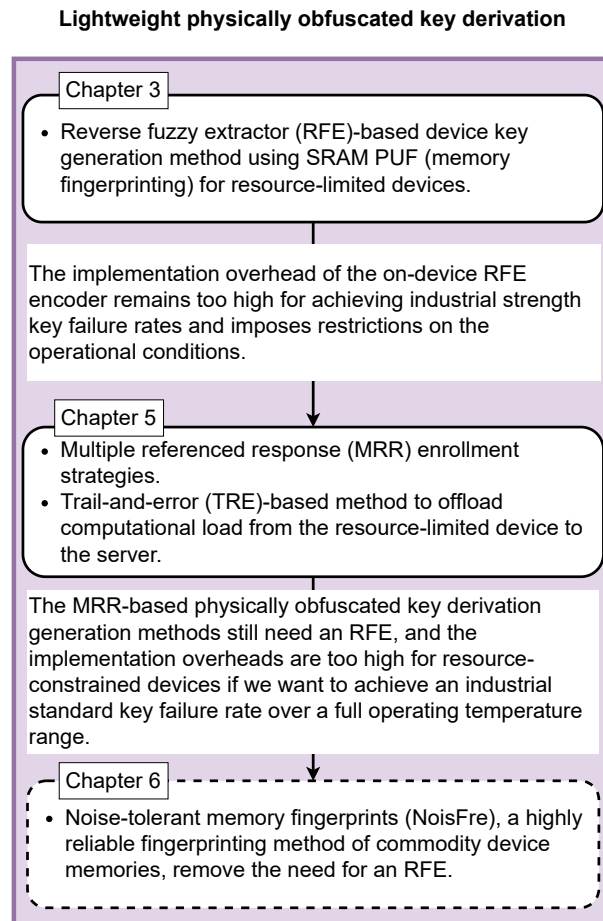
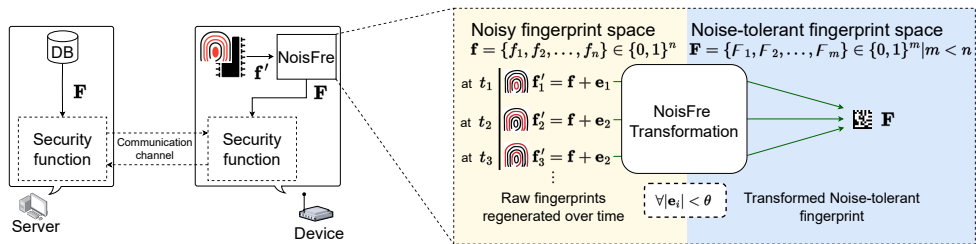


Figure 5.9: Upcoming chapter sneak peek.

Chapter 6

Noise-Tolerant Secret Keys from Device Memory



Building hardware security primitives with on-device memory fingerprints is a compelling proposition given the ubiquity of memory in electronic devices, especially for resource-limited devices where cryptographic modules are often unavailable. However, as highlighted in Chapter 1, Chapter 3 and Chapter 5, the use of fingerprints in security functions is challenged by the small, but unpredictable variations in fingerprint re-generations from the same device due to measurement noise. This chapter formulates a novel and pragmatic approach to achieve highly reliable fingerprints from device memory able to *significantly reduce the overhead* imposed by reverse fuzzy extractor-based key generators employed to reconcile noisy key bits. We investigate the transformation of raw fingerprints into a noise-tolerant space where the generation of fingerprints is intrinsically highly reliable. We derive formal performance bounds to support practitioners to easily adopt our methods for applications. Subsequently, we demonstrate the expressive power of our formalisation by using it to investigate the practicability of extracting noise-tolerant fingerprints from commodity devices. Together with extensive simulations, we have employed 119 chips from five different manufacturers for extensive experimental validations. The results, including an end-to-end implementation demonstration with a low-cost wearable Bluetooth inertial sensor capable of on-demand and runtime key generation, show that key generators with failure rates less than 10^{-6} can be efficiently realised with noise-tolerant fingerprints with a single fingerprint snapshot to support ease-of-enrolment.

6.1 Motivation and Contribution

Due to the unpredictable noise such as thermal noise, supply voltage fluctuations, device ageing, fingerprints generated from the same device at different time instances are not exactly identical. Therefore, the usage of (reverse) fuzzy extractors [70], [71] becomes a common practice to derive reliable secure keys from device fingerprints to be used in a security function.

As seen in the studies in this thesis, the main issue with using the RFE-based methods is the high implementation overheads of the RFE encoder; thus, making it less attractive for resource-constrained platforms, such as the passively powered batteryless CRFID devices. In Chapter 5, a multiple reference response (MRR)-based enrolment strategy was proposed to reduce the complexity of the on-device RFE encoder. However, the implementation overhead is still too high for adoption into resource-limited devices, especially if a full operating temperature range and industrial standard key reliability are desired. Consequently, this chapter will investigate the challenging problems below:

Problem 1. How can we extract *intrinsically reliable fingerprints* from device memories?

Problem 2. If an approach does exist, is the method *pragmatic* and *usable* for fingerprinting memory resources on pervasive commodity computing devices?

Current memory fingerprinting schemes extract a fingerprint bit from each memory cell. Fingerprinting under this scheme is susceptible to noise. To the best of our knowledge, for commodity memories, existing techniques fail to accurately capture the noise-tolerance degree of each raw bit to formally determine those extremely reliable bits for direct key usage without the problematic error reconciliation.

We recognise that device memories are a cost-free and abundant source of entropy. Attributing to the ever-decreasing fabrication costs, the size of memory pervasively embedded within devices has become increasingly large. Hundreds of kilobytes (KiB), even in low-end devices, are common (see the devices we tested in Table 6.2). Consequently, we envision that the entropy of extracted information may be sacrificed for improved reliability. Therefore, we propose the concept of transforming the raw fingerprint space of high information density into a lower-dimensional space with the attribute of being largely invariant to noise—bit flips—observed in the digitised raw fingerprint space or memory biometrics. We refer to this noise-tolerant memory fingerprinting concept as **NoisFre**.

We illustrate our concept in Figure 6.1. Building upon a raw memory biometric source that is a *noisy fingerprint space*, we propose extracting new fingerprints \mathbf{F} in *the deliberately transformed*

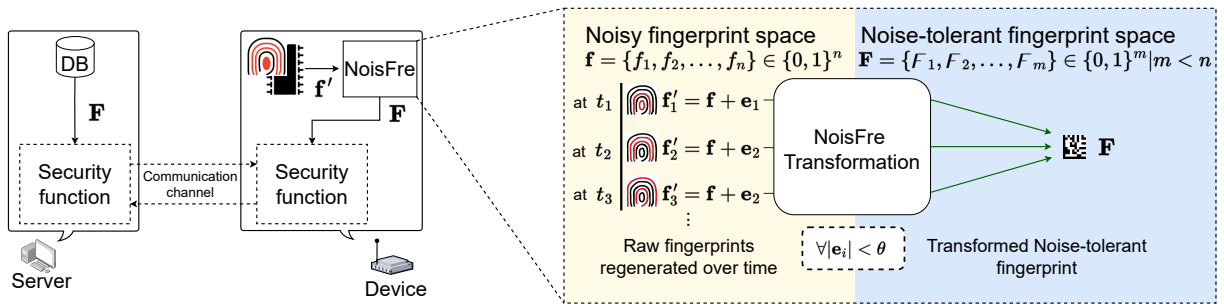


Figure 6.1: Illustrating the use of noise-tolerant memory fingerprints from commodity devices for security functions. We transform the raw fingerprint from an n -dimensional noisy fingerprint space to an m -dimensional noise-tolerant fingerprint space, where $m < n$. In the noise-tolerant space, as long as the noise e_i is less than a bound θ , the re-generated and transformed fingerprint can be correctly projected to the reference transformed fingerprint template \mathbf{F} securely enrolled and stored on the server. Red curves in the *fingerprint symbol* depict errors in the raw fingerprint upon re-generation at time instances t_1 to t_3 . Now, the \mathbf{F} obtained can serve as a root of trust or a root key for a security function.

noise-tolerant fingerprint space, which can tolerate a desirable noise bound θ . Here, as long as the noise e_i (induced number of raw fingerprint bit errors) is less than θ , the re-generated and transformed raw fingerprints are guaranteed to be *projected* to its reference counterpart \mathbf{F} enrolled at the server. More generally, the re-generated and transformed fingerprint \mathbf{F} is insensitive to bit errors (resulted from noise) in the raw fingerprint space. Therefore, it can be directly employed—without error reconciliation—as a root key in a security function, despite the noisy renditions of the raw fingerprints at times t_1 , t_2 , and t_3 .

Significantly, we recognise that the best strategy for fingerprint memory is not always directly from the raw noisy fingerprint space, such as directly treating the power-up state of an SRAM memory of a cell as a fingerprint bit, the foundation for all current memory fingerprinting schemes. We argue for exploiting the freely available, abundant entropy of memories. We do not focus on individual raw fingerprint bits but seek to find an invariant property of a group of raw bits to measure, so we can be less concerned of the complexity about the process generating those bits.

The main contributions of this chapter are summarised as follows:

- NoisFre approach for achieving highly reliable fingerprints.** We exploit the freely available and abundant entropy from memories to propose a *new concept*—NoisFre—for highly reliable fingerprinting of commodity device memories. The principle is based on transforming from a noisy raw fingerprint space to a lower dimensional, noise-tolerant fingerprint space capable of reconciling noise inherent across multiple measurements of the same raw fingerprint. To corroborate the proposed NoisFre concept, we have

developed two specific transformation methods: i) S-Norm and ii) D-Norm (**Problem 1**).

- **Analytical model formulations.** We formulate analytical models with the expressive power to support the design of security functions and evaluate the transformation methods. We express: i) an upper bound for the unreliability of the transformed F bits with respect to the transform function parameters; and ii) the expected fingerprint extraction efficiency—the number of transformed F bits that can be extracted from a given memory size (**Problem 2**).
- **Extensive evaluations to validate the analytical models.** We conduct elaborate and extensive evaluations with a synthetic chip model to obtain the massive number of repeated fingerprint measurements necessary to validate our formalisation of unreliability and extraction efficiency. Billions of repeated measurements were simulated using the synthetic chip model with bit-level modelling capable of capturing bit-error behaviour in SRAM device memories. Our formal models are confirmed to be worst-case bounds in practice (**Problem 2**).
- **Validations with Physical chip tests and dataset contribution.** We extensively test: i) 110 SRAM memory devices from three different manufacturers to experimentally validate NoisFre performance. We focus on SRAM memory, as it is the most commonly embedded memory, especially for low-cost IoT devices. Further, we employ: ii) seven Flash memories and iii) two EEPROM memories for validating the *generalisability* of NoisFre. We release the NRF12 memory fingerprint dataset that we collected and open-source code artefacts to facilitate future research. The download link to the NRF12 dataset is available in Section 1.4 (**Problem 2**).
- **Propose and evaluate NoisFre-based key generators.** To demonstrate the expressive power of our formalisation, we investigate the derivation of a root key—the foundation for realising various security functions. We demonstrate a 128-bit root key with an extremely low key failure rate of less than 10^{-6} can be directly obtained by transformed fingerprints to obviate the need for computationally intensive on-device RFE encoder. Significantly, a fingerprint snapshot or single measurement is sufficient for enrolling a key, a process we follow in all our experiments (**Problem 2**).
- **End-to-end security function design and implementation.** As a case study, we *implement a NoisFre key generator* and a security function on a low-end wearable Bluetooth inertial sensor. We extract a root key directly from native SRAM fingerprints

transformed into noise-tolerant fingerprint for use in a remote attestation primitive. By fundamentally obviating the state-of-the-art method necessary for reconciling noisy key bits, we demonstrate a significant overhead reduction (i.e., 54% compared to reverse FE and 82% compared to FE) and enhanced security. By utilising the power isolation features, we also demonstrate the realisation of *run-time and on-demand generation of robust SRAM fingerprints* \mathbf{F} on this low-end device (**Problem 2**). A video demo is available at:

<https://www.youtube.com/watch?v=O5NWZw-swpw>



scan to watch

6.1.1 Chapter Overview

Section 6.2 describe the NoisFre concept and devises two specific transformation methods. Section 6.3 formulates the reliability and extraction efficiency models of the NoisFre methods. Section 6.4 comprehensively validate the formalisation's through extensive experiments. Section 6.5 investigates the power of the formalisation's and demonstrates that the transformed fingerprints can directly meet the stringent reliability requirement of the cryptographic keys without any post stabilisation operations. Section 6.6 completes an end-to-end implementation case study of a remote attestation function on a low-end Bluetooth device by solely using the transformed fingerprints as the root key. Discussion and related work are provided in Section 6.7 and Section 6.8, respectively. Section 6.9 concludes this chapter.

6.1.2 Notations and Concepts

Adding to the general notations and conventions defined in Section 2.1 in Chapter 2, Table 6.1 summarises some key concepts introduced and referred to in this chapter.

6.2 NoisFre Transformation

We provide the impetus for developing NoisFre fingerprinting and its key insights, followed by two specific and *practical* NoisFre transformation methods.

6.2.1 Our Pragmatic Approach

In contrast to extracting one fingerprint bit from each memory cell, we propose a many-to-one transformation possessing a *property of invariance* to underlying raw bit patterns: more

6.2 NoisFre Transformation

Table 6.1: Table of notations in this chapter.

\mathcal{V}	A Verifier \mathcal{V} is a server and a smart gateway, for more details please refer to Section 6.6.
\mathcal{P}	A Prover \mathcal{P} is a device whose memory contents are validated in the remote attestation.
\mathcal{A}	An adversary \mathcal{A} is an attacker, for the adversary \mathcal{A} 's goal ability please refer to Section 6.7.3.
DB	Verifier's database, where each element is a three-tuple corresponding to each prover \mathcal{P} : i) the unique and immutable identification number id ; and ii) enrolled noise-tolerant fingerprint F and iii) a correspondent mask to the enrolled F (it is optional to stored the mask in DB , it can also be stored on-device).
f	A noisy raw fingerprint vector (for more details, please refer to Section 6.2.1).
e	A vector of error bits in the re-generated raw noisy fingerprint f' .
F	A transformed noise-tolerant fingerprint vector (for more details, please refer to Section 6.1).
mask	Positions of transformed bits F should be provisioned during the key enrolment phase and provided during the key re-generation phase. We refer to these positions using a mask (for more details, please refer to Section 6.5.1).
tag	A MAC tag check the integrity of the mask (for more details please refer to Section 6.5.1).
chal	A challenge randomly generated with RNG(), that is employed in the remote attestation security function.
resp	A response in the remote attestation security function.
bin	The target application program (App) code bin in the Prover \mathcal{P} 's memory (for more details, please refer to Section 6.6.1).
f	One bit in the noisy raw fingerprint vector f
F	One bit in the transformed noise-tolerant fingerprint vector F .
θ	The noise-tolerance parameter (for more details, please refer to Section 6.2.2 and Section 6.2.3).
n	The group size n (for more details, please refer to Section 6.2.2).
m	The block size m (for more details, please refer to Section 6.2.3).
BER_f	The expected BER of raw noisy fingerprints f .
BER_F	The expected BER of noise-tolerant fingerprints F .
\square'	An apostrophe denotes a quantity evaluated at different time, e.g., f' denotes a reproduced raw noisy response.
$\ \square\ _1$	The ℓ_1 -Norm value, which is the distance of the vector from an all-zero vector—or the Hamming weight of a vector, as described in Definition 6.2.1.
RNG()	A random number generator RNG() outputs a random number when invoked.
MAC()	MAC() is a message authentication code function. For example, $\mathbf{tag} \leftarrow \text{MAC}_F(\mathbf{mask})$ computes a MAC tag from the mask to ensure the <i>mask</i> integrity of the mask.
NoisFre.Transform()	The NoisFre transformation function described in Section 6.2.
Memory()	The Prover \mathcal{P} 's device memory, denotes the memory region used for device fingerprint extraction.
WORM	A write-once-read-many memory.

generally, invariant to the unpredictable, complex and dynamic raw fingerprint generating processes. Our desire is to project all of the fingerprint measurements conducted from the same

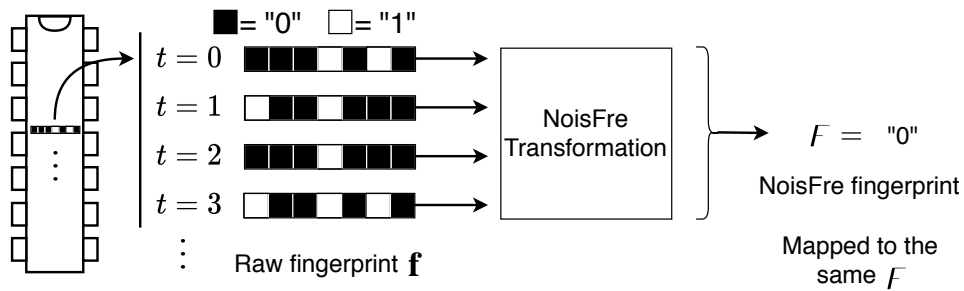


Figure 6.2: NoisFre transformation concept. A group of raw bit vectors exhibiting errors measured at different times can be transformed into the same NoisFre fingerprint bit F (e.g., ‘0’), attributing to the *invariance of the transform* patterns to: i) *permutations* of raw bit vectors (e.g., at time $t = 0$ and $t = 1$); and ii) vectors with different *combinations* of raw bits (e.g., at times $t = 2$ and $t = 3$)

block of memory at different time instances to the *exact* same transformed bit, $F \in \{0, 1\}^1$. The concept is illustrated in Figure 6.2. Note that:

- Bit errors leading to permutations of raw bits—where the positions of ‘1’ and ‘0’ values change in a bit vector but the number of ‘1’s and ‘0’s do not, as seen at $t = 0$ and $t = 1$ in Figure 6.2—are projected to the *exact* same transformed bit $F \in \{0, 1\}^1$.
- Bit errors leading to vectors constituting different combinations of ‘1’s and ‘0’s (e.g., the fingerprint from the same block of memory at time $t = 1$ with two ‘1’ binary bits and that re-generated at time $t = 2$ with only one ‘1’ binary bit in Figure 6.2) are projected to the *exact* same transformed bit $F \in \{0, 1\}^1$.

We observe that a transformed bit, F , is able to mitigate the impact from multiple raw fingerprint bit errors manifesting as permutations or combinations of an n -bit raw fingerprint, \mathbf{f} . The concept we propose is surprisingly simple but efficient and practical because of the *important but inadvertent* reality of large memory volumes intrinsic to devices. From a practical consideration, our critical insight is that memory embedded within modern electronics is large and *provides abundant entropy* to be exploited without additional costs for security functions. This fact is the foundation for our NoisFre transformation method: *trade-off entropy for reliability*.

This chapter proposes two specific NoisFre transformation methods: Single $\ell 1$ -Norm (S-Norm) and Differential $\ell 1$ -Norm (D-Norm).

6.2.2 Single $\ell 1$ -Norm Transformation (S-Norm)

The $\ell 1$ -Norm of a vector is the distance of the vector from an all-zero vector—or the Hamming weight of a vector, as described in Definition 6.2.1.

6.2 NoisFre Transformation

Definition 6.2.1 (ℓ_1 -Norm). Let \mathbf{f} be a binary vector length n representing a noisy raw fingerprint where f_j is the j^{th} bit in \mathbf{f} ; then the ℓ_1 -Norm of \mathbf{f} is defined as:

$$\|\mathbf{f}\|_1 \triangleq \sum_{j=1}^n f_j. \quad (6.1)$$

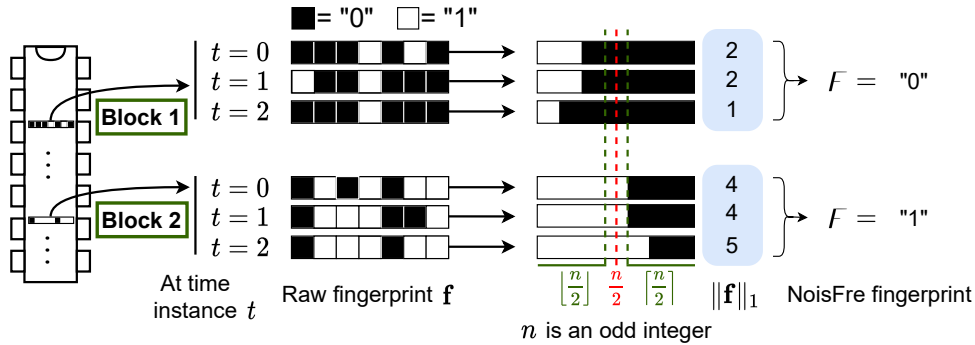


Figure 6.3: NoisFre transformation via S-Norm; the transformed bit F is extracted using the ℓ_1 -Norm of a group of raw bits. The group size is an odd integer number (e.g. 7 in this illustration) to ensure a balance between zeros and ones in the F bits. We illustrate the re-generated raw fingerprints from two blocks of memory—**Block 1** and **Block 2**—at times $t = 0, 1,$ and 2 .

Interestingly, an ℓ_1 -Norm of a vector is permutation invariant. Hence, a new bit F can be obtained by applying an ℓ_1 -Norm over a raw fingerprint vector \mathbf{f} as $\|\mathbf{f}\|_1$ and as described by the S-Norm below.

Definition 6.2.2 (S-Norm). Let the i -th raw fingerprint bit vector of n bits, where n is an odd integer, be \mathbf{f}_i . Then S-Norm transform is defined as:

$$F = \begin{cases} 1, & \|\mathbf{f}_i\|_1 \geq \lceil \frac{n}{2} \rceil \\ 0, & \|\mathbf{f}_i\|_1 \leq \lfloor \frac{n}{2} \rfloor \end{cases} \quad (6.2)$$

An illustrative example of S-Norm-based transformation is provided in Figure 6.3. When $\|\mathbf{f}\|_1 > \frac{n}{2}$, where n is an odd integer, the F bit is ‘1’, and otherwise ‘0’. This transform has the first desirable property of being insensitive to bit errors manifesting as permutations of a raw fingerprint \mathbf{f} . For example, despite the raw bit errors at time $t = 1$ for the raw fingerprint from **Block 1** that lead to a permutation in respect to the bit vector referenced at time $t = 0$, the corresponding ℓ_1 -Norm remains invariant; the memory **Block 1** is still projected to $F = ‘0’$. The transform has the second desirable property of being insensitive to bit errors manifesting as combinations of an n raw fingerprint bit vector. Notably, errors that lead to different combinations of raw binary ‘1’ and ‘0’ values can increment or decrement ℓ_1 -Norm but these can still be projected to the same transformed F bit. For example, see ℓ_1 -Norm values

for memory **Block 1** at time $t = 1$ compared to $t = 2$ in Figure 6.3. Hence, S-Norm achieves the two qualities we desire from a transform expressed in Section 6.2.1.

In Figure 6.3, we can expect the transformed F bit from **Block 2** with ℓ_1 -Norms at the decision boundary, defined in equation (6.2), to more likely be affected by error bits within the raw fingerprint, resulting in different combinations of an n -bit raw fingerprint. A resulting combination of n bits with a single change in the number of raw binary ‘1’ bits can lead to an ℓ_1 -Norm projection that crosses the decision boundary. We recognise the resulting F bits from such raw fingerprints to effectively display low reliability. Therefore, we propose winnowing raw fingerprints based on treating ℓ_1 -Norm as a reliability measure for F bits. For this purpose, we define the *S-Norm-based Selection* method described in Definition 6.2.3 and generalise the approach using a noise tolerance parameter θ to provide an upper bound of tolerance on raw bit errors or the combinations of n -bit patterns; the transform will faithfully project to a specific F bit. Interestingly, the evaluation of ℓ_1 -Norm only require a *single measurement*, while a larger θ can be chosen to facilitate higher noise tolerance. Therefore, we propose selecting based on the ℓ_1 -Norm of raw fingerprint vectors obtained from a single measurement defined as being at time $t = 0$. Notably, this approach facilitates rapid characterisation of raw fingerprints from device memories, as ℓ_1 -Norms can be acquired in a single measurement. All our experimental and theoretical analyses assume such a characterisation.

Definition 6.2.3 (S-Norm-based Selection). Let the raw fingerprint in the i -th n -bit block extracted at time t be \mathbf{f}_i^t . Then for a chosen noise tolerance parameter $\theta \in \mathbb{N}^0$, an extracted raw fingerprint vector \mathbf{f}_i^0 is selected at time $t = 0$ if:

$$\|\mathbf{f}_i^0\|_1 \leq \lfloor \frac{n}{2} \rfloor - \theta \quad \text{or} \quad \|\mathbf{f}_i^0\|_1 \geq \lceil \frac{n}{2} \rceil + \theta \quad (6.3)$$

To understand and demonstrate the significant role of the noise tolerance parameter θ in the mitigation of raw bit errors, we consider the distribution of $\|\mathbf{f}^0\|_1$. We used the experimental dataset obtained from Nordic Semiconductor chips detailed in Table 6.2. Figure 6.4 plots the resulting distribution of enrolling measurements (at time $t = 0$) for two cases of a small and a large θ for an \mathbf{f} of $n = 15$ -bit. As expected, the distribution of $\|\mathbf{f}^0\|_1$ approximates a bell curve.

Consider the groups of \mathbf{f}^0 raw fingerprints (green bar) at the boundary of the selection criteria in equation (6.3), where $\|\mathbf{f}_i^0\|_1 = \lceil \frac{n}{2} \rceil + \theta$ for the two cases of a small and large θ . These groups represent those closest to the decision boundary, $\|\mathbf{f}_i^0\|_1 = \lceil \frac{n}{2} \rceil$ (green line), consequently representing those most likely to lead to a bit error in a transformed bit F when raw bit errors changes the ℓ_1 -Norm of \mathbf{f}^0 . When θ is small ($\theta = 2$), a change in more than two bits in a

6.2 NoisFre Transformation

given pattern can lead to $\|\mathbf{f}\|_1$ crossing the decision boundary $\frac{n}{2}$ in a subsequent raw fingerprint extraction, resulting in a F bit flip.

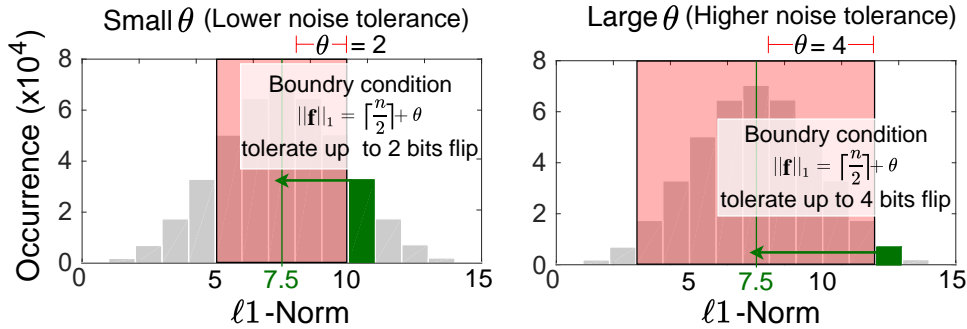


Figure 6.4: Illustrating the role of the noise tolerance parameter θ in S-Norm-based selection. The plots show the ℓ_1 -Norm distribution of raw noisy fingerprints. It approximates normal distribution. A larger θ ensures the transformed F bits can tolerate a higher degree of noise.

In contrast, when θ is large, set to 4, the ℓ_1 -Norm of the selected fingerprint vectors or $\|\mathbf{f}^0\|_1$ are further away from the decision boundary. Consequently, a change in more than four bits in a \mathbf{f} is needed to flip the corresponding F in a subsequent fingerprint evaluation. Such a probability is smaller than a change in more than two raw fingerprint bit flips for vectors selected with $\theta = 2$. Therefore, we can expect the F bits selected employing a larger θ to be significantly more reliable. Our study, thus far, leads to the following observations:

1. The S-Norm $\|\mathbf{f}\|_1$ yields a representation analogous to the reliability of the new bit F .
2. The S-Norm transformed bits are invariant to permutations and combinations of raw bit patterns. Further, θ provides a desirable lower bound on raw bit errors tolerated by the transform.
3. There is an expected trade-off evidence in Figure 6.4. While increasing θ increases the noise tolerance of the transform, it reduces the number of noise-tolerant fingerprint bits extractable from a given memory.

6.2.3 Differential ℓ_1 -Norm Transformation (D-Norm)

Considering *Observation 3* and the distribution in Figure 6.4, we recognise that a distance measure capable of presenting a bimodal distribution could provide an intrinsic separation of groups of underlying raw fingerprint bits with the potential to yield higher numbers of noise-tolerant bits. We hypothesise that a differential distance measure may afford such a desirable distribution and propose the D-Norm transform based on a differential distance measure.

Definition 6.2.4 (D-Norm). Let the lowest and highest ℓ_1 -Norm of m groups (each group is an n -bit vector) be l and h , respectively, where:

$$h \triangleq \arg \max_{\mathbf{f}_i | i \in \{1, \dots, m\}} (\|\mathbf{f}_i\|_1) \tag{6.4}$$

$$l \triangleq \arg \min_{\mathbf{f}_i | i \in \{1, \dots, m\}} (\|\mathbf{f}_i\|_1) \tag{6.5}$$

Now, following the general definition in Section 6.2.1, the D-Norm transform is defined as:

$$F = \begin{cases} 1, & h - l \geq 0 \text{ and } [h] < [l] \\ 0, & l - h < 0 \text{ and } [h] > [l] \end{cases} \tag{6.6}$$

Here, we denote the spatial index i (memory address, in practice) of the vector \mathbf{f}_i chosen for h based on equation (6.4) or l based on equation (6.5) using a square bracket, ‘ $[]$ ’.

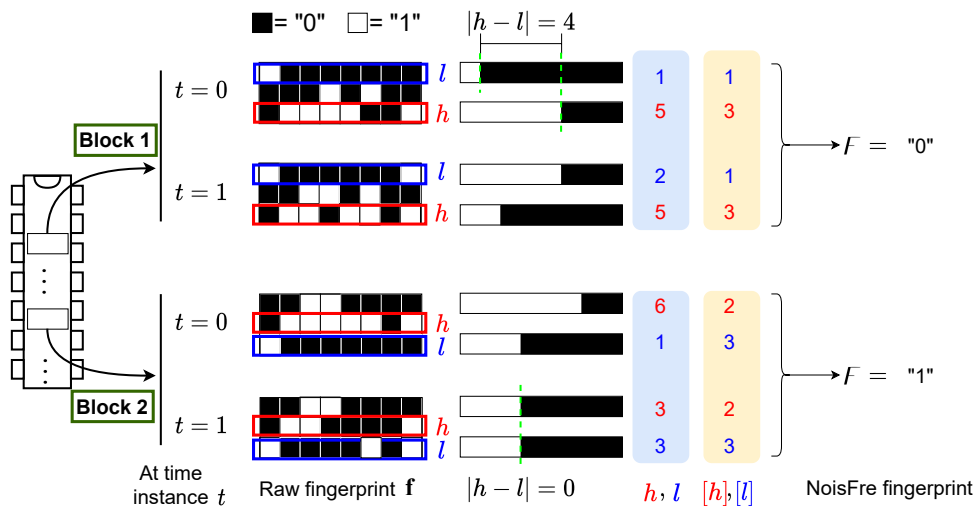


Figure 6.5: D-Norm-based NoisFre transform illustration, where $n = 8$ and $m = 3$. We show the results of reading out two blocks of memory (**Block 1** and **Block 2**). Here, each block ($n \times m$ bits) is formed by accessing three bytes from a byte-level addressable memory, and each block provides a new bit F . Hence, the new bit F is a transformation from a block of raw bits with $m = 3$ groups, where each group is an $n = 8$ bit vector. The raw bit values are measured at two different time instances, $t = 0$ and $t = 1$, from each block to illustrate the manner in which the D-Norm transform is reliable against raw bit error.

The D-Norm-based transformation is illustrated in Figure 6.5. In the illustration, the ℓ_1 -Norm of two blocks of $m = 3$ groups of $n = 8$ bit vectors are evaluated at time $t = 0$. In subsequent evaluations of the fingerprint at $t = 1$:

- In **Block 1**, we can observe the *permutation invariance property*, similar to the S-Norm. For example, the highest ℓ_1 -Norm at $t = 0$ and $t = 1$ is $h = 5$ for the third 8-bit vector despite repeated generation of the raw bits not being exact.

6.2 NoisFre Transformation

- In **Block 2**, we further observe the difference of $h - l$ is $6 - 1 = 5$ at $t = 0$ and shows an *extreme case* of $3 - 3 = 0$ at $t = 1$, where F bit of ‘1’ remains invariant. Which reflects the *combination invariance property*.

In both **Block 1** and **Block 2**, the fingerprint bit F remains robust to the raw fingerprint bit error patterns observed at different measurement times. However, a combination of n bits with a single change in the number of raw binary ‘1’ bits can lead a D-Norm projection at the proximity of the decision boundary in equation (6.6) to cross that boundary. Hence, the resulting F bits from such raw fingerprints effectively display low reliability. Therefore, similar to S-Norm, we propose winnowing raw fingerprints based on their $|h - l|$ projections. We describe the *D-Norm-based Selection* method in Definition 6.2.5 and generalise the approach using a noise tolerance parameter θ to bound the combinations of bit patterns the transform needs to tolerate, using the differential ℓ_1 -Norm of the raw fingerprint vectors measured once (i.e., at $t = 0$).

Definition 6.2.5 (D-Norm-based Selection). From a block of m different n -bit raw noisy fingerprint vectors \mathbf{f}_i^0 for $i \in \{1, \dots, m\}$ extracted at $t = 0$, the block is selected for fingerprinting the device using D-Norm if h and l as defined in equation (6.4) and equation (6.5) satisfy:

$$|h - l| \geq \theta \quad (6.7)$$

To understand the significance of the D-Norm-based selection method and the role of the noise tolerance parameter θ , we employ the Nordic Semiconductor chip fingerprint dataset used in S-Norm. The resulting distribution of enrolling measurements (at time $t = 0$) for two cases of a small and a large θ for blocks of $n \times m$ raw fingerprint bits is shown in Figure 6.6. Interestingly, the distribution of $|h - l|$ approximates a bimodal distribution; each mode represents those vectors mapping to $F = \text{‘1’}$ and ‘0’, respectively. *Importantly, the two clear groupings of $n \times m$ bit blocks based on the D-Norm distance measure results in an intrinsic separation.*

Now, consider the blocks of $h - l$ raw fingerprint bit vectors (green bar) in blocks at the boundary of the selection criteria, in equation (6.7), where $|h - l| = \theta$ for the two cases of a small and a large θ . These blocks of bits represent those most likely to lead to a bit error in a transformed bit F . When θ is small, e.g. $\theta = 2$, two bit flips in the raw fingerprint in a subsequent measurement is enough to push $|h - l|$ across the $h - l = 0$ decision boundary, defined in equation (6.6), and result in a F bit flip. In contrast, when θ is large, e.g. $\theta = 4$, at least five raw fingerprint bit changes are required to flip the F bit in a subsequent evaluation. Therefore, we can expect the F bits selected upon a larger θ to be more reliable.

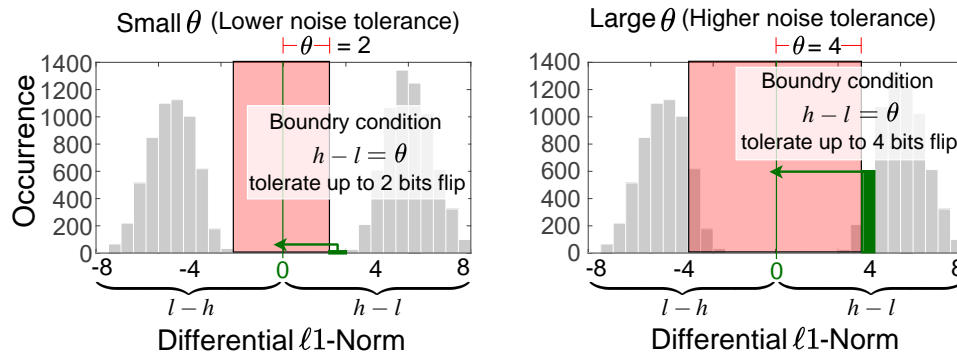


Figure 6.6: The role of the noise tolerance parameter θ in D-Norm-based selection. The plots depict the differential ℓ_1 -Norm distributions—the difference between the h and l —of raw fingerprints. The distribution of differences leads to a bimodal distribution. Here, $n = 16$ and $m = 32$, where $n \times m$ raw bits are transformed into a 1-bit F . If $[h]$ has a lower memory address than $[l]$, the D-Norm would be $h - l$ which is positive and plotted on the right half of the x-axis (otherwise, left). The reliability of F increases when the ℓ_1 -Norm difference between selected groups is away from 0. *Significantly, the proportion of F that tolerates a chosen noise tolerance θ is greatly increased under D-Norm compared to the S-Norm.* For example, the uncovered (grey area) for D-Norm is much larger than that of S-Norm in Figure 6.4 for the same θ values.

The D-Norm method effectively sacrifices more of the available entropy ($n \times m$ raw bits are transformed into 1-bit F) than S-Norm. However, the differential distance measure $h - l$ is bimodal and, thus, *D-Norm is expected to yield a significantly higher number of noise-tolerant F bits.*

6.3 Formalising Performance Measures

We now formulate and derive analytical models to: i) provide an upper bound for the unreliability of noise-tolerant fingerprint bits; and ii) evaluate the expected number of noise-tolerant fingerprint bits that can be extracted from each of the transform methods—the *extraction efficiency*. We summarise the analytical formulations from our **detailed derivations differed to Appendix F.1** for interested readers.

6.3.1 Reliability

We employ the well-known measure of bit error rate (BER) to quantify the reliability of transformed fingerprint \mathbf{F} , more details please refer to Section 2.3.2.

6.3 Formalising Performance Measures

S-Norm Reliability. The expected BER of noise-tolerant fingerprints BER_F of the S-Norm transformation is formulated in equation (6.8); we defer details of the derivation to Appendix F.1.2:

$$\text{BER}_F = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - \theta} \left((1 - \text{binocdf}(\theta + i, \lfloor \frac{n}{2} \rfloor + \theta, \text{BER}_f)) \times \text{binopdf}(i, \lfloor \frac{n}{2} \rfloor - \theta, \text{BER}_f) \right) \quad (6.8)$$

Here, binopdf and binocdf are density and cumulative density functions of a binomial distribution, respectively. The BER_F is a function of θ , n and the expected BER of the raw noisy fingerprint bits, BER_f . If we select the worst-case BER_f of any memory chip, equation (6.8) provides a worst-case (upper-bound) assessment of BER_F (for more details of expected BER and worst-case BER, please refer to Section 2.3.2 in Chapter 2).

D-Norm Reliability. A D-Norm transform employs a block of m groups—each group with n raw fingerprint bits—to be transformed into a 1-bit F . The expected BER of noise-tolerant fingerprints BER_F of D-Norm is expressed in equation (6.9); we defer the derivation of the formula to Appendix F.1.3:

$$\text{BER}_F = \sum_{i=0}^{n-\theta} \left((1 - \text{binocdf}(\theta + i - 1, n + \theta, \text{BER}_f)) \times \text{binopdf}(i, n - \theta, \text{BER}_f) \right) \quad (6.9)$$

We can observe the reliability of transformed bits from the D-Norm to be related to θ , n and BER_f . Again, equation (6.9) provides an upper-bound estimation when the worst-case BER_f is assumed. Notably, the BER_F is independent of the number of groups m within the block.

6.3.2 Extraction Efficiency

We define extraction efficiency η as the number of obtainable transformed bits, F , *subject to a given noise-tolerance* θ , from the total number of available memory bits expressed in KiB.

S-Norm Extraction Efficiency. The extraction efficiency of S-Norm can be expressed as below; the detailed derivation is deferred to Appendix F.1.4:

$$\eta_{\text{SNorm}} = \frac{1}{n} \times \left(1 - \text{binocdf}(\lfloor \frac{n}{2} \rfloor + \theta, n, 0.5) + \text{binocdf}(\lfloor \frac{n}{2} \rfloor - \theta - 1, n, 0.5) \right) \times (1024 \times 8) \quad (6.10)$$

Table 6.2: Memory Datasets.

Manufacturer Model ¹	Abbr	Tech Node	Memory Type	Memory Size	Quantity	Repeat Times	Operating Range ²	Enrolling Condition	Worst Condition	Worst BER _f
Nordic (ours) nRF52832	NRF12	55 nm	SRAM	64 KiB	12 + 88 ⁶	100	−15-80 °C	25 °C	80 °C	6.09%
ISSI [138], [209] IS61WV25616BLL	ISSI4	110 nm	SRAM	256 KiB	4	30	25-80 °C	25 °C	80 °C	8.29%
IDT [138], [209] IDT71V416S	IDT6	130 nm	SRAM	512 KiB	6	50	25-80 °C	25 °C	80 °C	5.42%
Winbond [54] W29N02GV	WINB7	46 nm	FLASH	69,696 Bytes/ 256 MiB ³	7	99	0-100,000 P/E Cycles	0th P/E Cycle	after 100,000th P/E Cycles ⁴	16.26%
Microchip (ours) 24LC256	MICRO2	350 nm	EEPROM	2 KiB/ 32 KiB ⁵	2	100	14-80 °C	14 °C	80 °C	16.37%

¹The NRF12 and MICRO2 datasets we collected will be released, remaining public datasets are from <https://www.trust-hub.org/data>.

²Notably these public datasets focus on room temperature and high-temperature evaluations. Other operating corners are incomplete.

³The tested memory size in the public WINB7 dataset is 69,696 bytes, while the total memory size is 256 MiB.

⁴Experimental studies demonstrate that the BER_f of Flash memory is mainly affected by the programming/erase (P/E) cycles, equivalent to wear-out or aging, but negligibly affected by voltage and temperature [54]. The maximum endurance is 100,000 according to the datasheet.

⁵The EEPROM chip has 32 KiB capacity, while the first 2 KiB memory is evaluated here.

⁶In addition to the 12 chips from the NRF12 dataset with three corner measurements {−15, 25, 80} °C, we evaluated 88 additional chips with a single 25 °C corner measurement.

Here, the term $1 - \text{binocdf}(\lfloor \frac{n}{2} \rfloor + \theta, n, 0.5)$ expresses the case when the ℓ_1 -Norm of an n -bit \mathbf{f} is larger than the selection threshold $\lfloor \frac{n}{2} \rfloor + \theta$, assuming that the probability of each bit being ‘1’/‘0’ is 50%. While the term $\text{binocdf}(\lceil \frac{n}{2} \rceil - \theta - 1, n, 0.5)$ formulates the alternative case when the ℓ_1 -Norm of a n -bit \mathbf{f} is less than or equal to $\lceil \frac{n}{2} \rceil - \theta - 1$. Both cases comprise vectors that satisfy the selection criterion in equation (6.3). We can see that the overall extraction efficiency should be the sum of the above two cases divided by n —recall that n raw bits transform into a 1-bit F . The 1024×8 term expresses the extraction efficiency as bit/KiB—number of selected reliable bits F out of 1 KiB memory.

D-Norm Extraction Efficiency. A D-Norm transform obtains a 1-bit F from a block of m , n -bit raw fingerprint vectors. We define the probability that a given block will meet the selection criterion ($|h - l| \geq \theta$) in equation (6.7) as $P_{\text{DNorm}}^{\text{select}}$ (recall that we refer to the lowest ℓ_1 -Norm as l , and the highest ℓ_1 -Norm as h , out of all m groups within a block). The direct derivation of $P_{\text{DNorm}}^{\text{select}}$ is non-trivial. Instead, we use a different but equivalent problem and defer the details to Appendix F.1.5. We formulate the extraction efficiency of D-Norm as:

$$\eta_{\text{DNorm}} = \frac{1}{n \times m} \times P_{\text{DNorm}}^{\text{select}} \times (1024 \times 8) \quad (6.11)$$

Here, the term of $\frac{1}{n \times m}$ expresses $n \times m$ raw bits producing a single F bit, while the 1024×8 constant facilities express the result in terms of bits/KiB of memory. Given the complexity of

6.4 Experimental Validations

formulating η_{DNorm} , the fitness of the formalised expression is further validated through running extensive numerical experiments (defined in Section 6.4), with the results detailed in Figure F.3 in Appendix F.1.

6.3.3 Summary

Our formulation of *reliability* allows a security practitioner to evaluate, for a given transform, suitable transform parameters (e.g., the number of bits n to employ in a raw fingerprint vector \mathbf{f} and noise tolerance parameter θ) for extracting new fingerprint \mathbf{F} . The extracted \mathbf{F} will have an expected worst-case error bound given by $\text{BER}_{\mathbf{F}}$. Then, the η yields the total number of such noise-tolerant bits $\text{BER}_{\mathbf{F}}$ that can be extracted from a given memory size.

6.4 Experimental Validations

For comprehensively evaluating NoisFre we used 119 commodity chips consisting of three memory types pervasive in COTS devices, especially in low-end IoT devices and extensive simulation based experiments with billions of bit generations to overcome the practical hurdle of demonstrating extremely low BER and key failure rates with physical chips. In the following:

- We validate our analytical models for *reliability* and *extraction efficiency*.
- We we assess the performance of the noise-tolerant fingerprints by evaluating the uniqueness and uniformity of \mathbf{F} .

6.4.1 Evaluation Approaches

We consider three evaluation approaches described below.

Predictions (Analytical model). In this evaluation, we use the analytical models formalised in Section 6.3 to predict extraction efficiency and the $\text{BER}_{\mathbf{F}}$ of the transformed fingerprints.

Simulations (Synthetic chip model). To evaluate the reliability of the transformed fingerprint, a massive number of repeated measurements and the management of the data for analysis are required. For example, if we want to validate whether a 128-bit NoisFre enabled key can achieve a failure rate of 10^{-6} as done in Section 6.5.2, the $\text{BER}_{\mathbf{F}}$ needs to be no more than 7.81×10^{-9} .

To test this with physical measurements, approximately 10^8 repeated measurements are required from the same chip instance. Such a measurement process would take more than nine years and generates roughly 6 Terabyte (TB) of data—this is merely for one 64-KiB chip. Such a massive testing regime is impractical, as detailed in **Measurement (Physical chips)**. Instead, we employed a synthetic memory chip model (detailed in Appendix F.1.1). The model follows the *physical unclonable function (PUF) response model* summarised in [126] and assumes each bit to have a binomial probability of a bit flip across repeated measurements based on employing a *worst-case* BER measured from a physical SRAM chip (see Table 6.2) as the binomial probability parameter value for \mathbf{p} . Using the synthetic chip model, for instance, 100 million (10^8) times of simulations can be completed in approximately 53 hours or 2.2 days using a laptop equipped with quad-core Intel Core i7-10510U CPU and 16 Gigabyte (GiB) RAM. The synthetic chip, models bit errors and when applied with the worst-case BER is sufficient for evaluating reliability and extraction efficiency. Therefore, we employ the data from the simulated measurements to determine η and the BER_F .

Measurements (Physical chips). Performing massive testing on physical chips is impractical. For example, obtaining 100 repeated measurements from an nRF52832 physical chip (in the NRF12 dataset) takes four minutes and 45 seconds, and this generates 6.25 Megabyte (MiB) of data (the SRAM memory size of the single chip is 64 KiB). Then, we can estimate that 100 million (10^8) repeated measurements for a *single* physical chip under a *single* operating corner will take 3,298.6 days (or nine years) and generate 5.96 TB of data. Therefore, we confirm extraction efficiency and the transformed fingerprints' BER validated using the synthetic chip model with a limited number of *repeated* physical chip measurements. However, we dedicate the physical chip measurements across a large batch of 100 chips to evaluate the quality of the transformed bits because properties such as fingerprint uniformity and uniqueness are affected by fabrication variations not incorporated in the synthetic chip model used for simulations. The datasets we used are described in below:

- Specifications of: i) three SRAM; ii) one Flash memory; and iii) one EEPROM datasets are summarised in Table 6.2 and described in detail in Section 2.3.4 in Chapter 2. Each dataset is obtained from chips from a *different manufacturer*. Further, the datasets describe multiple repeated measurements of raw fingerprint bits under *each* operating condition—see the operating range in Table 6.2.
- We use the NRF12 dataset for extensive validations, considering the fact that it is collected with the broadest operating range and highest number of repeated measurements (100

6.4 Experimental Validations

repeated measurements). In addition, we use the remaining datasets to corroborate the generality of our approaches. When we evaluate the BER of transformed bits, $\text{BER}_{\mathbf{F}}$, we report the average from repeated evaluations. Notably, the enrolled reference template is only based on the first *single measurement*.

6.4.2 Validating Extraction Efficiency and Bit Error Rate

We employ simulations with the synthetic chip model to conduct the necessary massive number of repeated measurements to assess the reliability of transformed fingerprint \mathbf{F} . To generate the results, for each parameter combination (i.e., n, θ in S-Norm and m, n , and θ in D-Norm) of a NoisFre transform in Figure 6.7 for S-Norm and Figure 6.8 for D-Norm, we simulated *one million* repeated measurements using a synthetic chip with a memory capacity of up to 16 MiB²⁴

Using Simulations

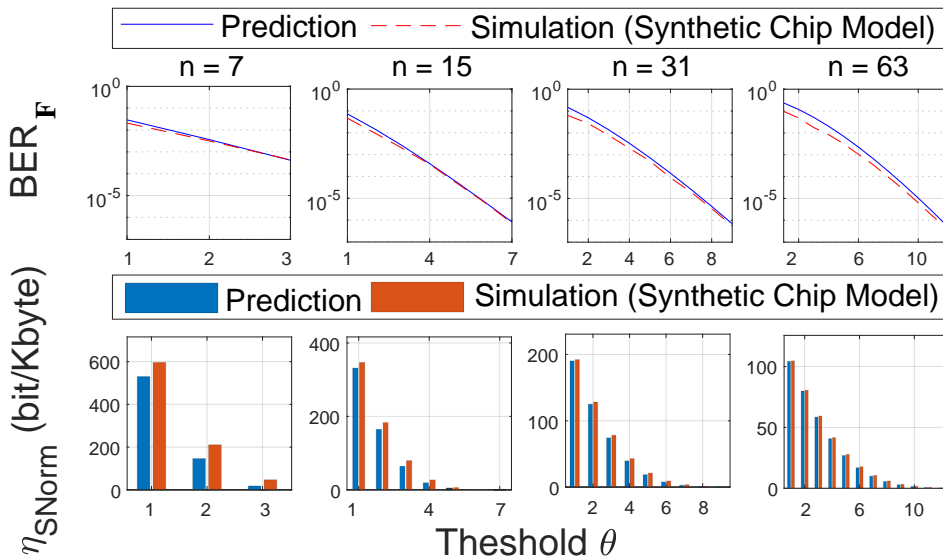


Figure 6.7: S-Norm $\text{BER}_{\mathbf{F}}$ and extraction efficiency η_{SNorm} validation using the synthetic chip model constructed based on the NRF12 chip dataset. The evaluation is conducted for different n and noise-tolerance parameter θ , where θ ranges from 1 to $n/2$. Here, n raw bits are transformed into one F bit.

S-Norm. The evaluation results from S-Norm are detailed in Figure 6.7 under various n and θ settings. Based on Figure 6.7, we can confirm that the formalisation of $\text{BER}_{\mathbf{F}}$ in equation (6.8) provides a conservative estimation of the selected F bits. The results for extraction efficiency

²⁴We start with a memory size of 64 KiB, the SRAM capacity of the nRF52832 chip, but we double the size when a 128-bit \mathbf{F} cannot be obtained. Thus, for each parameter setting, at least 128 bits are ensured to be produced.

are in good agreement with equation (6.10) used to predict the number of F bits that can be expected from a given chip.

D-Norm. The validation results of D-Norm are shown in Figure 6.8. As expected, the BER_F plotted in Figure 6.8 reduces substantially as the θ is increased. Again, we can confirm that the formalised BER_F in equation (6.9) is a conservative estimate because it is always shown to be higher than the synthetic chip model results. Further, the extraction efficiency derived in equation (6.11) provides an accurate prediction of the number of bits of F that can be expected from a given chip under various D-Norm settings (n , m , and θ).

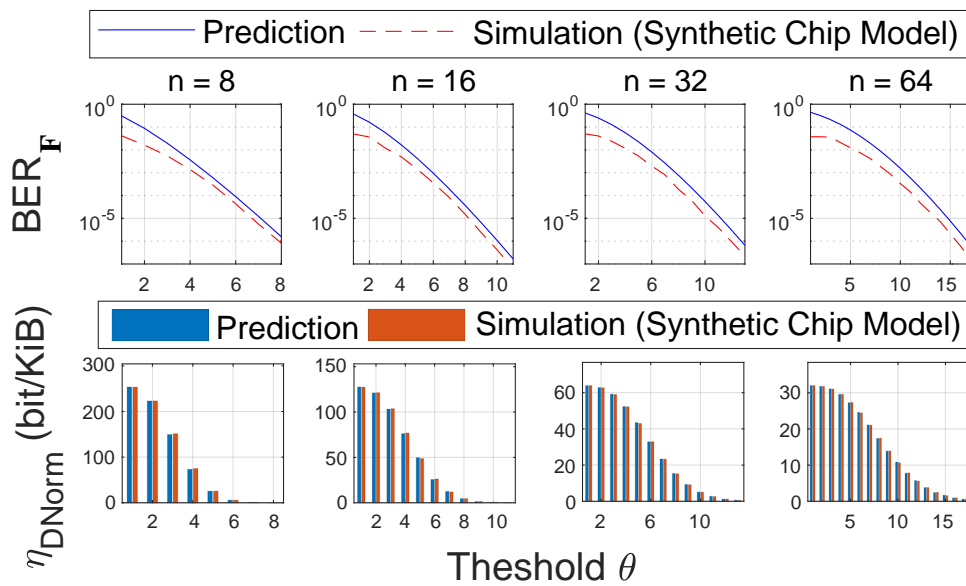


Figure 6.8: D-Norm BER_F and extraction efficiency η_{DNorm} validation on the synthetic chip model constructed based on the NRF12 chip dataset. The evaluation is conducted under different n and noise-tolerance parameter θ where θ ranges from 1 to n and $m = 4$. Here $n \times m$ raw bits are transformed into one F bit.

Using Measurements

Three SRAM datasets (our NRF12, public ISSI4 and IDT6 datasets); one public WINB7 dataset; and our MICRO2 dataset are used to validate the generality of the NoisFre approach based on *physical chip measurements*. The results of S-Norm and D-Norm validated on these five datasets are detailed in Figure 6.9. In contrast to our simulation-based study in Section 6.4.2, here we use a synthetic chip of identical data capacity and worst-case BER_f matching the physical chip under investigation and simulate 100 repeated measurements in sympathy with the physical measurement regime.²⁵ Overall, we can observe from the plots in Figure 6.9 that simulations with the synthetic chip model agree well with the measurements for both S-Norm and D-Norm.

²⁵One hundred measurements are due to the impracticality of conducting the necessary number of repeated measurements with physical chips, as detailed in Section 6.4.1

6.4 Experimental Validations

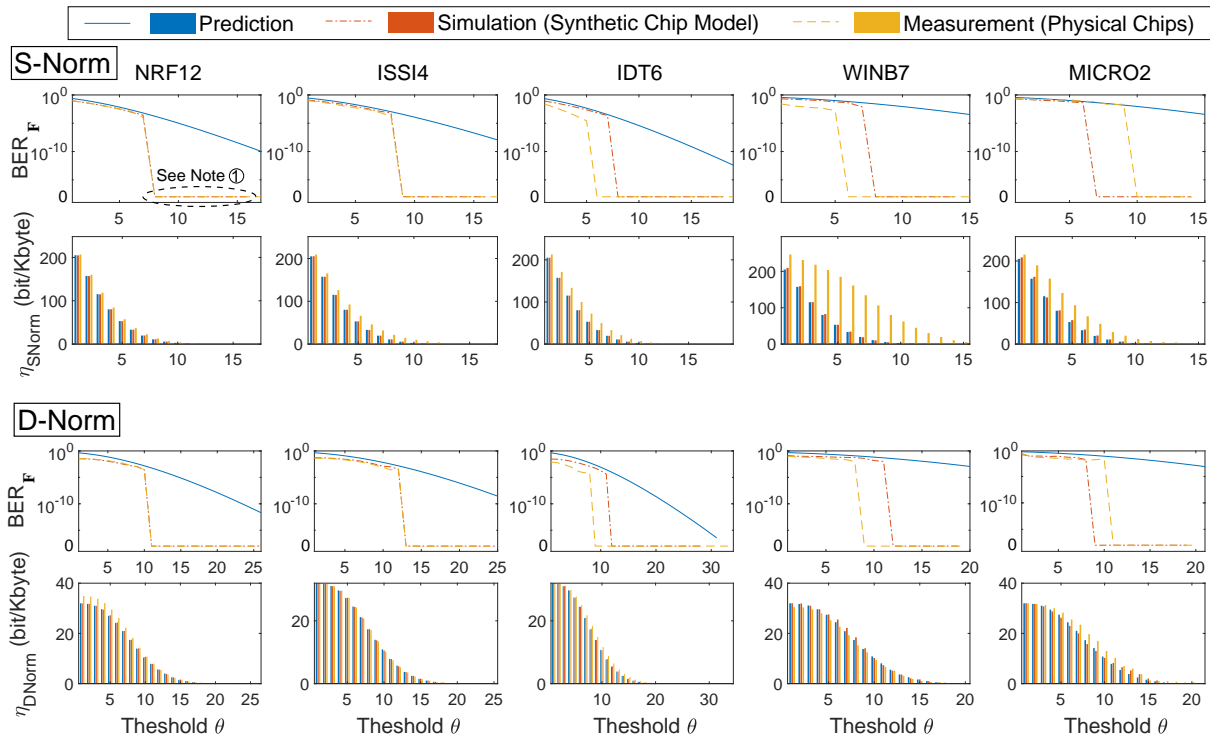


Figure 6.9: S-Norm and D-Norm validation on the datasets including three types of SRAM memories from three different manufacturers (NRF12, ISSI4, and IDT6), one type of Flash memory, and one type of EEPROM memory with S-Norm setting ($n = 63$) and D-Norm settings ($n = 64, m = 4$). Note ①: the number of repeated measurements is finite (see Table 6.2) and inadequate to demonstrate any errors when the expected BER_F is considerably less than $1/(\text{number of repeated measurements})$.

Based on our comprehensive experimental validations on SRAM memories from three different manufacturers, Flash memories, and EEPROM memories, we can now conclude that our formalised model of the unreliability, BER_F , and extraction efficiency, η , in Section 6.3 are indeed reliable measures. Most importantly, the formalised models serve as bounds for BER_F and η in practice; the measurement results and synthetic chip model results are the same or better than those predicted by the analytical models.

6.4.3 Evaluating Uniformity and Uniqueness

In addition to the two crucial performance measures we formulated, reliability and extraction efficiency, we further consider measures that evaluate other qualities of the transformed bits in terms of *uniqueness* and *uniformity* (see [112] for a definition of these measures). In the following, our evaluations are based on the measurements obtained from the 100 physical chips in the *augmented NRF12 dataset*.

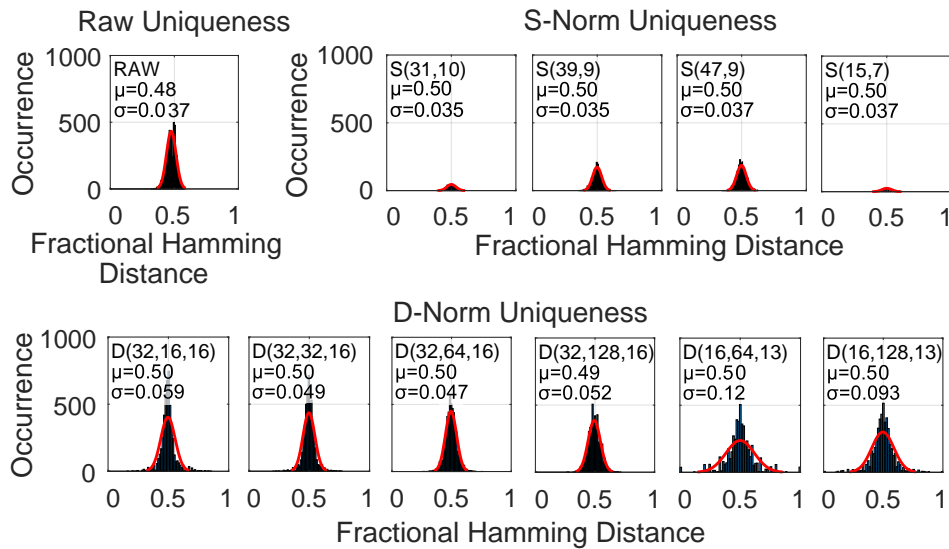


Figure 6.10: Uniqueness evaluation using physical nRF52832 chips. The plots summarise the mean (μ) and standard deviation (σ) across a subset of S-Norm parameters, $S(n, \theta)$, and D-Norm parameters, $D(n, m, \theta)$ applied to our large dataset of 100 chips.

Uniqueness Evaluation. Essentially, uniqueness measures how different the fingerprints are between devices. However, the formal definition of uniqueness based on fractional Hamming distance²⁶ [112] cannot be directly applied for \mathbf{F} fingerprints because the transformed bits are generated from different physical memory blocks from chip to chip, and the number of such bits obtained could also vary from chip to chip. To account for this, we propose evaluating the uniqueness of S-Norm and D-Norm transformed fingerprints based on the following approach:

1. Given a set of transformation parameters (such as n, m , and θ for D-Norm), we extract all the F bits for each of the N (i.e., 100) devices.
2. Given a pair of devices out of $\binom{N}{2}$, we identify the device which produces the \mathbf{F} string with the smaller number of F bits within this pair, and truncate the longer bit string to the same length. Then, we calculate the fractional inter-chip Hamming distance for this pair.
3. We repeat the process in Step 2) for all the $\binom{N}{2}$ pairs to obtain the uniqueness measurement over the 100-chip dataset.

The uniqueness of raw fingerprints, S-Norm fingerprints, and D-Norm fingerprints is illustrated in Figure 6.10. The mean uniqueness of the raw fingerprints is 0.48, with a standard deviation of 0.037. For S-Norm, the mean uniqueness achieves the ideal value of 0.50 under all tested settings, and the largest standard deviation is 0.037 under the setting of ($n = 15$ and $\theta = 7$)

²⁶Fractional Hamming distance (FHD) is a distance measure between two vectors of equal length, defined as the number of positions in the two vectors with different values, normalised by the vector length.

6.5 Deriving Cryptographic Keys for Security Functions

and ($n = 47$ and $\theta = 9$). The mean uniqueness of the D-Norm fingerprints also exhibits the ideal value of 0.50, except for settings ($n = 32, m = 128, \theta = 16$), but the mean uniqueness of 0.49 is still nearly the ideal value. In general, as the number of extracted fingerprints in a tested sample decreases, we also observe an increase in standard deviation; this is expected because of the resulting small sample size for statistical analysis.

Uniformity (Bias) Evaluation. Uniformity measures the balance between zeros and ones in a fingerprint vector. The uniformity distribution of raw fingerprints, S-Norm fingerprints and D-Norm fingerprints is illustrated in Figure 6.11. The uniformity of the raw fingerprint is very close to the ideal value of 0.5, with a very small standard deviation. The uniformity of S-Norm and D-Norm methods—across the various parameter settings—is close to the ideal value, albeit with a slight bias toward ‘1’. Notably, such slight biases are acceptable for key derivation and can be simply compensated by using a few more fingerprint bits when deriving a key [201].

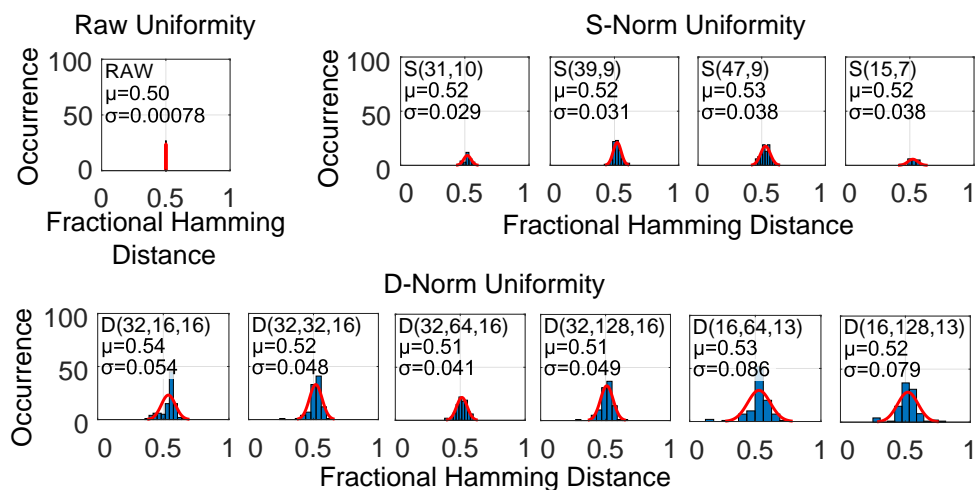


Figure 6.11: Uniformity evaluation using physical nRF52832 chips. The plots show the mean (μ) and standard deviation (σ) for uniformity across a subset of S-Norm parameters, $S(n, \theta)$, and D-Norm parameters, $D(n, m, \theta)$ applied to our large dataset of 100 chips.

6.5 Deriving Cryptographic Keys for Security Functions

We demonstrate the *expressive power of our formalisation* by investigating the derivation of root keys from commodity memory chips facilitated by our analytical models. The dynamic and direct generation of cryptographic keys from memory fingerprint transformations into noise-tolerant bits is a basis for building security functions because: i) memory biometrics

is a true source of randomness; and ii) it removes the need for a protected non-volatile memory—keys can be generated on-demand and ‘*forgotten*’ after usage.

In the following sections, we elaborate on a method for employing the new fingerprint \mathbf{F} obtained from the NoisFre transformation to realise a cryptographic key generator (Section 6.5.1) and evaluate the practical realisation of such a key generator (Section 6.5.2); we defer the security analysis of the key generation process to Section 6.7.3.

6.5.1 A Method for Realising a NoisFre Key Generator

A typical memory fingerprint-based key generation method involves two steps: i) a one-time secure key enrolment on the server-side; and ii) on-demand secure key re-generation on the device-side [38], [56], [72], [124]. Positions of transformed bits F should be provisioned during the key enrolment phase and provided during the key re-generation phase. We refer to these positions using a **mask**. Recall that we have referred to those raw bits that produce a 1-bit F as a block. For the S-Norm, one block has n raw bits, while one block has $n \times m$ raw bits in the D-Norm; for both methods, n raw bits form one ℓ_1 -Norm. In the discussion that follows, we consider key generation under two practical settings:

- Devices with write-once-read-many (WORM) memory for storage of the **mask** defined to select the memory regions to be used in the NoisFre transform prior to deployment.
- Devices without WORM memory where the **mask** has to be transmitted, for example, through a wireless communication channel.

On-Server Secure Key Enrolment

First, we describe the one-time secure key enrolment process, depicted in Figure 6.12. This process

Protocol. The one-off on-server secure key enrolment protocol with NoisFre is as follows:

1. Fingerprint memory is a memory region from which the raw device memory fingerprint \mathbf{f} is extracted.
2. The raw fingerprint \mathbf{f} is processed by the server. The NoisFre Transform Selection process determines a noise-tolerant fingerprint vector \mathbf{F} and the corresponding **mask** based on the parameters n , m , and θ determined by a security practitioner. Notably, a practitioner

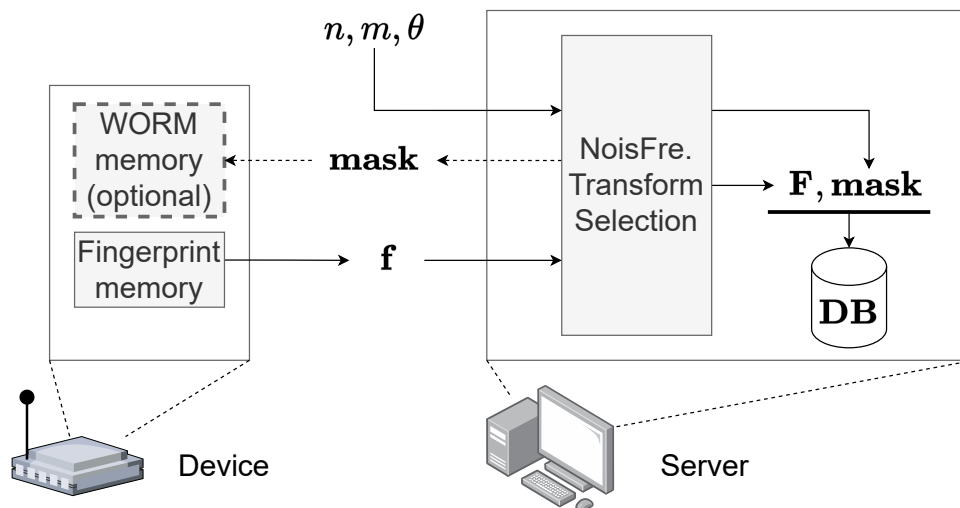


Figure 6.12: NoisFre secure key enrolment process. The raw fingerprints are extracted and used in the NoisFre Transform Selection process (as defined in equations (6.6) and (6.7) for D-Norm) to determine the selected memory addresses (**mask**) for subsequent use in security functions. Although we have only focused on the generation of a single mask, several masks may be defined to allow the server to subsequently generate different secret keys on-demand.

can employ the analytical expressions derived in Section 6.3 to determine the appropriate parameter values.

3. Both **F** and the **mask** are stored in the server's secure database (**DB**, indexed by, for example, the device identification number [**id**], although not explicitly shown here for simplicity).
4. *Optionally*, the **mask** can be stored inside the device's WORM memory.

Dynamic On-Device Secure Key Generation

Now, we consider the realisation of on-device secure key generation with a device memory fingerprint biometric. We illustrate the key generation method in Figure 6.13.

Protocol. The dynamic on-device secure key generation protocol with NoisFre is as follows:

1. If the device implements WORM memory to store the **mask**, as in Figure 6.13 (a), the server fetches device-specific information from the **DB**, such as the enrolled **F**.
2. If the device does not implement WORM memory, the server fetches device-specific information from the **DB**, such as the enrolled **F** and **mask**, as in Figure 6.13 (b). The **mask** is transferred from the server to the device over a (non-secure) wireless communication channel. To ensure the integrity of the **mask**, a message authentication code (MAC) tag is computed by the server as $\mathbf{tag} \leftarrow \text{MAC}_{\mathbf{F}}(\mathbf{mask})$ and appended to the **mask**.

3. The device dynamically generates a new noisy raw fingerprint \mathbf{f}' from the fingerprint memory.
4. The device computes $\mathbf{F} \leftarrow \text{NoisFre.Transform}(\mathbf{f}', \mathbf{mask})$, where $\text{NoisFre.Transform}()$ is a function defined by, for example, the D-Norm transform in equation (6.6).
5. If the device does not implement a WORM memory, then the \mathbf{mask} is sent by the server, as shown in Figure 6.13 (b); the device computes $\mathbf{tag}' \leftarrow \text{MAC}_{\mathbf{F}}(\mathbf{mask})$. To check the integrity of the mask, the \mathbf{tag}' is compared to the \mathbf{tag} supplied by the server. If the two values match, output the NoisFre fingerprint \mathbf{F} ; otherwise, output \perp .
6. Now both the server and the device share the same highly reliable \mathbf{F} to be used as a shared secret in a security function.

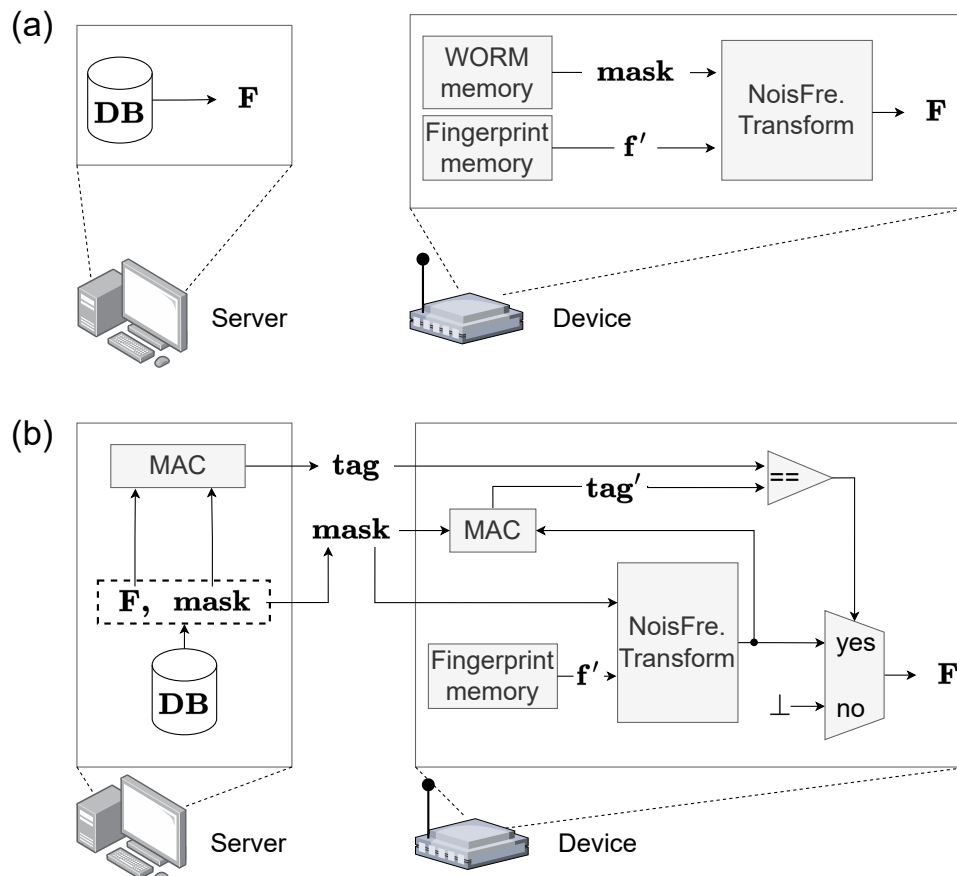


Figure 6.13: On-device NoisFre key generation: (a) the \mathbf{mask} is stored in a device's WORM memory; and (b) the \mathbf{mask} is supplied by the server over the wireless communication channel, if there is no WORM memory available on-device, for example. Although the illustration shows the production of one \mathbf{F} -based key using one \mathbf{mask} , it is possible to enrol and generate several keys if desired.

6.5.2 Evaluations

We begin our systematic evaluation of cryptographic key generation with the following question and employ the formal models and the physical chip measurements for our evaluations.

The Reliability of a k -bit NoisFre Fingerprint \mathbf{F}

Transformed fingerprint \mathbf{F} can be directly utilised as a cryptographic key because they are invariant to a desirably high number of noise-induced bit error patterns—these F bits exhibit a high noise tolerance. The overall failure rate $P_{\mathbf{F}}^{\text{fail}}$ of a k -bit noise-tolerant key \mathbf{F} can be expressed as:

$$P_{\mathbf{F}}^{\text{fail}} = 1 - (1 - \text{BER}_{\mathbf{F}})^k \tag{6.12}$$

Recall that the formalised $\text{BER}_{\mathbf{F}}$ in Section 6.3.1 is conservative. Therefore, the $P_{\mathbf{F}}^{\text{fail}}$ in equation (6.12) will also yield a conservative estimation. We expect a key failure rate in practice to be lower than our prediction here. This hypothesis is validated with an extensive simulation-based on a large simulated chip with up to *one billion* repeated noise-tolerant key bit extraction, as illustrated in Figure 6.14.

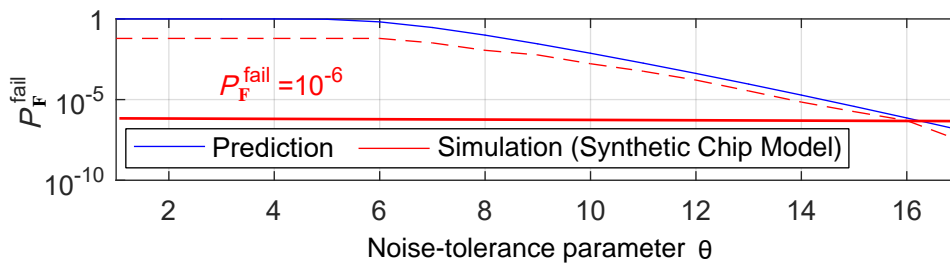


Figure 6.14: Validation of equation (6.12). The simulated chips are based on the worst-case $\text{BER}_{\mathbf{f}} = 6.09\%$ from the NRF12 chip dataset. The parameters selected are $n = 32$, $m = 16$ and varied D-norm parameter θ from 1, ..., 17. We conducted 10 million re-evaluations of a 128-bit noise-tolerant fingerprint for each value of $\theta = 1, \dots, 16$ and one billion evaluations for $\theta = 17$. Our results corroborates equation (6.9) and equation (6.12) as an upper bound on the failure rate of a NoisFre fingerprint employed as a cryptographic key. An even lower $P_{\mathbf{F}}^{\text{fail}}$ is achievable if a larger θ is used. We halted our investigation at $\theta=17$ as it answers the question we investigated.

Next, considering a practitioner’s desire for a $P_{\mathbf{F}}^{\text{fail}} < 10^{-6}$ performance target²⁷ for typical industrial applications, as highlighted in [155] and recent studies [56], [74], [196], [210]–[212], we investigate the following question.

²⁷Notably, there is nothing fundamentally preventing us from aiming for a lower key failure rate. We can see from Figure 6.14 that a larger θ will achieve a lower failure probability.

The Most Efficient Transformation Method

Consider the problem: What is the most efficient transformation method presenting the highest extraction efficiency while ensuring sufficient reliability for \mathbf{F} to be *direct* use as a 128-bit cryptographic key with a failure rate lower than 10^{-6} under *worst-case* raw fingerprint BER_f ?

We employ NRF12 SRAM-based synthetic data to facilitate the massive number of evaluations necessary to address the question. The evaluation process is described below:

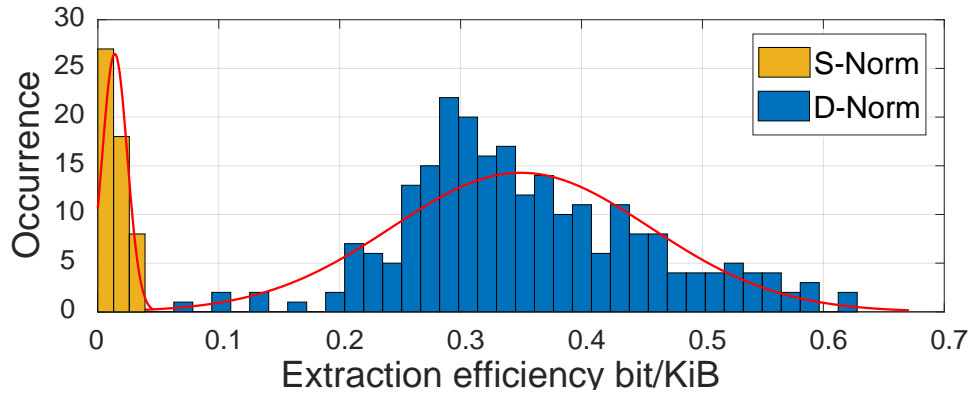


Figure 6.15: Extraction efficiency comparison between D-Norm and S-Norm. For S-Norm and D-Norm extractions, we have evaluated 16,384 and 8,388,480 parameters combinations, respectively. The θ of D-Norm may take any value in $[1, n]$, while in S-Norm, the θ is restricted within $[1, n/2]$. Meanwhile, D-Norm extraction employs the additional parameter, m . Therefore, the possible combinations of parameters for D-Norm are magnitudes larger than that of S-Norm.

1. We determine the $\text{BER}_{\mathbf{F}}$ corresponding to a 128-bit key with a failure rate of 10^{-6} using equation (6.12). The resulting $\text{BER}_{\mathbf{F}}$ is 7.81×10^{-9} .
2. For each $n \in \{1, 2, 3, \dots, 256\}$, evaluate the minimum θ for the required $\text{BER}_{\mathbf{F}}$ (equation (6.8) for S-Norm and equation (6.9) for D-Norm) to ensure $\text{BER}_{\mathbf{F}} < 7.81 \times 10^{-9}$. In these equations, we employ the mean of the worst-case $\text{BER}_f = 6.09\%$ of NRF12 dataset to compute an upper bound for $\text{BER}_{\mathbf{F}}$.
3. For S-Norm, the extraction efficiency η is calculated with equation (6.10) using the n and θ determined in the previous step.
4. D-Norm requires us to further determine the m value that can provide the highest η . As observed in Figure F.3 in the Appendix, η changes smoothly with respect to m . To reduce the search-time overhead, we applied a grid-based search technique: i) evenly select j sample points from the entire domain of $m \in \{1, 256\}$; ii) calculate the η for each $m = 1, 2, 3, \dots, j$; iii) find the m values corresponding to the highest and the second highest η ;

6.5 Deriving Cryptographic Keys for Security Functions

iv) refine the search domain to be between the two points found in step iii); and v) repeat from i) to iv) to locate the m that gives the highest η .

The results from our investigation are depicted in Figure 6.15; here, we plot the occurrences of extraction efficiency as a function of η from all the combinations of S-Norm parameters (n and θ) and D-Norm parameters (n , θ and m). We can conclude that the D-Norm always affords significantly higher extraction efficiencies conditioned on the 128-bit $P_{\mathbf{F}}^{\text{fail}} < 10^{-6}$ constraint. *Therefore, in the following discussion, we focus on the D-Norm.*

Given: i) different sizes of memories embedded within various COTS electronics; and ii) $\text{BER}_{\mathbf{f}}$ characteristics of noisy fingerprints from different memory technologies:

The Lowest Key Failure Rate

Consider the question: What is the *lowest key failure rate* $P_{\mathbf{F}}^{\text{fail}}$ achievable for a 128-bit key \mathbf{F} from *each* memory technology and manufacturer considered in our study?

This scenario resembles a practical application setting where the computing platform or micro-controller unit, for example, needs to be selected based on meeting security and application requirements. We can assume that inherent (worst-case) $\text{BER}_{\mathbf{f}}$ of raw fingerprints are known (i.e., published measurement studies on memory technologies). Thus, we test our suite of memory technologies using the following approach:

1. For each memory dataset listed in Table 6.3, we conduct an exhaustive parameter search using possible combinations of D-Norm parameters ($n, m \in [1, 128]$, and $\theta \in [1, n]$) using our analytical models. This step identifies the (n, m, θ) combination exhibiting the lowest $P_{\mathbf{F}}^{\text{fail}}$ while still providing least 128-bit \mathbf{F} .
2. We employ the formulated equation (6.9) to obtain the $\text{BER}_{\mathbf{F}}$ of the extracted \mathbf{F} using the identified m, n, θ and the mean of $\text{BER}_{\mathbf{f}}$ characterised across the chips in a given memory type dataset.
3. We use $\text{BER}_{\mathbf{F}}$ substituted into equation (6.12) to determine the best $P_{\mathbf{F}}^{\text{fail}}$ of the selected and transformed \mathbf{F} with at least 128 bits.

Results are summarised in Table 6.3. Taking the expected $\text{BER}_{\mathbf{f}}$ across the smallest SRAM dataset, the lowest $P_{\mathbf{F}}^{\text{fail}}$ expected from a chip with SRAM capacity of 64 KiB is in the magnitude of 10^{-5} . Notably, $P_{\mathbf{F}}^{\text{fail}}$ reported in Table 6.3 is conservatively estimated from formulations.

Table 6.3: The lowest key failure rate $P_{\mathbf{F}}^{\text{fail}}$ achievable for obtaining a 128-bit key \mathbf{F} for *each* investigated memory dataset using D-Norm. worst-case $\text{BER}_{\mathbf{f}}$ is the mean of the value calculated across the chips in a given dataset. Notably, as described in Section 6.5.2 and illustrated in Figure 6.14, equation (6.12) provides a conservative upper bound, the actual key failure rates will be much lower in practice.

Dataset (Type)	worst-case $\text{BER}_{\mathbf{f}}$ (mean)	Memory size	n	m	θ	$P_{\mathbf{F}}^{\text{fail}}$ equation (6.12)
NRF12 (SRAM)	6.09%	64 KiB	29	65	13	4.04×10^{-5}
ISSI4 (SRAM)	8.29%	256 KiB	50	128	19	3.56×10^{-5}
IDT6 (SRAM)	5.42%	512 KiB	83	128	25	5.29×10^{-9}
WINB7 (Flash)	16.26%	256 MiB ¹	120	128	41	2.52×10^{-4}
MICRO2 (EEPROM)	16.37%	32 KiB ¹	14	61	9	4.01×10^{-1}

¹Recall that the tested size of WINB7 and MICRO2 datasets are 69 KiB and 2 KiB. When calculating the number of selected noise-tolerant bits, the memory sizes are scaled up by assuming the entire 256 MiB (WINB7) and 32 KiB (MICRO2) memory spaces are available for fingerprinting.

In practice, $P_{\mathbf{F}}^{\text{fail}}$ is expected to be much better. Importantly, with more abundant and freely available on-chip SRAM, represented in the IDT6 dataset, a remarkably low key failure rate of 5.29×10^{-9} is achievable.

As expected, the higher worst-case $\text{BER}_{\mathbf{f}}$ of the MICRO2 and WINB7 datasets implies that the techniques in NoisFre are not able to select a 128-bit \mathbf{F} with a satisfactory $P_{\mathbf{F}}^{\text{fail}}$. However, the WINB7 tested benefits from a high memory capacity (256 MiB compared to 32 KiB for MICRO2) and we can achieve orders-of-magnitude better $P_{\mathbf{F}}^{\text{fail}}$ than MICRO2.

In summary, for SRAM—the most prevalent memory type in IoT devices—a 128-bit key with a key failure rate less than 10^{-6} can be efficiently obtained given an adequate SRAM memory capacity. However, for memory types exhibiting severely high $\text{BER}_{\mathbf{f}}$, for example, MICRO2 and WINB7, the method itself is insufficient to gain a satisfactory $P_{\mathbf{F}}^{\text{fail}}$. Although, NoisFre does significantly reduce the key failure rate given the higher capacity of WINB7 for selecting bits. Notably, with such high $\text{BER}_{\mathbf{f}}$ memory characteristics, even the state-of-the-art, efficient method of RFE-based key generators are unlikely to deliver a computationally tractable solution on resource-limited devices. We discuss this limitation further in Section 6.9.

6.6 Security Function Implementation for Comparison

Here, we describe a case study implementing a NoisFre-based key generator followed by performance and implementation overhead comparisons against the lightweight, state-of-the-art (R)FE-based method.

6.6 Security Function Implementation for Comparison

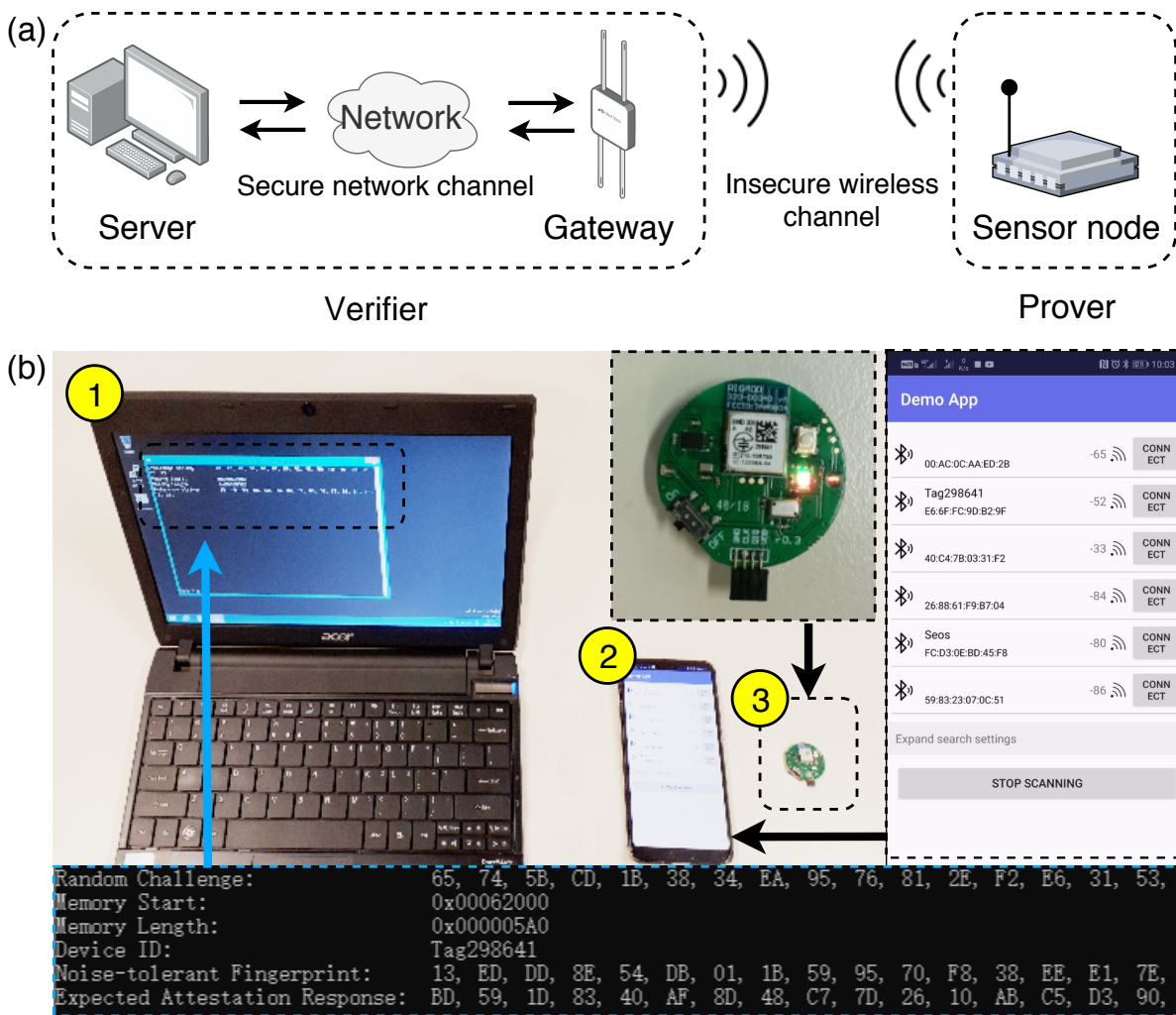


Figure 6.16: NoisFre (a) System overview; and (b) Experiment setup: Verifier \mathcal{V} consists of: ① a laptop as the cloud server; ② a smartphone as the gateway; device is ③ a commercial widely used nRF52832 Bluetooth-LE sensor. Watch the demo video for more details:

<https://www.youtube.com/watch?v=O5NWZw-swpw>



scan to watch

6.6.1 An Overview

The entities, a Verifier \mathcal{V} and a Prover \mathcal{P} , involved in this case study are illustrated in Figure 6.16 (a). The Verifier \mathcal{V} consists of a server and a wireless network gateway (smartphone). The Prover \mathcal{P} refers to a wireless sensor node (Bluetooth sensor). In this setup, the server functions as a coordinator, holds the enrolled Prover \mathcal{P} 's information in the database, and issues commands to instruct the Prover \mathcal{P} to perform remote attestation. The gateway bridges the communication between the server and the Prover \mathcal{P} . The traffic between the server and the gateway is assumed to be secure by applying standard security protection mechanisms. The

Prover \mathcal{P} , communicating wirelessly, is deployed in an (insecure) environment. Details of the corresponding attestation protocol are provided in Figure 6.17. Our case study aims to:

- Implement a lightweight remote attestation routine suitable for a Prover \mathcal{P} with a constrained resource by following [30].
- Experimentally demonstrate that *SRAM fingerprints can be accessed on-demand* and at *run-time* by exploiting the low-cost micro controller unit (MCU)'s memory power control features—SRAM regions are arranged in blocks can be individually powered *on* or *off*.

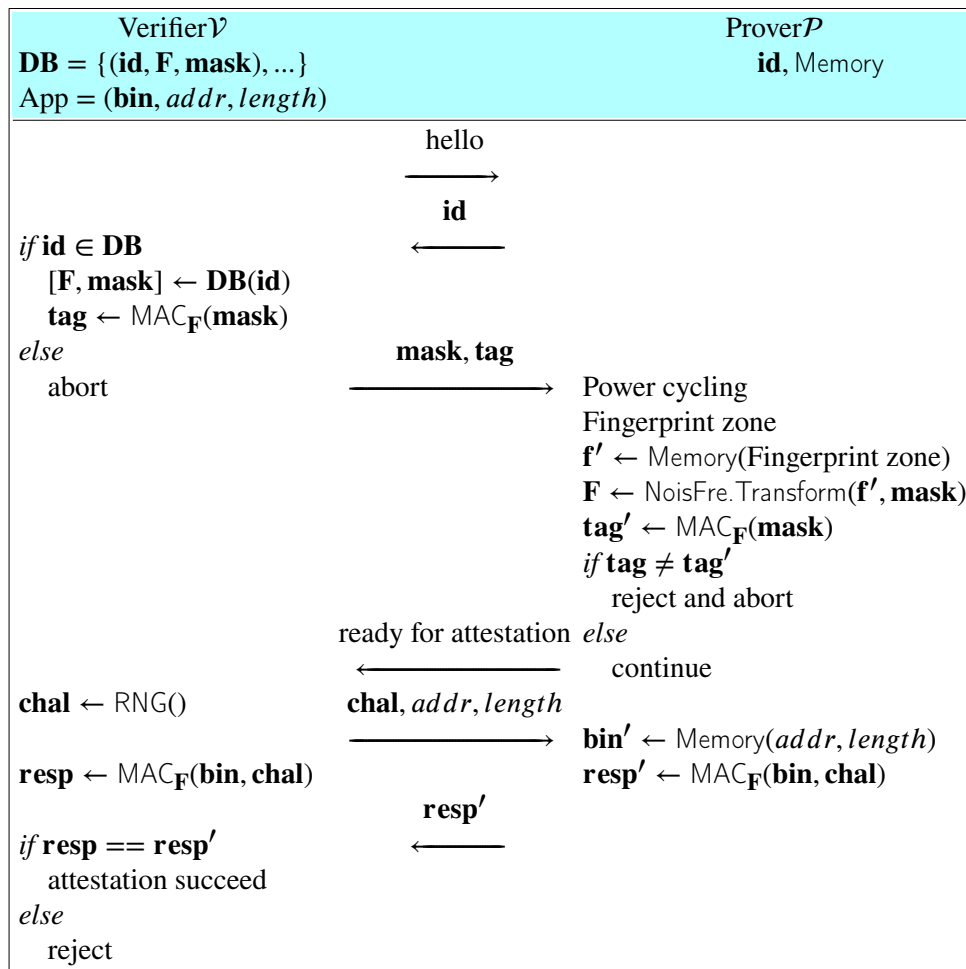


Figure 6.17: Remote attestation protocol, with mask transmitted from the Verifier \mathcal{V} , In case the WORM is not supported on the Prover \mathcal{P} .

Remote Attestation Mechanism. An overview of the remote attestation mechanism based on a NoisFre key generator is illustrated in: i) Figure 6.17, where we assume the Prover \mathcal{P} has no WORM memory available for storing a **mask** and that it has to be transmitted over the wireless communication channel (the worst-case setting in terms of implementation overhead);

6.6 Security Function Implementation for Comparison

and ii) Figure 6.18, where we assume the Prover \mathcal{P} has available WORM memory. We assume the Prover \mathcal{P} has already undergone the enrolment phase we described in Section 6.5.1. The enrolment is conducted by the Verifier \mathcal{V} in the current setting.

A remote attestation can be requested anytime. First, the Verifier \mathcal{V} scans for visible Provers \mathcal{P} by sending a ‘hello’ message. Once there is a Prover \mathcal{P} in the horizon responding with its unique identifier **id**, the Verifier \mathcal{V} fetches the Prover \mathcal{P} ’s information (e.g., **F** and **mask**) from the secure database **DB** by using the **id** as an index. Second, if the received **id** matches one of that stored in the Verifier \mathcal{V} ’s **DB**, the Verifier \mathcal{V} instructs the Prover \mathcal{P} to perform attestation—by sending the **mask** and MAC **tag** for Provers \mathcal{P} with no WORM memory, as in Figure 6.17. In this context, the Prover \mathcal{P} performs a power cycling of memory banks *solely* corresponding to the fingerprint zone²⁸ and dynamically generates \mathbf{F}_i following the steps described in Section 6.5.1. After confirming a ready acknowledgement from the Prover \mathcal{P} , the Verifier \mathcal{V} randomly generates a challenge (a nonce) **chal**, and sends it to the Prover \mathcal{P} along with the address *addr* and the *length* of the target application program (App) code **bin** in the Prover \mathcal{P} ’s memory. The Prover \mathcal{P} ’s response **resp**’ is generated using MAC computed with the noise-tolerant fingerprint **F**. The Verifier \mathcal{V} compares the received response **resp** with a locally calculated reference response **resp**’. The remote attestation is accepted if **resp** and **resp**’ match and rejected otherwise.

If the Prover \mathcal{P} implements WORM memory for storing a **mask**, the protocol can be simplified as shown in Figure 6.18; in our end-to-end demo implementation, we consider this simpler case, and describe the implementation details in Figure F.5 in Appendix. F.2.

6.6.2 Overhead Comparisons

Implementation Details. We provide an overview of the system implemented in Figure 6.16 (b) and defer details to Appendix F.2. Further, we refer the reader to our open-source code release²⁹ for detailed descriptions of our implementation, including dynamic and run-time key generation from SRAM fingerprints. A video demonstration of the end-to-end implementation is available at:

<https://www.youtube.com/watch?v=O5NWZw-swpw>



scan to watch

We implemented a D-Norm-based key generator on an nRF52832 chip with the smallest on-chip SRAM capacity and BER_f of 4.93% tested under -15 to 80 °C operating range. We used $n=32$,

²⁸Each memory bank can be individually powered off by exploiting particular power control registers, thus enabling run-time SRAM fingerprinting.

²⁹See <https://github.com/AdelaideAuto-IDLab/NoisFre>

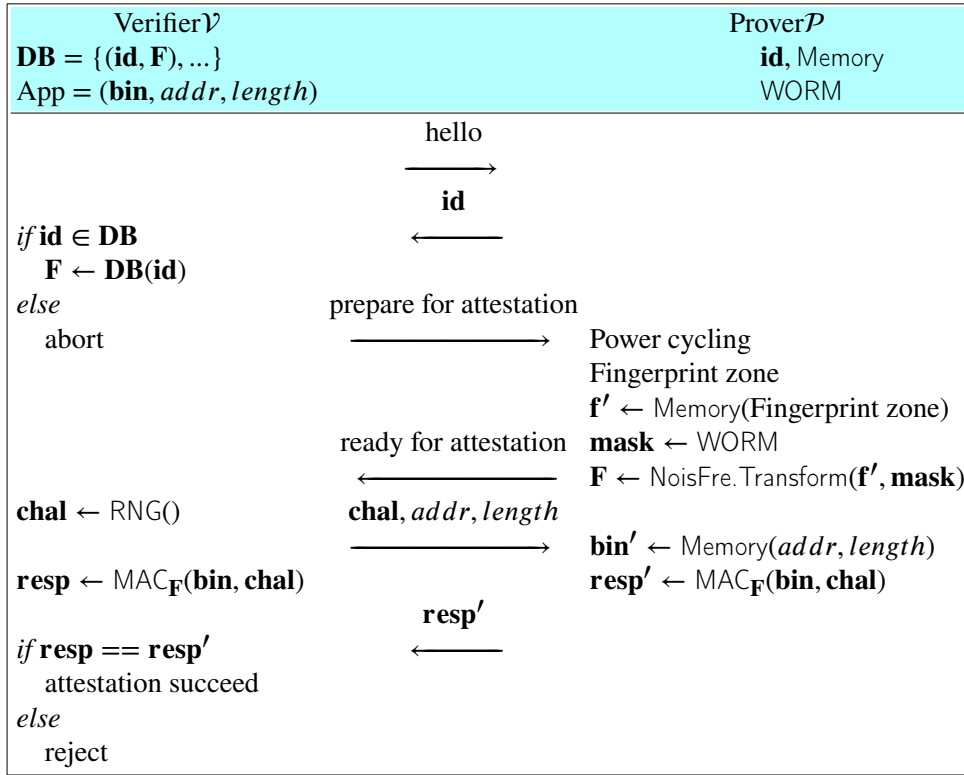


Figure 6.18: Remote attestation protocol, with mask stored in Prover \mathcal{P} 's WORM. In the demo, we implement this version.

$m=48$, $\theta=13$ for D-Norm parameters determined by equation (6.9), (6.11), and (6.12) to be able to extract a 128-bit NoisFre key capable of a key failure rate below 9.15×10^{-6} .

For comparisons, we implemented the (R)FE-based key generators summarised in Table 6.4 to achieve a key failure rate to closely match 10^{-6} . As discussed in Section 2.3.3, in an FE, the device executes the computationally-heavy decoding function, while in an RFE, the device executes the more lightweight encoding function. In our end-to-end demonstration, to achieve a comparable failure rate to that of the D-Norm-based NoisFre key generator, the (R)FE implementation needs 13 parallel blocks of ($n = 63$, $k = 10$, $t = 13$) BCH code³⁰ to provide a similar key failure rate.

Implementation Overhead. The implementation overhead evaluates the usage of two system resources: clock cycles (for code executions) and random-access memory (for run-time data). Overall, in terms of obtaining a 128-bit reliable key with a key failure rate of 9.15×10^{-6} , the implementation of the D-Norm-based NoisFre method with parameters ($n=32$, $m=48$, $\theta=13$)

³⁰BCH code is a class of cyclic error-correcting codes, named after its inventors Bose, Chaudhuri, and Hocquenghem, constructed using polynomials over Galois field

6.6 Security Function Implementation for Comparison

takes 51,044 clock cycles³¹. If the **mask** is provided by the server and transmitted over a wireless channel, an additional 45,622 clock cycles are required for mask integrity checks.

In contrast, the FE-based and the lightweight state-of-the-art RFE-based method introduces significantly higher overheads to achieve a 128-bit reliable key with a slightly inferior key failure rate of 2.45×10^{-5} . Specifically, as evaluated and shown in Figure 6.19, the on-device FE decoding and RFE encoding functions consume 285,311 and 109,850 clock cycles, respectively. Both methods need an additional 60,755 clock cycles for helper data integrity checks. In comparison with the state-of-the-art FE and RFE, for meeting a comparable key failure rate, NoisFre reduces clock overhead by 72% and 43%, respectively, if the mask or helper data is transmitted over the wireless channel requiring helper data integrity checks. However, if the mask or helper data for all of the method are stored on a device's WORM memory, clock cycles required in comparison to NoisFre reduces by 82% (compared to FE) and 54% (compared to RFE).

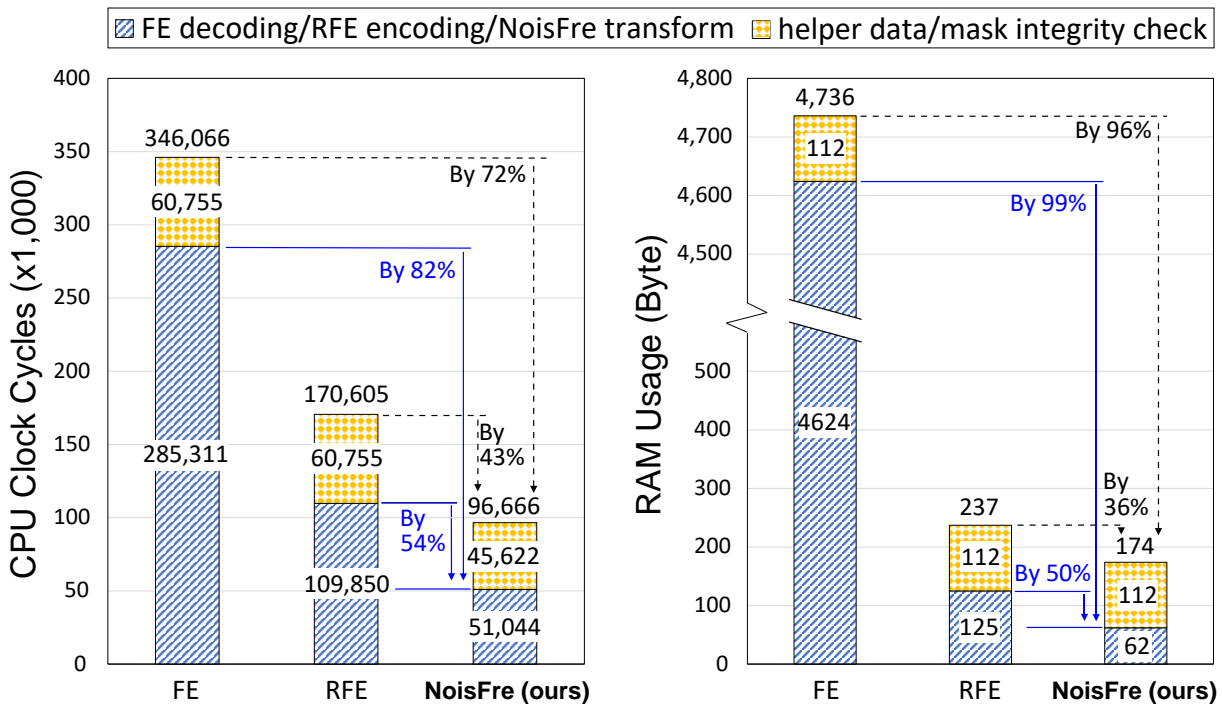


Figure 6.19: Comparison of implementation overhead of the proposed NoisFre key derivation against traditional (R)FE-based key derivation. The integrity checks are necessary if the helper data or the mask is transmitted over a wireless channel.

It is worth emphasising that we have compared NoisFre with an RFE capable of deriving a key with a failure rate of $P_F^{\text{fail}} \approx 10^{-6}$. However, as we show in Table 6.3, if an chip from the

³¹This was tested with nRF52832 SoC, via J-link EDU V10.1 debugger, with nRF5 SDK Ver. 15.3.0, Keil uvision 5.25.2.0 and ARM CC compiler Ver. 5.06 Update 6. Optimisation setting = -O3.

Table 6.4: Implementation overhead of R(FE) employing BCH codes.

Fingerprint source (BER _F)	BCH(n,k,t)	Block number	Key failure rate	Key size	Helper data size	Clock cycles		
						Fuzzy Extractor decoding	Reverse Fuzzy Extractor encoding	Helper data integrity check
NRF12 (4.93%)	(63,10,13)	13	2.45×10^{-5}	130	689	285,311	109,850	60,755
IDT6 (5.42%)	(127,15,27)	9	1.69×10^{-9}	135	1008	967,599	188,487	84,013

IDT6 dataset is used in the implementation, we can obtain a key with a significantly lower key failure rate by exploiting the *free*, abundant memory; now, $P_{\mathbf{F}}^{\text{fail}}$ can be $\approx 5 \times 10^{-9}$. Attempting to achieve such a small $P_{\mathbf{F}}^{\text{fail}}$ using an (R)FE will lead to significantly higher overheads. The (R)FE-based key provisioning method introduces increasing execution overheads if a lower key failure rate is desired as illustrated in Table 6.4. For example, if an IDT6 SRAM chip is used as the fingerprinting barometric source instead of the nRF52832 chips' internal SRAM, a 128-bit key with failure rate of 1.69×10^{-9} requires 188,487 (3.69 times larger) clock cycles with the REF-based method or 967,599 (18.95 times larger) clock cycles with the FE-based method, compared with 51,044 clock cycles for our NoisFre-based method. *Hence, in contrast to (R)FE methods, the on-device computational overhead of the proposed NoisFre key generator remains constant, regardless of the desired key reliability and only depends on the size of the key to be derived.*

6.7 Discussion

6.7.1 Generality of NoisFre

Although our work focused predominantly on SRAM, considering its *ubiquity in low-end IoT devices and the simplistic nature of fingerprint extraction*, the NoisFre fingerprinting methods presented are applicable for other memories, including Flash (WINB7) and EEPROM (MICRO2) memories validated in our study. In principle, it can be applied to other hardware fingerprinting methods [213], [214], given an abundant raw digital fingerprint bit space.

6.7.2 Provisioning Fingerprints at Run-time

Flash and EEPROM memory fingerprints can be accessed during run-time. However, for SRAM fingerprinting, the most common method is to utilise its initialisation pattern at power-up as a fingerprint, although there are other means [115]; for example using data retention voltage [115] or intentionally putting SRAM cells under a meta-stable state. Those methods usually require

customised peripheral circuitry, which tends to be unavailable in COTS devices. Thus, SRAM fingerprinting generally requires power cycling to read the start-up values. As a matter of fact, some low-end microcontrollers allow direct control over the powering of individual SRAM banks [122] (e.g., the low-end nRF52832 studied in this chapter). Consequently, by leveraging such a feature, SRAM fingerprint-based root keys can be requested during *run-time*.

6.7.3 Security Analysis

We have looked at the problem of achieving a pragmatic, on-device key derivation method using noisy memory fingerprints. NoisFre fundamentally obviates the need for computationally intensive on-device Error Correction Code (ECC) logic for the task. It is thus immune to HDM attacks [73], [74] that strategically tamper the helper data *associated with the ECC* to weaken or compromise the key extracted using the state-of-the-art (R)FE methods. The vulnerability is induced by the usage of *ECCs* (were introduced in Chapter. 2). Various ECCs are examined and shown to be vulnerable to HDM attacks [74]. A generic countermeasure against HDM attacks appears to be an open challenge. The NoisFre scheme has sought to remove the necessity for helper data associated with key generation in an RFE and, thus, avoid the HDM attacks that exploit helper data. In the following, we consider the security of our proposed key derivation method in the context of prior methods based on the state-of-the-art (R)FE methods.

Threat Model

Memory fingerprint-based key provisioning studies rarely explicitly define a threat model [196], [201], [215] and operate under the assumption that the key material (i.e., memory fingerprint) cannot be directly accessed. However, studies focusing on incorporating key derivation methods to provide a security function, such as authentication or remote attestation [62], [142], [177], [216], follow a threat model. Therefore, we follow the threat model reasoned therein, along with the assumption that the key material cannot be directly accessed.

Specifically, we consider that an adversary cannot access the raw fingerprint and temporary data stored in RAM or internal chip registers during key derivation. The attacker can tamper with public information used to assist the key derivation. Notably, in prior work, such information would be the ECC associated helper data in a (R)FE-based reliable key derivation method [74]—in our key derivation approach, we assume the *mask* is public information. The mask is sent to a device over a communication channel together with a method for assessing the integrity of the mask or is stored in WORM.

Mask Manipulation Attack

In use cases where the mask is sent to a device over a communication channel, it is possible for an attacker to manipulate the mask. Therefore, we consider *mask manipulation attacks*.

In the context of a NoisFre-based key generator, a MAC **tag** is produced over the mask using the derived **F** to ensure the integrity of the mask—more specifically, $\mathbf{tag} \leftarrow \text{MAC}_{\mathbf{F}}(\mathbf{mask})$, with **F** being the reliable secret key, as illustrated in Figure 6.13 (b). The **mask** and MAC **tag** can be publicly stored off-chip and/or stored on-chip. Subsequently, the MAC **tag** can be re-generated to validate the integrity of a mask stored on-device or transmitted to the device prior to the use of the key derived on-demand, as illustrated in Figure 6.13 (b). Now, the probability of making a modification without being detected is $\frac{1}{2^k}$ with k the length of the derived key. It will be $\frac{1}{2^{128}}$ for a typical 128-bit key.

Although we adopted a simple mechanism in this study to ensure mask integrity, other mechanisms have been proposed to ensure the integrity of helper data in the context of state-of-the-art (R)FE methods [73]. Thus, we can also employ these existing methods to ensure the integrity of the mask for NoisFre key derivation method.

Brute-force Attack

For completeness, we also assess the attack complexity of a brute-force attack on a NoisFre-based key derivation method. The attacker may utilise a brute-force attack to determine the derived key. However, this is extremely challenging when the key is appropriately sized. For a brute-force attack, the probability of finding the correct derived key is $\frac{1}{2^k}$, which is computationally infeasible given a reliable key with a typical length of $k = 128$ bits.

Ageing Attack

The data stored in a SRAM cell can gradually affect its start-up state. This is called data-dependent ageing [68]. Given that the key derivation is based on a physical primitive, we also consider *ageing attacks* that may attempt to exploit the small changes in behaviour of memory cells that occur as a result of ageing the underlying electronic components.

In use cases where a write access protected (e.g., using a memory protection unit [MPU]) memory cannot be allocated for generating fingerprints and where the memory space used for fingerprints must be shared with user application code, an attacker may utilise malicious code on the device to continuously write specific memory patterns to the SRAM used for device

6.8 Related Work

fingerprinting. Such an attempt can accelerate ageing and can potentially degrade the reliability of a NoisFre key generation method.

In use cases where a dedicated memory cannot be allocated for generating fingerprints, several simple mitigation strategies already exist. First, the ageing effect is data dependent. The user can employ an anti-ageing method, such as writing reverse data patterns to mitigate the ageing effect validated as an efficient approach to counter ageing [68]. Second, the SRAM unreliability induced by ageing, even over six years, is small—only 2% [68]. Hence, a simple anti-ageing method for NoisFre is to allow the server to intentionally assume a higher worst-case BER_f during the enrolment phase to count for or tolerate the ageing effect by trading off a slight increase in SRAM volume required to retain the same NoisFre key reliability. If the available memory volume is constrained, a further low-cost anti-ageing measure is for the server to adopt the trial-and-error method reported in [217] to recover the least reliable transformed F bits, because the server can ascertain the bit-specific reliability of each F bit. Notably, in this approach, all the computation overhead is offloaded to the server without imparting any overhead to the device.

6.8 Related Work

Besides memories, various on-board sensors, such as cameras, accelerometers, gyroscopes, magnetometers, and other components, such as CPU magnetic radiations, are utilised to provide fingerprints [218]–[226]. Other recent works also explore commodity scanners to fingerprint 3D objects to track them [227] and exploit the package variations as fingerprints for anti-counterfeiting [228]. However, to obtain hardware fingerprints, those fingerprint extractions are relatively complicated in comparison with memory, especially SRAM, enabled fingerprints.

Notably, hardware fingerprinting is closely related to the notion of PUFs [55]–[57], [229], [230]. Commodity memory fingerprinting, such as SRAM PUF and Flash PUF, is not new. However, mounting them on low-end IoT devices to derive a usable key for security functions relies on post-error correction to reconcile bits errors, which is cumbersome in terms of both overhead and security in practice. Our simple yet efficient NoisFre memory-fingerprinting approach addresses this gap.

We exploit the idea of a differential measurement in the formulation of the D-Norm method based on the base distance (ℓ_1 -Norm) to improve the *extraction efficiency* (number of F bits with a desired noise tolerance) from a given memory. Interestingly, in PUF studies, formulating methods to exploit a differential gap or comparison has been utilised by *extrinsic*

PUFs—implemented with additional hardware—such as ring oscillator PUF (RO-PUF) [59], [229]–[231] and arbiter PUF (APUF) [232] to obtain responses with improved reliability. The concept has subsequently been applied in [229], [230] to enhance reliability and address ageing of electronic components in RO-PUFs facilitated by the ease with which RO frequency differences are already measured and can be directly used. In our *intrinsic memory* studies, we exploit a base distance, ℓ_1 -Norm, to generate a differential measurement to build the D-Norm transform for memory PUFs *intrinsic* to COTS devices. As discussed in Section 6.2.3, we recognised that the differential formulation can yield significantly more reliable bits compared to S-Norm employing (simply, the ℓ_1 -Norm base distance). Thus, in this chapter, we combine these two mathematical concepts (base distance with a differential measure) together to extract more noise-tolerant bits from a memory PUF—a method that can be used with intrinsic memories widely exist in COTS devices.

6.9 Chapter Summary

By exploiting ubiquitously embedded memory within commodity computing devices, the proposed NoisFre approach constructively extracts transformed memory fingerprints that were embodied with a high tolerance to noise affecting the generation of fingerprints. With a simple, single, one-off fingerprint enrolment measurement, NoisFre is able to judiciously identify highly reliable transformed fingerprints serving as a hardware root key or root of trust to directly support various security functions for a wide range of COTS electronic devices. Besides formalisation of two specific S-Norm and D-Norm fingerprint transformation methods and extensive empirical validations on SRAM, Flash, and EEPROM memories using 119 physical chips in total, we have conducted a case study with an end-to-end implementation of a remote attestation security service employing NoisFre fingerprints to significantly reduce the overhead in comparison with the state-of-the-art RFE method for constructing reliable fingerprints for a key generator. We also demonstrate how SRAM fingerprints can be generated at run-time by utilising individual memory-bank power control features on MCUs. Overall, NoisFre is a *simple but practical* method, especially for existing low-end commodity electronic devices.

The study in this chapter is not without limitations. As shown in Figure 6.15, the highest extraction efficiency (i.e., the number of fingerprint bits with a $\text{BER}_F < 7.81 \times 10^{-9}$ that can be extracted from a unit-sized memory block) that NoisFre can achieve is 0.62 bits per KiB. Hence, extracting a usable (e.g., 128-bit) secure key from a highly resource-constrained device with a mere 2 KiB memory space (i.e., the SRAM size of the passively powered computational radio frequency identification (CRFID) device studied in Chapter 3 and Chapter 4) with NoisFre is

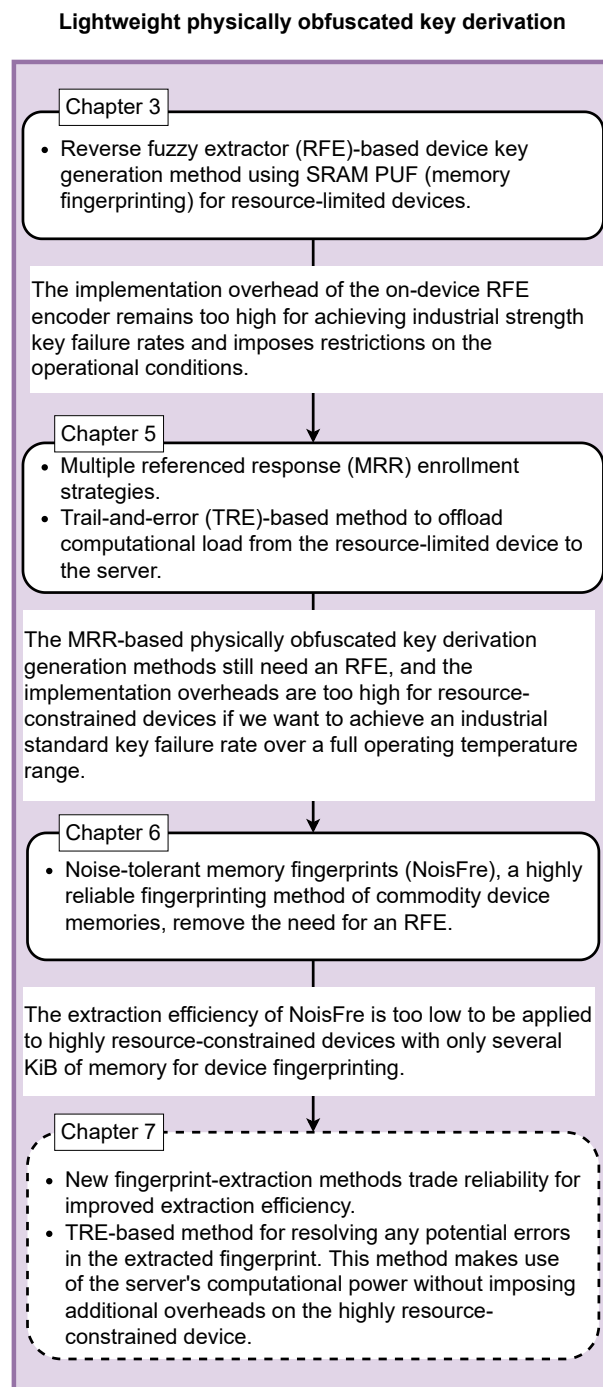
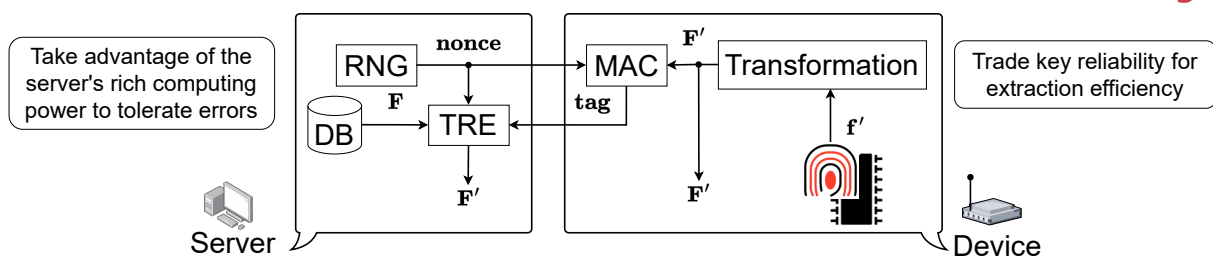


Figure 6.20: Upcoming chapter sneak peek.

still not immediately possible. In the upcoming Chapter 7, we will look at approaches to extract more noise tolerant bits from a given memory. We contrast the proposed NoisFre method in this chapter against the corresponding problem to be explored in Chapter 7 in Figure 6.20.

Chapter 7

Noise-Tolerant Key Generators for Highly Limited On-device Memory



The NoisFre concept proposed in Chapter 6 enables *simple and efficient yet practical* construction of key generators from noisy COTS device memory fingerprints. However, extraction efficiency (the number of noise-tolerant bits extracted compared to memory size) can be a limiting factor for highly resource-constrained devices with limited on-chip memory capacity. This chapter introduces NoisFre-Lite to address the considerable challenge of deriving keys from SRAM memory in contexts where *resources are highly constrained, in terms of memory size available for device fingerprinting*. NoisFre-Lite builds on NoisFre by similarly transforming raw SRAM fingerprints into noise-tolerant fingerprints but also developing two new algorithms for the D-Norm transform-based noise-tolerant bit selection around a compromise between reliability and extraction efficiency. Employing these algorithms means few noise-tolerant bits can remain below the level of reliability needed to produce a highly reliable key generator. But, the proposed NoisFre-Lite utilises a server's rich resources to withstand transformed bit errors via a trial-and-error (TRE) approach that ultimately establishes a highly reliable key. We evaluate NoisFre-Lite in the context of the battery-less CRFID device studied in Chapter 3, which represents a highly resource-constrained device. NoisFre-Lite easily derives a 128-bit key with a key failure rate below 10^{-6} from a 2 KiB SRAM memory. NoisFre-Lite induces an on-device computational overhead as low as 20,238 clock cycles, a 92.8% reduction compared to the state-of-the-art reverse-fuzzy-extractor-based generation method, which requires 280,063 clock cycles.

7.1 Motivation and Contribution

Although physically obfuscated key derivation from device memory fingerprints is a highly secure and cost-efficient key storage solution, mounting such techniques on resource-constrained devices is challenging due to the implementation overhead of key generators constructed from such memory fingerprints.

The NoisFre scheme proposed in Chapter 6 fundamentally obviates the need for the computationally-intensive on-device reverse fuzzy extractor encoder with a method that maps the raw, noisy fingerprint space into the noisy-tolerant fingerprint space to enable extraction of transformed noisy-tolerant fingerprints. A crucial benefit of NoisFre is that it provides a noise-tolerance measure for each transformed noise-tolerant bit to judiciously select highly reliable bits to meet the stringent reliability requirement of a root key. This means that on-device key generation no longer relies on an error correction operation.

However, the transformation and selection process sacrifices the entropy of the raw, noisy fingerprint space: a large number of raw fingerprint bits can only be transformed into a single F bit, and selecting noise-tolerant bits with a high degree of noise-tolerance further discards raw fingerprint bits. Consequently, although NoisFre benefits from large memory sizes in many COTS devices, the methods explored in Chapter 6 remain difficult to implement on highly resource-constrained devices where, having tens of thousands of bytes of memory, is increasingly becoming common [233].

Hence this chapter investigates the following problem:

- How can we generate reliable root keys from pervasive SRAM memory fingerprints under *stringent restrictions* on: i) *computational capability*; and ii) *memory size*?

The following summarises this chapter's contributions to addressing this problem and the main results from the experiments conducted:

- **Noise-tolerant key generation for highly resource-limited devices.** We propose NoisFre-Lite, which can provision root keys on highly resource-constrained devices by re-purposing memory in contexts of extremely limited computational and memory resources. The approach devises *two complementary algorithms that transform and select raw, noisy fingerprint bits into noise-tolerant bits* using the D-Norm method, improving extraction efficiency (extracted highly reliable noise-tolerant bits compared to raw memory fingerprint bits). NoisFre-Lite is built around its ability to trade key reliability

for improved extraction efficiency. Subsequently, *to optimise the key failure rate, we devise the TRE algorithm*, which restores the errors in the transformed bits by utilising the abundant server-side computational power and the prior information concerning the expected worst-case noise-tolerance degree from the transformation and selection method. Importantly, we effectively use the server’s computational power without adding any computational overhead to the highly resource-constrained device.

- **Analytical model formulations.** We formulate analytical models for each of the proposed transformation and selection algorithms to describe their performance. These models will help practitioners accurately estimate key size and key failure rate, given a specific device’s memory capacity. We also formulate analytic models to predict the number of error bits for a given device fingerprint reliability, and an upper-bound of on-server computing load for running the TRE algorithm.
- **Extensive evaluations.** We extensively evaluate NoisFre-Lite in terms of key failure rate, extraction efficiency, uniformity (bias), uniqueness, bit-aliasing of the derived key and on-server TRE complexity. The experiments affirm the efficacy of our method and the validity of our formulations.
- **Practical use case and comparisons.** We implement and test a NoisFre-Lite key generator on a battery-less CRFID device with a constrained 2 KiB SRAM memory—a highly resource-limited device. NoisFre-Lite is demonstrated to be capable of obtaining a 128-bit key with a failure rate below 10^{-6} using only 20,238 clock cycles—this represents a 92.8% reduction in comparison to the state-of-the-art RFE-based method which requires 280,063 clock cycles.

7.1.1 Chapter Overview

The rest of this chapter is organised as follows. First, Section 7.2 provides an overview of the proposed NoisFre-Lite concept. Section 7.3 details the proposed algorithms for improving extraction efficiency. Next, Section 7.4 describes and applies the on-server TRE technique, which restores erroneous bits in the derived device fingerprint key over re-generations, if any exist. Section 7.5 formally derives analytical expressions for key failure rate, extraction efficiency and the time complexity of the TRE method. We evaluate the proposed NoisFre-Lite techniques using synthetic chip model-based simulations and physical chip measurements in Section 7.6. Section 7.7 demonstrates a use case of the proposed

7.1 Motivation and Contribution

Table 7.1: Table of notations used in this chapter.

S	A server S is a single entity comprising a networked host computer and a wireless gateway (such as a RFID reader).
D	A device D is a highly resource-constrained device.
\mathcal{A}	An adversary \mathcal{A} is an attacker. For adversary \mathcal{A} 's goals and ability, please refer to Section 7.7.3.
DB	Server's database, where each element is a three-tuple describing each device D in terms of: i) its unique and immutable identification number id ; ii) its enrolled noise-tolerant fingerprint F ; and iii) its noise-tolerance parameter θ .
f	A noisy raw fingerprint vector.
F	A transformed noise-tolerant fingerprint vector.
mask	Positions of transformed bits F should be provisioned during the key enrolment phase and provided during the key generation phase. We refer to these positions using a mask .
nonce	The nonce is a randomly generated number that should be used only once in the secure communication context to prevent replay attacks.
tag	A MAC tag computed over the nonce (for more details, please refer to Section 7.7.1).
e	A vector of error bits in the re-generated raw noisy fingerprint \mathbf{f}' .
ϵ	A vector of error bits in the re-generated noise-tolerant fingerprint \mathbf{F}'
τ	A vector of possible error bits to be corrected using the TRE method (for more details, please refer to Section 7.7.1).
f	One bit in the noisy raw fingerprint vector \mathbf{f}
F	One bit in the transformed noise-tolerant fingerprint vector \mathbf{F} .
θ	The noise-tolerance parameter.
x	The TRE capability—expected number of erroneous bits $ \epsilon $ in a re-generated noise-tolerant fingerprint \mathbf{F}' in the worst-case scenario (for more details, please refer to Section 7.4).
n	The group size n .
m	The block size m used in NoisFre (Baseline method).
η	The extraction efficiency, measured as how many F bits can be extracted from 1 KiB of memory.
$\text{BER}_{\mathbf{f}}$	The expected BER of raw noisy fingerprints \mathbf{f} .
$\text{BER}_{\mathbf{F}}$	The expected BER of noise-tolerant fingerprints \mathbf{F} .
\square'	An apostrophe denotes a quantity evaluated at a different time. For example, \mathbf{f}' denotes a reproduced raw noisy response.
$\ \square\ _1$	The ℓ_1 -Norm value, which is the distance of a vector from an all-zero vector, also known as the Hamming weight of a vector, as described in Definition 6.2.1 in Chapter 6.
$\text{RNG}()$	A random number generator $\text{RNG}()$ outputs a random number when invoked.
$\text{MAC}()$	A message authentication code (MAC) function.
$\text{NFL.Trans}()$	The NoisFre-Lite transformation function, which is described in more detail in Section 7.3.
Memory	Denotes the Device D 's internal memory region used for device fingerprint key generation.
WORM	A write-once-read-many memory.

NoisFre-Lite method, namely, a NoisFre-Lite-based key derivation protocol. Quantitative comparisons between the proposed method's on-device implementation overhead and existing solutions demonstrate the significant efficacy and practicability of NoisFre-Lite as a method for generating keys on highly resource-limited devices. This is followed by a discussion of potential security risks and our defensive position against them. Section 7.8 concludes this chapter.

7.1.2 Notations and Concepts

Adding to the general notations and conventions defined in Section 2.1 in Chapter 2, Table 7.1 summarises some key concepts introduced and referred to in this chapter.

7.2 NoisFre-Lite Concept

This chapter focuses on two competing challenges hindering the adoption of NoisFre device fingerprints in highly resource-constrained devices: i) limited computing capability; and ii) limited intrinsic memory capacities for generating device fingerprints. Although we have addressed the first problem in Chapter 6 by transforming raw noisy device fingerprints to a noise-tolerant space, for devices with limited memory, it is challenging to directly apply NoisFre, because it is preferential to operate NoisFre on a device with abundant memory. This eventually leads to a conflict between the need for large memory capacities and the reality posed by very limited memory in highly resource-constrained devices that *is difficult to address solely from the device side*.

We consider tackling this conflict from two aspects. First, we reconsider the transform and selection approach used during the enrolment of noise-tolerant fingerprints. We propose the relaxation of noise-tolerant parameter θ during the fingerprint selection step. According to the analytical model derived in Section 6.3 in Chapter 6 and experimentally affirmed in Figure F.3 in Appendix F, the extraction efficiency η is expected to see a significant improvement as a result. We also consider redesigning the transformation method originally proposed in Section 6.2 in Chapter 6, to further improve the extraction efficiency η . The overall affect is to increase the number of noise-tolerant bits selected by making a small compromise on the bit reliability of noise-tolerant bits.

Second, instead of solely considering a formulation that addresses the problem from the device side, we also consider the role a server can play in the process of key generation. We take advantage of the server's resource richness to mitigate the impact of the reliability compromise we made in the generation of noise-tolerant bits.

In our method, the resourceful server attempts to revert an expected number of erroneous bits in a worse case BER setting, noting that *most bits* are reliable. The resource-rich server now performs a trial-and-error process to reconcile the small number of erroneous noise-tolerant bits that could have occurred during the re-evaluation of a fingerprint on the device side. The TRE technique exploits: i) the expected worst-case noise-tolerance degree over all bits in a

7.2 NoisFre-Lite Concept

transformed \mathbf{F} (i.e., the noise-tolerance parameter θ described in Section 6.2.3 in Chapter 6); and ii) the computationally resourceful server to remove the potentially erroneous noise-tolerant bits resulting from employing those less-noise-tolerant bits in the root key. In this manner, most of the computational burden is offloaded from the highly resource-limited device to the resource-rich server to address the resource constraints on the device and improve execution performance. Importantly, in terms of the TRE algorithm, its execution time is negligible at a resourceful server. Notably, our method is inspired by the TREVERSE approach in [234] where authentication in the presence of noisy fingerprint bits is achieved by trying to reconcile a certain number of erroneous bits produced by responses with low-reliability confidence obtained from a simulatable PUF (simPUF) enrolled at the server. The SimPUF is a parameterised model that could mimic a physical device biometric source's behaviour. For example, it provides a binary fingerprint vector and corresponding bit-wise reliability confidence. Researches [61], [114] show that for SRAM device fingerprints, bit-specific reliability models can be construed to build a simPUF. However, to establish an accurate model, a considerable amount of repeated measurements are required for each hardware instance, which is not preferable in practice. In contrast, we employ an analytical model to predict and make corrections to the potentially erroneous bits within the key generated on-device, at the server.

In summary, the method in NoisFre is inadequate for providing a sufficient number of transformed noise-tolerant bits with high reliability in instances of devices with limited on-chip memory without imposing a computational burden on the device. The concept we proposed mitigates the relatively low extraction efficiency of NoisFre without increasing the burden on the device. Given our solution builds upon NoisFre but is further optimised for highly resource-constrained devices, we name this solution **NoisFre-Lite**.

7.2.1 An overview of the NoisFre-Lite Scheme

We describe the proposed NoisFre-Lite scheme for deriving a highly reliable fingerprint, for example, to serve as a secure key, for security functions in Figure 7.1. The NoisFre-Lite fingerprint enrolment process is illustrated in Figure 7.1 (a), and the NoisFre-Lite fingerprint re-generation and its use in a security function is depicted in Figure 7.1 (c).

The fingerprint enrolment is an one-time process carried out in a secure environment. The raw, noisy fingerprint \mathbf{f} is read out from the highly resource-constrained device's memory space. The resourceful server employs a transformation and selection algorithm, as detailed in Figure 7.1 (b). The transformation and selection algorithm takes the \mathbf{f} as input, parameterised

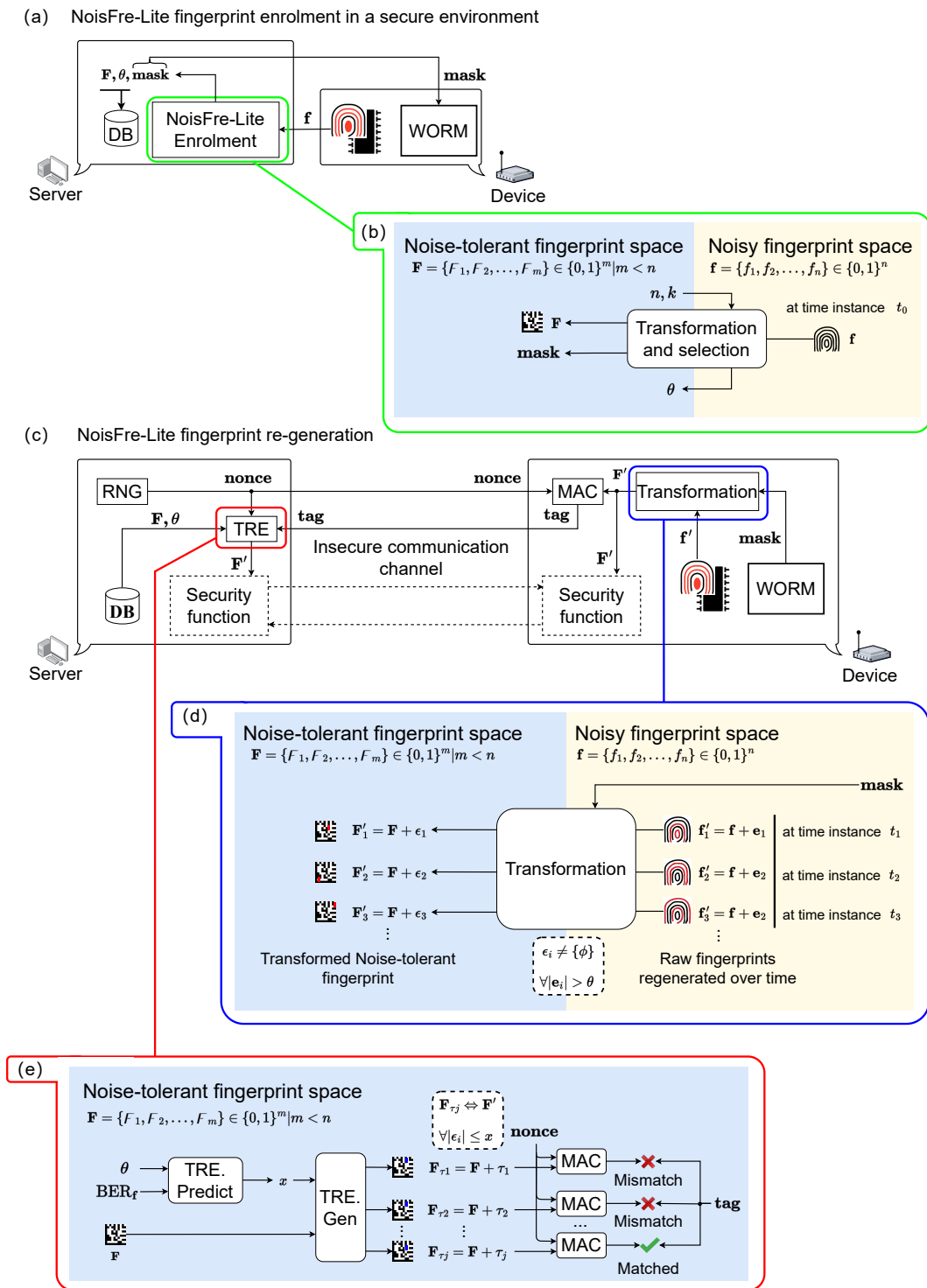


Figure 7.1: An overview of NoisFre-Lite scheme: (a) NoisFre-Lite fingerprint enrolment process; (b) An illustration of transformation and selection function; (c) NoisFre-Lite fingerprint re-generation process; here, the re-generated fingerprint \mathbf{F}' is used directly as a secret key for a security function; (d) The function of the lightweight noise-tolerant fingerprint transformation block on the *device*; and (e) The computationally intensive TRE block outsourced to the *server*.

7.2 NoisFre-Lite Concept

with group size n and the desired number of fingerprint bits k (parameters decided by a security practitioner, for instance k may be 128 bits, used as a root key). The transformation and selection algorithm generates three outputs: i) the transformed noise-tolerant fingerprint \mathbf{F} ; ii) a **mask** indicating the selected devices memory addresses used to produce \mathbf{F} ; and iii) a noise-tolerant parameter θ . \mathbf{F} and θ are securely stored in the server's database **DB** to provide information necessary for the TRE function in subsequent re-generation phases. The **mask** can be stored in a device's WORM, alternatively the **mask** can also be stored on the server and supplied to the device during the fingerprint re-generation phase, as discussed in Section 6.5.1 in Chapter 6.

The NoisFre-Lite transformation and selection algorithm transforms device fingerprints from the noisy fingerprint space to the noise-tolerant fingerprint space, as originally proposed in NoisFre (see Section 6.2 in Chapter 6). However, the new algorithm: i) aims to consider a trade-off of reliability for improved extraction efficiency; ii) improves the extraction efficiency η by removing the $n \times m$ -bit block-based transformation approach used in NoisFre; and iii) instead of treating θ as an immutable value to achieve the desired F bit error rate, the new transformation and selection algorithm adaptively relaxes the constraint imposed by θ to transform and select more F bits.

After the highly resource-constrained device is deployed in field, all communications between the server and the device are initiated through an insecure communication channel. The NoisFre-Lite fingerprint re-generation provides a basis for the underlying security functions, for example to build a secure communication channel. First, the device dynamically generates a new noisy raw fingerprint \mathbf{f}' from its memory. Notably, the noisy fingerprints generated at each time instance t_i are likely to contain different errors \mathbf{e}_i . The transformation function is identical to NoisFre D-Norm-based transformation described in Section 6.2.3 in Chapter 6. However, due to the side-effect of trading reliability for efficiency in the enrolment phase, the reliability attributes of F bits obtained from the memory addresses located by the **mask** maybe inadequate. If the number of error bits ($|\mathbf{e}_i|$) is greater than the noise-tolerant parameter θ , for a noisy fingerprint \mathbf{f}_i re-generated at time t_i , the transformed \mathbf{F}' in the noise-tolerant fingerprint space will suffer from error bits ϵ_i . The existence of ϵ_i make the noise-tolerant \mathbf{F}'_i to be unequal to the enrolled \mathbf{F} , and render it difficult to use directly in a security function, for example, as a root key.

Instead of selecting only noise-tolerant bits F with a high noise-tolerance level, as in NoisFre, NoisFre-Lite allows error bits ϵ to exist in re-generated \mathbf{F}' . As a counter measure, we propose the TRE technique to resolve those erroneous bits ϵ , as illustrated in Figure 7.1 (e), based on the enrolled reference \mathbf{F} , the θ and worst-case BER_f (defined in Section 2.3.2 in Chapter 2) recorded from a given memory technology and manufacture.

In realising the TRE method, to avoid directly exposing \mathbf{F}' over the insecure communication channel, the server generates a **nonce** from its random number generator $\text{RNG}()$, and the **nonce** is transmitted to the device. A MAC **tag** is computed by the device over the **nonce** with a $\text{MAC}()$ function keyed with the \mathbf{F}' . Subsequently the **tag** is sent to the server. Now, provided the number of erroneous bits $|\epsilon|$ is below the TRE capability x , the server is guaranteed to reconstruct a $\mathbf{F}_{\tau i}$ from the referenced \mathbf{F} and exactly match the re-generated \mathbf{F}' on the device. Recall that the TRE is executed by the resource-rich server, and does not add an overhead to the highly resource-constrained device; hence, NoisFre-Lite requires the same on-device overhead as NoisFre.

This intuitive combination of NoisFre and the approach for tolerating errors with a trial-and-error process—TRE—is the basis for NoisFre-Lite; a method for lightweight memory-root-key generation even: i) under substantial memory constraints; and ii) without the overhead imposed by relying on the reverse-fuzzy-extractor-based methods as discussed in Section 3.3.3 in Chapter 3. The next section details the two transformation and selection algorithms proposed for improving extraction efficiency.

7.3 Transformation and Selection Algorithms

This section develops two new transformation and selection algorithms (the Fixed-d and the Variable-d) that address the key limitation of NoisFre—its relatively low extraction efficiency (see Section 6.9 in Chapter 6). Although this was not an impediment for devices with large enough memory capacity for fingerprinting (e.g., 512 KiB), it is very difficult to integrate NoisFre into highly resource-constrained devices (e.g., the CRFID device studied in Chapter 3 with only 2 KiB of SRAM available for device fingerprinting).

As a basis for the algorithms, we adopt the D-Norm transform developed in Chapter 6 and this ensures that we can adopt the analytical model developed in Chapter 6 to analyse the performance of the new methods. Here, we consider the following:

- **Constraint.** Each bit vector used for generating an ℓ_1 -Norm can be used only once. This avoids the risk of correlations between transformed fingerprint bits.
- **Goal.** The primary goal is to maximise extraction efficiency to facilitate the use of memory fingerprints for security function in highly resource-constrained devices (i.e., devices with an extremely small amount of memory available for fingerprinting).

7.3 Transformation and Selection Algorithms

In NoisFre, the extraction efficiency η is inversely proportional to the noise-tolerance parameter θ , as formulated in equation (6.11) in Chapter 6 and experimentally affirmed in Figure F.3 in Appendix F. The most direct approach to increasing the η is to simply reduce the θ value. But, has the side effect of sacrificing the transformed noise-tolerant fingerprint's reliability. This becomes the central concept of NoisFre-Lite: *to trade reliability for improved extraction efficiency*.

In addition to simply decreasing the θ value, we study the NoisFre transformation algorithm itself to understand whether we can improve η . Here, it is worth recapitulating the D-Norm-based transformation and selection procedure developed in Chapter 6 (referred to as the *Baseline* method in this chapter).

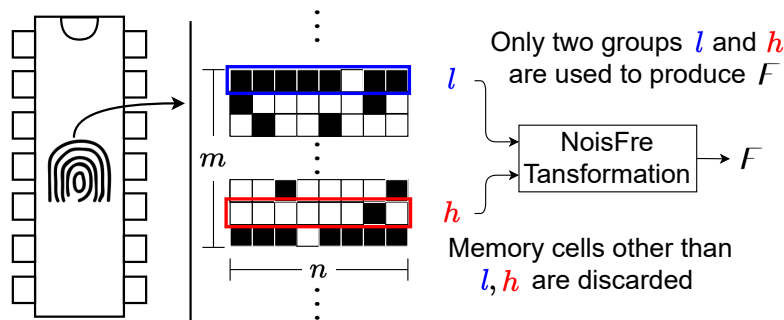


Figure 7.2: The problem with the Baseline D-Norm-based transformation and selection procedure. Within an $n \times m$ block, only two groups (h and l) produce a single F bit, whilst the remaining $m - 2$ groups are simply discarded.

The Baseline method generated only one noise-tolerant fingerprint bit F from $n \times m$ raw memory bits. The binary value of the F bit was only determined by the relative address of h (the group with the highest ℓ_1 -Norm value) and l (the group with the lowest ℓ_1 -Norm value). This approach *discarded the vast majority of memory cells*, as exemplified in Figure 7.2. However, beyond the selected (h , l) pair, there may exist combinations that also meet the noise-tolerance parameter θ . If we could utilise as many memory cells as possible, we might expect extraction efficiency to improve and facilitate an adaptation to a limited memory setting. Thus, we recognise that the block-based selection method is a significant cause of the low extraction efficiency observed in NoisFre. Therefore, we consider methods to remove this block-based restriction. This leads to the problem of how transformation and selection can be performed across the memory space whilst maximising extraction efficiency and meeting bit specific reliability requirements.

Interestingly, the ℓ_1 -Norm computed from SRAM start-up values can assess the noise-tolerance of a raw fingerprint bit vector or provide a measure of noise-tolerance. Thus, in contrast to

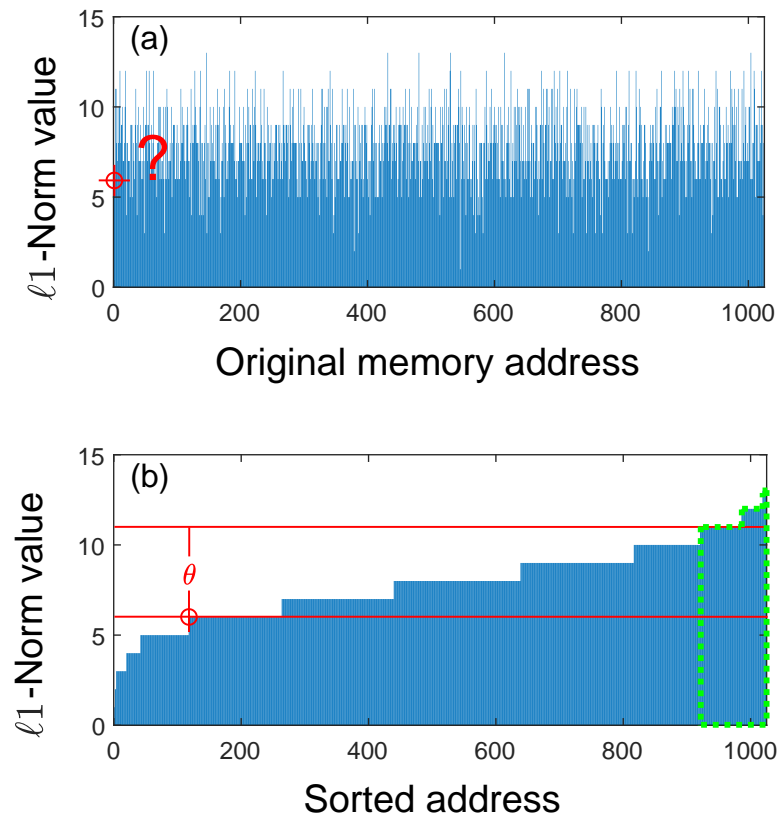


Figure 7.3: The basis for the two proposed methods: (a) the ℓ_1 -Norm pattern before sorting and (b) after ℓ_1 -Norms are sorted in an ascending order. The data is randomly sampled from the synthetic chip model described in Section F.1.1, with parameter $n=16$. For a given ℓ_1 -Norm (marked by red circle) it is much easier to locate another ℓ_1 -Norm (from the region marked with green dash line) to satisfy the selection condition $|h - l| \geq \theta$ whilst reducing the discarded ℓ_1 -Norm bit vectors after the sorting.

employing blocks of bits to apply a transformation method and discarding unused groups of bit vectors, we propose sorting groups of n -bit vectors based on their ℓ_1 -Norm values, prior to transformation and selection of noise-tolerant bits. This approach allows pairing two n -bit vectors from the sorted list, where $2 \times n$ bits can be transformed into a F bit as opposed to $m \times n$ block of bits. Therefore, the proposed approaches could contribute to making effective use of raw fingerprint values. The approaches proposed are dissimilar to DRV-hashing proposed by Xu *et al.* [115], which we discussed in Section 2.3 in Chapter 2. DRV-hashing sorts each SRAM cell’s DRV value in ascending order and matches those pairs from the two ends in the sorted table to generate the PUF response. This technique maximises resilience against small DRV fluctuations, common-mode DRV shift induced by thermal noise and changes in operational temperature. Notably, in contrast to the problem posed by measuring DRV values in commodity devices—the authors [115] recognised that measuring DRV requires generating fine-tuned test voltages on device, a feature not yet supported by most commodity devices—there are no special requirements on the hardware platform when employing ℓ_1 -Norm functions. Hence,

7.3 Transformation and Selection Algorithms

the approaches we propose are generic, practical, and forms the basis for the transform and selection methods we investigate and simplifies the analysis of the approach.

To illustrate the basis for the two algorithms we propose, Figure 7.3 shows sorted ℓ_1 -Norms obtained from a synthetic chip model. Prior to sorting, it is difficult to search for two groups that satisfies the selection condition $|h - l| \geq \theta$ that reduces the discarded ℓ_1 -Norms bit vectors, as depicted in Figure 7.3 (a), to transform to a F bit. Previously, we employed a block of $n \times m$ bits and selected within a block, as hinted in Figure 7.3. However, after the ℓ_1 -Norms are sorted, as shown in Figure 7.3 (b), for any ℓ_1 -Norm value we can search and locate another ℓ_1 -Norm to form a pair for a D-Norm transformation and selection by evaluating whether the vertical distance is greater than or equal to θ in a manner that reduces the discarded ℓ_1 -Norms bit vectors.

Therefore, instead of using the $n \times m$ block-based method as in NoisFre, we apply sorting as a common pre-processing step for the two transformation and selection algorithms developed in this section. Having sorted ℓ_1 -Norms, the next step is to develop algorithms capable of performing transformation and selection from the sorted ℓ_1 -Norms with due consideration for extraction efficiency.

7.3.1 Sorted D-Norm Selection with Fixed Distance (Fixed-d)

We first develop a variant of the D-Norm-based transformation and selection method with the design target of maximising extraction efficiency by evenly selecting the h, l pairs with the same fixed distance (fixed $|h - l|$ value). This is based on the *wooden barrel principle*: the capacity of a barrel is determined by the shortest wooden bar. If we select all h, l pairs with the same $|h - l|$ value, there will be no ‘short bar’ that fails the selection condition θ . Therefore, we maximise extraction efficiency. We refer to this concept as sorted D-Norm selection with fixed distance (Fixed-d) and illustrate it in the toy example in Figure 7.4 (a) before describing it in detail in Algorithm 1. The algorithm starts from the largest θ value (i.e., $\theta = n$). Following an exhaustive search, if the number of F bits extracted remains below the required fingerprint size k , the proposed θ is relaxed (in decrements of one) to trade reliability for improved extraction efficiency. Subsequently, the search is repeated to determine if the new value of noise tolerance, θ , is able to meet the required fingerprint size k .

In contemplating a Fixed-d method, it is intuitive to consider the consequence of not attempting to determine and select all $|h - l|$ bit blocks satisfying a θ value (the minimal distance between

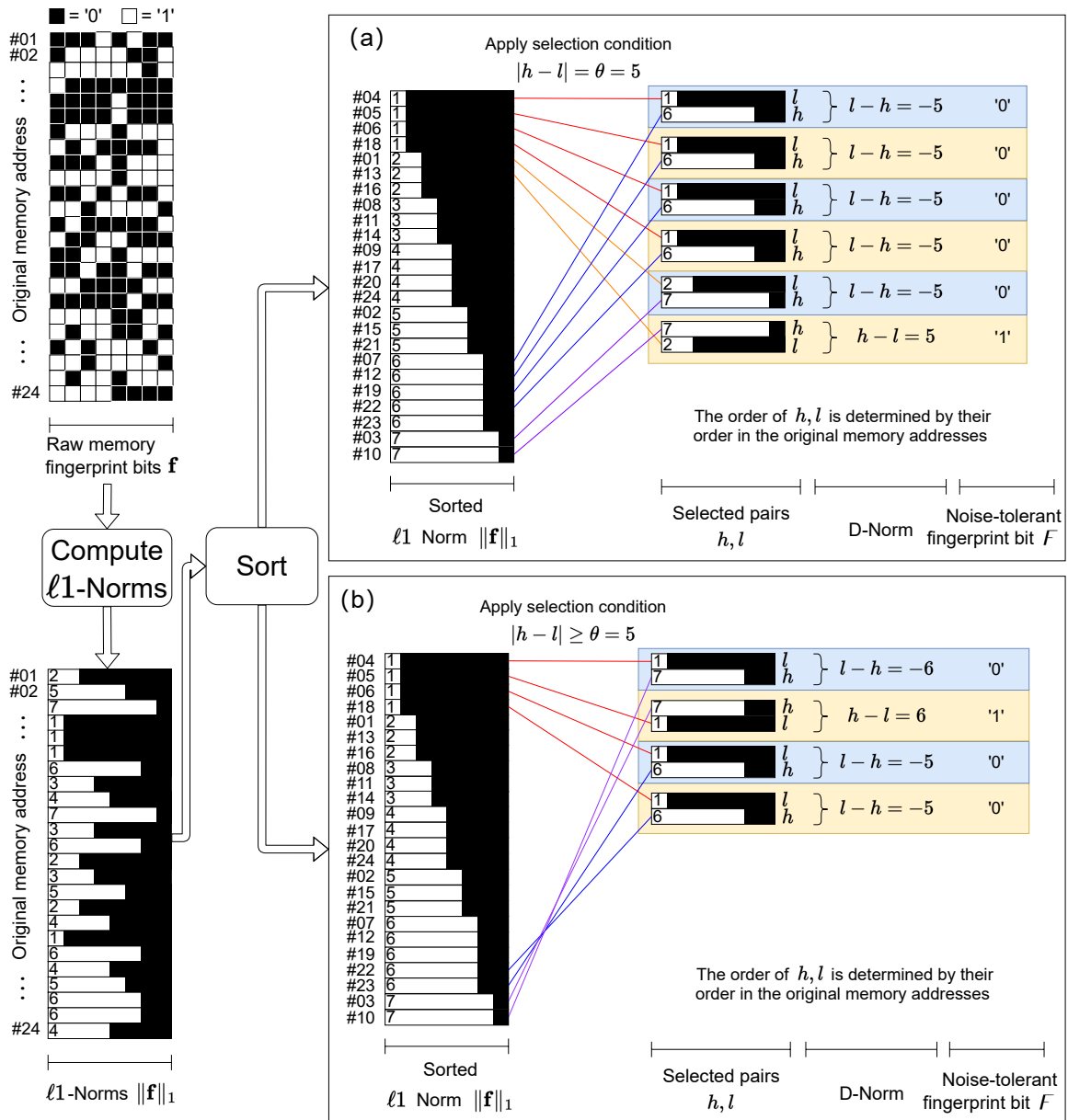


Figure 7.4: Sorted D-Norm with: (a) fixed distance (Fixed-d); and (b) variable distance (Variable-d). We select to use $n = 8$, assume $\theta = 5$ was obtained through Algorithm 1 and we use the same θ value in both methods, and assume $BER_f = 5.30\%$ (the same as the MSP20 dataset across the temperature range of 0°C to 40°C studied in Section 3.2.2 in Chapter 3). The Fixed-d method extracts six F bits from 192 raw bits (32.00 bit/KiB), and the Variable-d extracts four F bits from 192 raw bits (21.34 bit/KiB). The $|h - l|$ value of Fixed-d is 5 and the average $|h - l|$ value of Variable-d is 5.5, corresponding to BER_F of 3.23×10^{-4} and 1.87×10^{-4} , respectively. In summary, Fixed-d achieves higher extraction efficiency, and Variable-d demonstrates better fingerprint reliability. Here, the # tag at the left side of each sorted ℓ_1 -Norm denotes its original memory address in the raw memory fingerprints.

h and l that can be used) in selecting k, F bits. Accordingly, we propose the Variable-d method as an alternative.

7.3 Transformation and Selection Algorithms

Algorithm 1: The Fixed-d method.

Input: raw noisy memory fingerprint \mathbf{f} , group size n and required noise-tolerant fingerprint bit size (e.g. key size) k ;

Output: transformed noise-tolerant fingerprint \mathbf{F} , noise-tolerance parameter θ and a **mask** indicates all memory addresses selected to produce F bits;

```
1 Reshape  $\mathbf{f}$  into  $n$ -bit groups and calculate the  $\ell_1$ -Norm for each group;
2 Sort  $\ell_1$ -Norms in ascending order and obtain a sorted sort_norm;
3 The orig_idx monitors each group's order in the original unsorted  $\ell_1$ -Norm;
4  $\mathbf{F} \leftarrow \text{empty}$ ;
5 mask  $\leftarrow \text{empty}$ ;
6 Mark all elements in sort_norm as available;
7 for  $\theta = n, n - 1, \dots, 0$  do
8   for  $i = 1, 2, \dots, |\text{sort\_norm}|$  do
9     if sort_norm[ $i$ ] is unavailable then
10      skip and continue with  $i = i + 1$ ; // one  $\ell_1$ -Norm can only be used
11      once.
12    end if
13     $l \leftarrow \text{sort\_norm}[i]$ ;
14    for  $j = i+1, i+2, \dots, |\text{sort\_norm}|$  do
15       $h \leftarrow \text{sort\_norm}[j]$ ;
16      if sort_norm[ $i$ ] is unavailable OR  $\theta > |h - l|$  then
17        skip and continue with  $j = j + 1$ ; // one  $\ell_1$ -Norm can only be
18        used once, and selected pairs must meet the
19        selection condition.
20      end if
21      if orig_idx[ $i$ ] > orig_idx[ $j$ ] then
22         $F \leftarrow 1$ ;
23      else
24         $F \leftarrow 0$ ;
25      end if
26      Append  $F$  to the  $\mathbf{F}$ ;
27      Append orig_idx[ $i$ ] and orig_idx[ $j$ ] to the mask;
28      Mark both sort_norm[ $i$ ] and sort_norm[ $j$ ] as unavailable;
29      if  $k \leq |\mathbf{F}|$  then
30        return  $\mathbf{F}$ ,  $\theta$  and mask;
31      end if
32    end for
33  end for
34  /* Unable to extract  $k$  noise-tolerant bits, try  $\theta = \theta - 1$ 
35     (trade reliability for improved extraction efficiency).
36     */
37 end for
```

Algorithm 2: The Variable-d method.

Input: raw noisy memory fingerprint \mathbf{f} , group size n and required noise-tolerant fingerprint bit size (e.g. key size) k ;

Output: transformed noise-tolerant fingerprint \mathbf{F} , noise-tolerance parameter θ and a **mask** indicates all memory addresses selected to produce F bits;

```

1 Reshape  $\mathbf{f}$  into  $n$ -bit groups and calculate the  $\ell_1$ -Norm for each group;
2 Sort  $\ell_1$ -Norm in ascending order and obtain a sorted sort_norm;
3 The orig_idx keeps track of each group's order in the original unsorted  $\ell_1$ -Norm;
4  $\mathbf{F} \leftarrow \text{empty}$ ;
5 mask  $\leftarrow \text{empty}$ ;
6 Mark all elements in sort_norm as available;
7 for  $\theta = n, n - 1, \dots, 0$  do
8   for  $i = 1, 2, \dots, |\text{sort\_norm}|$  do
9     if sort_norm[ $i$ ] is unavailable then
10      skip and continue with  $i = i + 1$ ; // one  $\ell_1$ -Norm can only be used
11      once.
12    end if
13     $l \leftarrow \text{sort\_norm}[i]$ ;
14    for  $j = |\text{sort\_norm}|, |\text{sort\_norm}|-1, |\text{sort\_norm}|-2, \dots, i+1$  do
15       $h \leftarrow \text{sort\_norm}[j]$ ;
16      if sort_norm[ $i$ ] is unavailable OR  $\theta > |h - l|$  then
17        skip and continue with  $j = j - 1$ ; // one  $\ell_1$ -Norm can only be
18        used once, and selected pairs must meet the
19        selection condition.
20      end if
21      if orig_idx[ $i$ ] > orig_idx[ $j$ ] then
22         $F \leftarrow 1$ ;
23      else
24         $F \leftarrow 0$ ;
25      end if
26      Append  $F$  to the  $\mathbf{F}$ ;
27      Append orig_idx[ $i$ ] and orig_idx[ $j$ ] to the mask;
28      Mark both sort_norm[ $i$ ] and sort_norm[ $j$ ] as unavailable;
29      if  $k \leq |\mathbf{F}|$  then
30        return  $\mathbf{F}$ ,  $\theta$  and mask;
31      end if
32    end for
33  end for
34  /* Unable to extract  $k$  noise-tolerant bits, try  $\theta = \theta - 1$ 
35   (trade reliability for improved extraction efficiency).
36  */
37 end for

```

7.3.2 Sorted D-Norm Selection with Variable Distance (Variable-d)

The sorted D-Norm with variable distance (Variable-d) is similar to the Fixed-d method and also requires that the ℓ_1 -Norms be sorted first, as exemplified in Figure 7.4 (b). In contrast to Fixed-d, the Variable-d approach aims to maximise the number of highly reliable F bits from a given memory size. Notably, it only differs from the Fixed-d method in Algorithm 2 at lines 13 and 16: instead of formulating a search for the $|h - l| = \theta$ to employ, the Variable-d method attempts to select the h, l pairs with the largest possible $|h - l|$ values before gradually selecting pairs with smaller $|h - l|$ values until no further pairs, subject to the noise-tolerance parameter θ , are available. Again, the θ value will be relaxed (in decrements of one) if the requested noise-tolerant fingerprint size k cannot be satisfied. This is indeed the basis of the proposed methods—trading reliability for improved extraction efficiency.

7.3.3 Summary

This section has proposed two new variants of the D-Norm-based fingerprint transformation and selection method, namely, the Fixed-d method and the Variable-d method. Both approaches share the same basic principle of trading reliability for improved extraction efficiency, and both sort ℓ_1 -Norm values before performing transformation and selection. Sorting is only conducted once by the server during the enrolment phase (i.e., the on-server fingerprint enrolment as described in Section 7.7.1) and is no longer required during subsequent fingerprint re-generations.

The Fixed-d method provides better extraction efficiency (first, selecting *all* F bits that satisfies the same noise-tolerance parameter θ as possible) but, in doing so, compromises reliability. Meanwhile, the Variable-d method achieves better reliability for the transformed fingerprint \mathbf{F} . Because, it attempts to first, select F bits that maximise the differential, $|h - l|$, values. Consequently, a resultant fingerprint bit vector (for example the secret key) will possess a lower $\text{BER}_{\mathbf{F}}$. However, it compromises extraction efficiency.

Thus, it is apparent that trading reliability for improved extraction efficiency can engender unacceptably high key failure rates, as Section 6.4.2 in Chapter 6 illustrates. The next section addresses this dilemma.

7.4 Tolerating Errors with Trial-and-error

This section considers the problem posed by the desire to increase both extraction efficiency and fingerprint reliability. As Section 7.2 recognised, it is difficult to solve this dilemma within a highly resource-limited device’s capabilities. Accordingly, it is imperative to consider other parties in a communication system, which reveals that the two essential entities in a typical communication system (shown in Figure 7.1) are highly heterogeneous, meaning that the server is much more resourceful than the device. Therefore, we propose a trial-and-error (TRE) technique to reconcile errors caused by the resulting, reduced noise-tolerance parameter θ (a function of trading reliability for improved extraction efficiency as detailed in Section 7.3), by utilising the server’s plentiful computational power. TRE is performed by the server; consequently, places no additional implementation overhead on the highly resource-limited device. The following subsections elaborate on the TRE technique developed in NoisFre-Lite.

7.4.1 Design of the TRE algorithm

We start with the simplifying assumption that each F bit in the noise-tolerant fingerprint is equally likely to be erroneous. However, it is possible for each F bit to have different bit error rates; the exploitation of this additional information, we leave for future work (for more details, please refer to Section 8.3.6 in Chapter 8). In a practical application, a k -bit noise-tolerant fingerprint \mathbf{F} will fail to serve as a root key in security functions if there are more than *zero* erroneous bits. The probability of such a key failing is described analytically in equation (7.1).

$$P_{\mathbf{F}}^{\text{fail}} = 1 - \text{binocdf}(0, k, \text{BER}_{\mathbf{F}}) \quad (7.1)$$

Where $\text{binocdf}()$ is the cumulative function of the Binomial distribution and the $\text{BER}_{\mathbf{F}}$ is the expected bit error rate of noise-tolerant fingerprints \mathbf{F} . Notably, in NoisFre-Lite the D-Nrom selection criterion ($|h - l| \geq \theta$) still holds, therefore, we can continue to use the equation (6.9) derived in Section 6.3 in Chapter 6 to calculate the $\text{BER}_{\mathbf{F}}$ for a given BER_f and θ .

The TRE approach will be able to correct up to x error bits (i.e., expected number of erroneous bits in a re-generated \mathbf{F}' in the worst-case operating condition) by exhaustively attempting to flip the power set of binary sub-stings in the key. We use a toy example to facilitate understanding of the power set. For example, a set $\mathbb{S} = \{a, b, c\}$ has three elements a , b , and c . The power set of \mathbb{S} is $\mathcal{P}(\mathbb{S}) = \{\phi, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$, where ϕ denotes an empty set. With TRE, the maximum number of allowable erroneous bits $|\epsilon|$ in \mathbf{F}' can be relaxed from zero

7.4 Tolerating Errors with Trial-and-error

to x bits, since the TRE method is employed to mitigate the impact of erroneous bits. Now, the expected key failure rate $P_{\mathbf{F}}^{\text{fail}}$, under at most x bit errors, becomes:

$$P_{\mathbf{F}}^{\text{fail}} = 1 - \text{binocdf}(x, k, \text{BER}_{\mathbf{F}}). \quad (7.2)$$

Hence, the TRE capability, x , for any \mathbf{F} with size k and known $\text{BER}_{\mathbf{F}}$ can be determined using equation (7.3), where $P_{\mathbf{F}}^{\text{fail}}$ is the expected failure rate of the key desired, after applying the TRE technique.

$$(\text{TRE Capability}) \ x = \text{binoinv}(1 - P_{\mathbf{F}}^{\text{fail}}, k, \text{BER}_{\mathbf{F}}) \quad (7.3)$$

Where $\text{binoinv}()$ is the inverse function of the Binomial cumulative density function $\text{binocdf}()$. Therefore, the number of bits to be tried using the TRE approach (the x) can be easily estimated using equation (7.3), given knowledge of the k , the $\text{BER}_{\mathbf{F}}$ and the desired $P_{\mathbf{F}}^{\text{fail}}$. Notably, given that the TRE capability is derived based on assuming the worst-case $\text{BER}_{\mathbf{F}}$, the determined value of x accounts for the worst-case operating condition.

We refer to the x as the TRE capability—expected number of erroneous bits $|\epsilon|$ in a re-generated \mathbf{F}' in the worst-case scenario. That is, for any number of error bits $\epsilon : |\epsilon| \leq x$, the on-server TRE mechanism can effectively reconstruct all error bits.

An exemplar case (see Figure 7.5) can facilitate understanding of the mechanics of the TRE approach. First, we have a re-generated fingerprint \mathbf{F}' marked ① that contains several error bits ϵ (highlighted in red boxes). The re-generated fingerprint \mathbf{F}' is secure and never exposed to other parties, meaning neither the server nor a hypothetical attacker has direct access to the noise-tolerant fingerprint, dynamically generated on-demand. Instead, the device computes **tag** (a MAC tag) over the re-generated fingerprint and makes the tag available to the server and the tag is assumed to be public information.

Although the server may compute the MAC tag over the reference fingerprint ②, the existence of potential error bits ϵ could lead to the two MAC tags mismatching. The server employs the analytical model for expected number of bit errors, equation (7.3), to predict the number of error bits that could exist in the re-generated fingerprint. This example uses $x = 2$, which assumes the number of error bits ϵ is less than or equal to 2.

The server, given its abundant computational capability, can search $\binom{k}{x}$ combinations to examine all possible locations of error bits as in ③. For each for each error pattern, the server employs the systematic strategy of using the power set described earlier to try to revert any possible

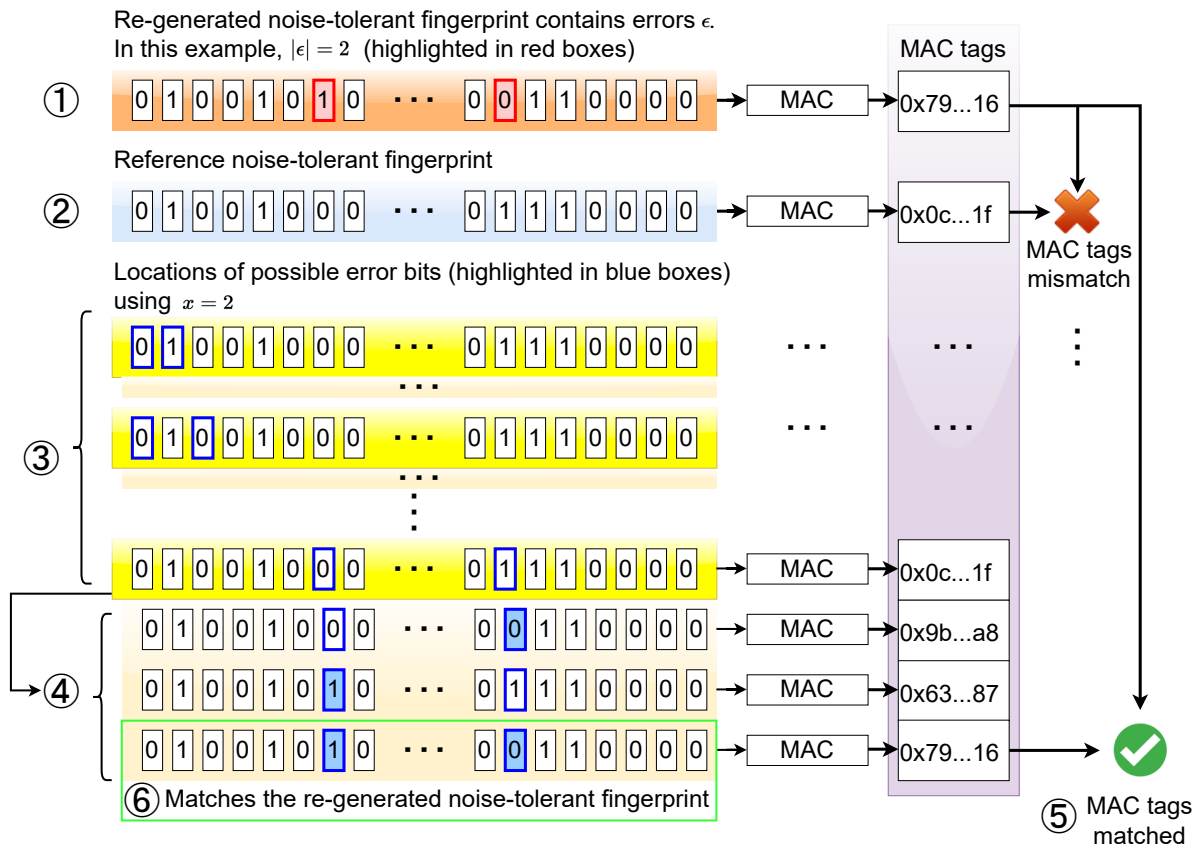


Figure 7.5: Example demonstrating the TRE approach, showing: ① a re-generated noise-tolerant fingerprint which contains error bits ϵ , in this example, the number of error bits is $|\epsilon| = 2$ (highlighted in red boxes); ② a reference noise-tolerant fingerprint securely managed by the server; ③ the server generates all possible combinations of error bit locations, for $x = 2$ in this example; ④ for each possible combination of error bit locations, generate the power set and attempt to reconcile error bits. Here we only expand and demonstrate with one bit vector in ④ containing the successful match for the re-generated noise-tolerant fingerprint; ⑤ a matched MAC tag indicates the current noise-tolerant fingerprint generated in TRE ⑥ matches the noise-tolerant fingerprint re-generated on the device. We describe the approach underlying the TRE method in Algorithm 3.

combinations of error bits, as indicated by ④. If the MAC tag value of the current TRE fingerprint F_r matches the MAC tag received from the remote device, as in ⑤, we ascertain that the current TRE fingerprint is identical to the re-generated fingerprint ⑥. We elaborate the on-server TRE algorithm in Algorithm 3. The on-server TRE algorithm takes several input values including the enrolled noise-tolerant fingerprint F , its size k , the TRE capability x predicted using equation (7.3), the same random **nonce** sent to the device, the MAC tag received from the device.

Having now elaborated two new transformation and selection algorithms that trade reliability for improved efficiency and mitigated the consequent increase in unreliability using this section's TRE method, benefiting from the server's rich computing power, the following section develops

7.5 Formalising Performance Measures

Algorithm 3: The on-server TRE generation algorithm.

Input: The referenced noise-tolerant fingerprint \mathbf{F} , the size of the noise-tolerant fingerprint k , the TRE capability x predicted using equation (7.3), the same **nonce** as sent to the device, MAC **tag** received from the device;

Output: Re-generated noise-tolerant fingerprint \mathbf{F}' ;

```
1 tag' ← MACF(nonce);
2 if tag' = tag then
3   | return F' ← F;           // there is no error in the re-generated F'
4 end if
5 /* Search  $\binom{k}{x}$  combinations to examine all possible locations
   of error bits in  $\mathbf{F}$ . Here,  $\mathbb{L}$  is a set with  $x$  elements. */
6 for  $\mathbb{L}$  in  $\binom{k}{x}$  do
7   /*  $\mathcal{P}(\mathbb{L})$  is the power set generated from  $\mathbb{L}$ . */
8   for  $\tau$  in  $\mathcal{P}(\mathbb{L})$  do
9     |  $\mathbf{F}_\tau$  ← (invert bits in  $\mathbf{F}$  at indices  $\in \tau$ );
10    | tag' ← MAC $\mathbf{F}_\tau$ (nonce);
11    | if tag' = tag then
12      | return F' ←  $\mathbf{F}_\tau$ ;           // Fingerprint re-generation is
13      | successful
14    | end if
15  end for
16 return  $\perp$ ;           // Unable to reconcile error bits, fingerprint
re-generation is failed
```

analytical models for the proposed methods to support the practical use of NoisFre-Lite, ease analysis and evaluations.

7.5 Formalising Performance Measures

This section develops analytical models for the two variants of the D-Norm-based noise-tolerant fingerprint transformation and selection methods developed in Section 7.3: the Fixed-d and the Variable-d. Same as Chapter 6, our main focuses are key failure rate and extraction efficiency. We also formulate the on-server time complexity associated with executing the TRE mechanism developed in Section 7.4. This section's analysis is based on the synthetic chip model detailed in Appendix F.1.

For a given group size n , the occurrence of a ℓ_1 -Norm equals to q ($q \in \{0, \dots, n\}$) can be expressed as:

$$P(\|\mathbf{f}\|_1 = q) = \text{binopdf}(q, n, 0.5) = \binom{n}{q} \cdot 0.5^q \cdot (1 - 0.5)^{n-q} \quad (7.4)$$

Where $\text{binopdf}()$ denotes the Binomial probability density function. The constant 0.5 indicates in an n -bit group, each bit has 50% probability of being '1'/'0' (recall, the synthetic chip model is constructed based on an assumption). Indeed, as shown in Figure 7.6, the data collected from physical chips is in a good agreement with the theoretical distribution of ℓ_1 -Norms based on equation (7.4). Therefore, we use equation (7.4) as the basis of all of the analyses in this section.

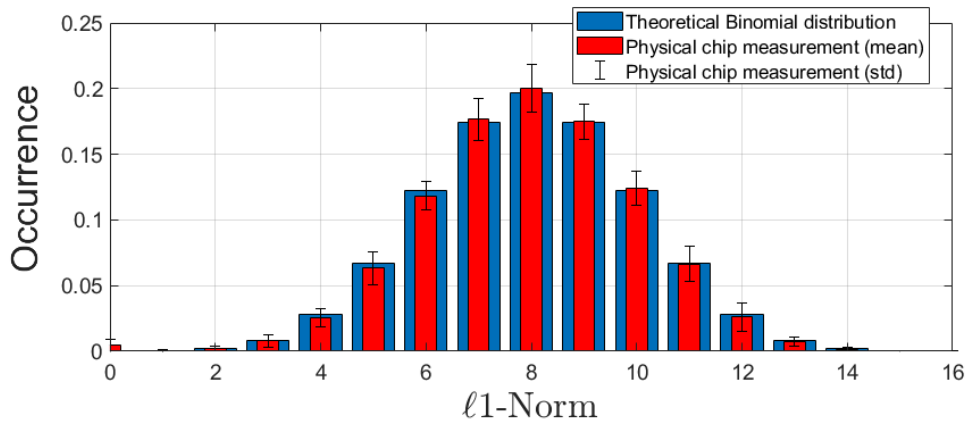


Figure 7.6: The distribution generated from data collected from the 25 physical chips studied in this chapter (for more details about physical chips, please refer to Section 7.6) is in a good agreement with the theoretical Binomial distribution plot of ℓ_1 -Norms based on equation (7.4). The physical chip has memory spaces of 2 KiB for device fingerprinting. Both the theoretical distribution and the physical chip measurement are obtained with parameter $n=16$.

7.5.1 Key Failure Rate

For device fingerprint reliability, we continue to use the bit error rate measurement, detailed in Section 6.3.1 in Chapter 6, to describe the reliability of F bits in \mathbf{F} . We further consider the key failure rate $P_{\mathbf{F}}^{\text{fail}}$, introduced in Section 6.5.2 in Chapter 6, for a k -bit \mathbf{F} .

Notably, in NoisFre-Lite the D-Nrom selection criterion ($|h - l| \geq \theta$) still holds, therefore, we can continue to use the equation (6.9) derived in Section 6.3 in Chapter 6 to calculate the $\text{BER}_{\mathbf{F}}$ for both Fixed-d and Variable-d with a given $\text{BER}_{\mathbf{f}}$ and resultant θ obtained from each algorithm.

We recognise that, in Variable-d method, the resultant θ is the smallest value determined and hence employing this value results in a worse than expected $\text{BER}_{\mathbf{F}}$ in practice. However, we

focus on considering the worst-case scenario, where all F bits are assumed to meet $|h - l| = \theta$. Therefore, the overall failure rate $P_{\mathbf{F}}^{\text{fail}}$ of a k -bit noise-tolerant key \mathbf{F} can still be expressed using equation (6.12) for both methods.

7.5.2 Extraction Efficiency

Following the definition in Section 6.3.2 in Chapter 6, we describe extraction efficiency in terms of the number of F bits extracted per unit memory space (1 KiB).

Fixed-d Extraction Efficiency. To analyse the extraction efficiency of the Fixed-d method, we apply the Algorithm 1 described in Section 7.3.1 to a demonstrative example with the following settings: memory size $|\text{mem}| = 2048$ bytes, $n = 16$ bits, $\theta = 3$. The selected ℓ -Norms are visualised in Figure 7.7, to ease understanding, selected l are coloured red, and selected h are coloured yellow, where $|h - l| = \theta$.

In the probability density function plot Figure 7.7, the area indicates probability. Hence, the extraction efficiency of Fixed-d, η_{Fix} , can be derived by simply considering the area coloured whilst employing the Fixed-d method. Because, the combination of a pair of ℓ 1-Norms, l and h , two groups produces one noise-tolerant bit F . In such case, only one coloured area (red or yellow) needs to be accounted for extraction efficiency.

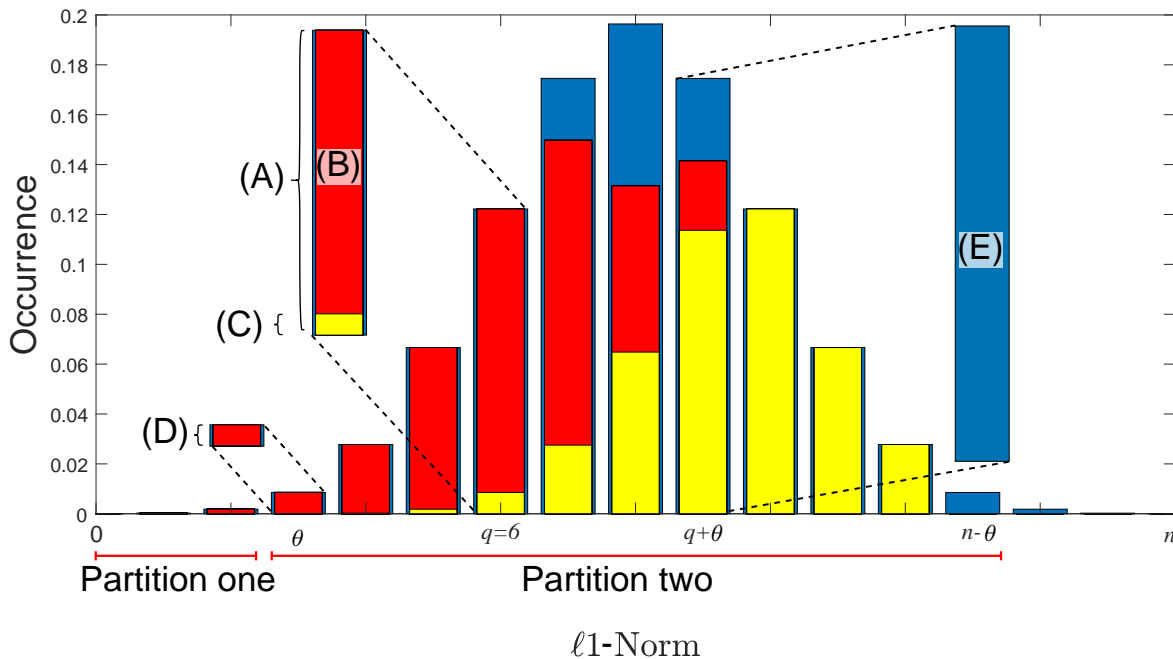


Figure 7.7: In Fixed-d method, when $\theta \leq \frac{n}{2}$, for example, $n = 16$ and $\theta = 3$, the selected l (coloured red), and the selected h (coloured yellow) are closely interwoven.

Subject to the D-Norm selection criterion $|h - l| = \theta$, the selected l values (area coloured in red) may span $\{0, 1, \dots, n - \theta\}$. Hence, q , as described in equation (7.4), is in the range $[0, n - \theta]$. To facilitate the analysis, we can split the red area into two partitions. First, for q in $[0, \theta - 1]$, all $\ell 1$ -Norms are selected as l . Therefore, the area of partition one, P_{one} , is simply $\sum_{q=0}^{\theta-1} \text{binopdf}(q, n, 0.5)$. However, the area of partition two, P_{two} , is somewhat complex to analyse. Considering the constraint defined in Section 7.3, some $\ell 1$ -Norms in the second partition have already been used to match $\ell 1$ -Norms in the first partition. For example, at $q = 6$ in Figure 7.7, the probability of $\|\mathbf{f}\|_1 = 6$, labelled (A), can be calculated using the formula in equation (7.4) by setting $q = 6$. However, some $\ell 1$ -Norms, labelled (C), have already been matched with $\|\mathbf{f}\|_1 = q - \theta$, labelled (D). Given the area of (C) and (D) is $\text{binopdf}(q - \theta, n, 0.5)$, the area of (B) can be calculated by deducting (C) from (A), i.e., $\text{binopdf}(q, n, 0.5) - \text{binopdf}(q - \theta, n, 0.5)$. Nonetheless, the maximum area of (B) is also limited by the available $\ell 1$ -Norms at $\|\mathbf{f}\|_1 = q + \theta$, labelled in (E), implying the smaller of (E) and (B) will be the final red region for $\|\mathbf{f}\|_1 = q = 6$. Based on this analysis, the extraction efficiency η_{Fix} of the Fixed-d method can be formulated as:

$$P_{\text{one}} = \sum_{q=0}^{\theta-1} \text{binopdf}(q, n, 0.5) \quad (7.5)$$

$$P_{\text{two}} = \sum_{q=\theta}^{n-\theta} \min\left(\underbrace{\text{binopdf}(q, n, 0.5) - \text{binopdf}(q - \theta, n, 0.5)}_{\text{(B)=(A)-(C)}}, \underbrace{\text{binopdf}(q + \theta, n, 0.5)}_{\text{(E)}}\right) \quad (7.6)$$

$$\eta_{\text{Fix}} = P_{\text{one}} + P_{\text{two}} \quad (7.7)$$

Variable-d Extraction Efficiency. In Variable-d method, $\ell 1$ -Norms can be considered to be, effectively, selected starting from the two ends of the Binomial distribution of the $\ell 1$ -Norms. Recall, to facilitate our analysis, we employ the theoretical distribution described by equation (7.4). Given that n and θ can be chosen flexibly, we can consider extraction efficiency under four cases for resulting distributions based on whether the chosen values for n and θ are odd or even. We consider the following four cases: a) both n and θ are even numbers; b) n is odd but θ is even; c) n is even but θ is odd; and d) both n and θ are odd numbers. As depicted in Figure 7.8, in an ideal setting, the selected $\ell 1$ -Norms (red and yellow regions) are always symmetrically allocated regardless of the chosen n and θ values. Consequently, the extraction

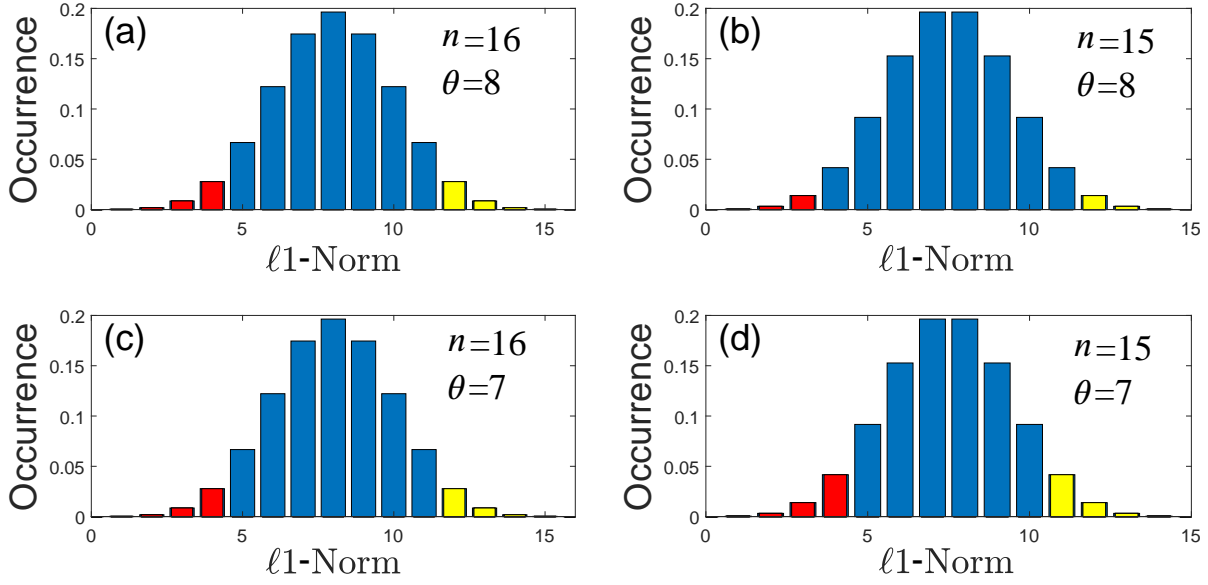


Figure 7.8: Selected area with different parameter combinations: (a) even n and even θ ; (b) odd n and even θ ; (c) even n and odd θ ; (d) odd n and odd θ . We can observe that the selected ℓ_1 -Norms are always sparsely and symmetrically located at two ends of the Binomial distribution histogram.

efficiency for the Variable-d method (η_{Var}) can be concisely formulated using equation (7.8), which simply describes the coloured regions at one end of the Binomial distribution.

$$\eta_{\text{Var}} = \sum_{q=0}^{\lfloor \frac{n-\theta}{2} \rfloor} \text{binopdf}(q, n, 0.5) \quad (7.8)$$

7.5.3 Trial-and-error Complexity

TRE complexity in iterations. To evaluate the complexity of the on-server TRE mechanism, we consider the required number of trials (including one MAC computation and the subsequent MAC tag comparison) rather than the actual time taken, because the actual computation times can depend on many factors such as the server’s hardware specifications, system loads and CPU temperatures. The following equation expresses the upper-bound (worst-case) complexity in iterations, for a TRE procedure conducted over a k -bit fingerprint \mathbf{F} with TRE capability of tolerating x error bits:

$$C(\text{TRE}(k, x)) = \binom{k}{x} \cdot 2^x = (2 \cdot k)^x. \quad (7.9)$$

Here, the term $\binom{k}{x}$ represents the total number of possible positions that x error bits may appear in a k -bit fingerprint \mathbf{F} , and the 2^x term computes the x -bit power set described in Section 7.4.

TRE complexity in execution time. As discussed above, the time overhead of the on-server TRE algorithm may be affected by many factors, such as hardware specification. However, it is beneficial to have an intuitive understanding of the on-server TRE computation time. To calculate the time overhead of our TRE algorithm $\mathcal{T}(\text{TRE}(k, x))$, we employ the following model to estimate the time required to perform the MAC computations.

$$\mathcal{T}(\text{TRE}(k, x)) = \left(C(\text{TRE}(k, x)) \times k \right) \left(\frac{1}{\text{MAC}_{\text{speed}}} \times \frac{1}{N_{\text{GPUcore}}} \times \frac{1}{N_{\text{CPUcore}}} \right) \quad (7.10)$$

Where $C(\text{TRE}(k, x))$ is the upper-bound complexity of TRE algorithm in iterations, k is the length of noise-tolerant fingerprint \mathbf{F} , $\text{MAC}_{\text{speed}}$ ³² is a hardware-specific factor to quantify the MAC computation speed in Mbps³³, the two coefficients N_{GPUcore} and N_{CPUcore} quantify the gains from parallelisation of the iterations across CPU and GPU cores.

TRE complexity growth rate. To characterise the growth rate of the worst-case TRE complexity, we can use the Big- \mathcal{O} notation. Because the noise-tolerant fingerprint size k is a constant, we can now substitute $2 \cdot k$ with the constant symbol c and refer to TRE capability x as problem size factor:

$$\mathcal{O}(\text{TRE}(x)) = \mathcal{O}(c^x) \quad (7.11)$$

The Big- \mathcal{O} notation of TRE method not only helps describe the complexity of the algorithm but also provides a reference for comparing potentially different TRE methods in the future as discussed in more details in Section 8.3.6 in Chapter 8.

7.6 Experimental Evaluations

This section comprehensively evaluates the Fixed-d method and the Variable-d method proposed in this chapter in terms of the following metrics (for detailed definitions, please refer to Section 2.3.2 in Chapter 2 and Section 6.3 in Chapter 6):

- Key failure rate, lower is better (Section 7.6.2).
- Extraction efficiency, higher is better (Section 7.6.3).

³²We obtained a $\text{MAC}_{\text{speed}} = 1,911$ Mbps when tested with OpenSSL 3.0.2 build 15 Mar 2022 using command "openssl speed -mac aes128" on a laptop equipped with i7-10510U @1.8GHz processor, utilising a single CPU thread.

³³Where Mbps stands for 1,000,000 bits per second.

7.6 Experimental Evaluations

- Uniformity (Bias), closer to 0.5 is better (Section 7.6.4).
- Uniqueness, closer to 0.5 is better (Section 7.6.5).
- Bit-aliasing, more bits close the ideal 0.5 is better (Section 7.6.6).
- (On-server) TRE complexity, fewer iterations are better (Section 7.6.7).

7.6.1 Evaluation Approaches

Following a methodology similar to that adopted in Section 6.4.1 in Chapter 6, we consider the following three evaluation approaches.

Predictions (Analytical model). This evaluation uses the analytical models formalised in Section 7.5 to predict key failure rate and extraction efficiency and of the transformed fingerprints as well as the complexity of the on-server TRE mechanism.

Simulations (Synthetic chip model). Evaluating the reliability of the transformed fingerprint demands a massive number of repeated measurements and robust management of data for analysis, as discussed in Section 6.4.1 in Chapter 6. Therefore, we employ data from the simulated measurements using the synthetic chip model detailed in Appendix F.1.1 to determine the key failure rate, the extraction efficiency and the TRE complexity.

Measurements (Physical chips). We employ physical chip measurements for evaluating the quality of fingerprints transformed with new algorithms developed in Section 7.3, because properties such as fingerprint uniformity, uniqueness, and bit-aliasing are affected by fabrication variations not incorporated into the synthetic chip model used for simulations. We employ 25 MSP430FR5969 (20 from the MSP20 dataset introduced in Section 2.3.4 in Chapter 2 and five additional chips measured for this study). The MSP430FR5969 MCU is embedded within the CRFID token studied in Chapter 3, and possessing only 2 KiB of internal SRAM for device fingerprinting, is representative of a highly resource-constrained device. To ensure a fair comparison, we fix key size at $k = 128$ bits (for consistency with our analyses in previous chapters) and group size at $n = 16$ (because the MSP430FR5969 MCU's native memory data bus is 16-bit [163]).

7.6.2 Key Failure Rate

To accurately validate the analytical models derived in Section 7.5 with ample sample size, we first leverage the ideal chip model with the same worst-case $\text{BER}_f = 5.30\%$ when the MSP20

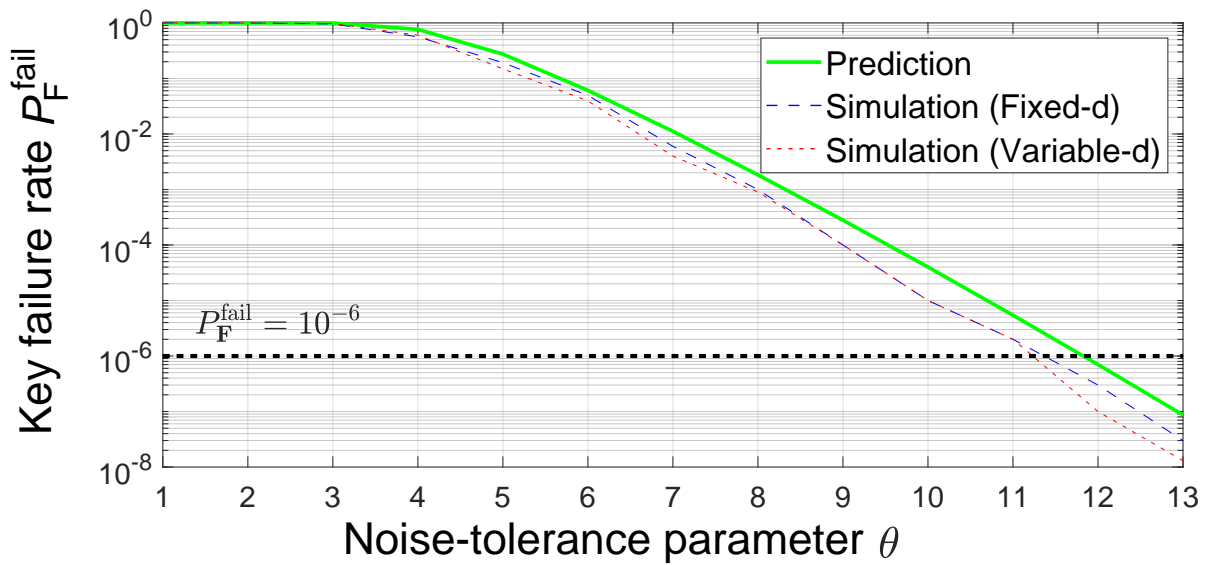


Figure 7.9: To evaluate the key failure rates of the Fixed-d and Variable-d methods, we compare the predictions against simulations produced by the synthetic chip model. Both the predictions and simulations are based on the worst-case $\text{BER}_f = 5.30\%$ from the MSP20 chip dataset measured across the temperature range of 0°C to 40°C to be consistent with the study in Chapter 3. Up to 100,000,000 simulated key re-generations are employed to generate the results for key failure rate. The results corroborate the prediction with equation (6.12) in Chapter 6 as an upper bound of the key failure rate of both the Fixed-d and Variable-d methods. This simulation result also reaffirms the position articulated in Section 7.3: the Variable-d method should demonstrate a lower key failure rate than the Fixed-d method. Notably, the target key failure rate less than or equal to 10^{-6} is achievable with both the Fixed-d and Variable-d methods at $\theta = 12$. Although further simulation results are possible, it would demand exponentially more simulation time and memory usage; hence, we conclude our investigation at $\theta = 13$.

is measured repeatedly within the operational range 0°C to 40°C (as used in Section 3.4.1 in Chapter 3).

The key failure rate is plotted in Figure 7.9. Because the predictions for the Fixed-d method and the worst-case the Variable-d method can be calculated using equation (6.12) in Section 6.5.2 in Chapter 6, they are depicted by the solid green line. Simulations of both the Fixed-d method and the Variable-d method, plotted as blue and red dash lines in Figure 7.9, show that both methods follow the trend of our analytic predictions, with the Variable-d method demonstrating slightly better reliability (i.e., a lower key failure rate). This simulation result also reaffirms the position articulated in Section 7.3: the Variable-d method should demonstrate a lower key failure rate than the Fixed-d method.

7.6.3 Extraction Efficiency

We continue to use the same setting as the key failure rate evaluation, and plot the result in a bar chart. As Figure 7.10 shows, the Baseline method cannot achieve a 64 bit/KiB extraction efficiency (equivalent to extracting a 128-bit device fingerprint from a 2 KiB memory),

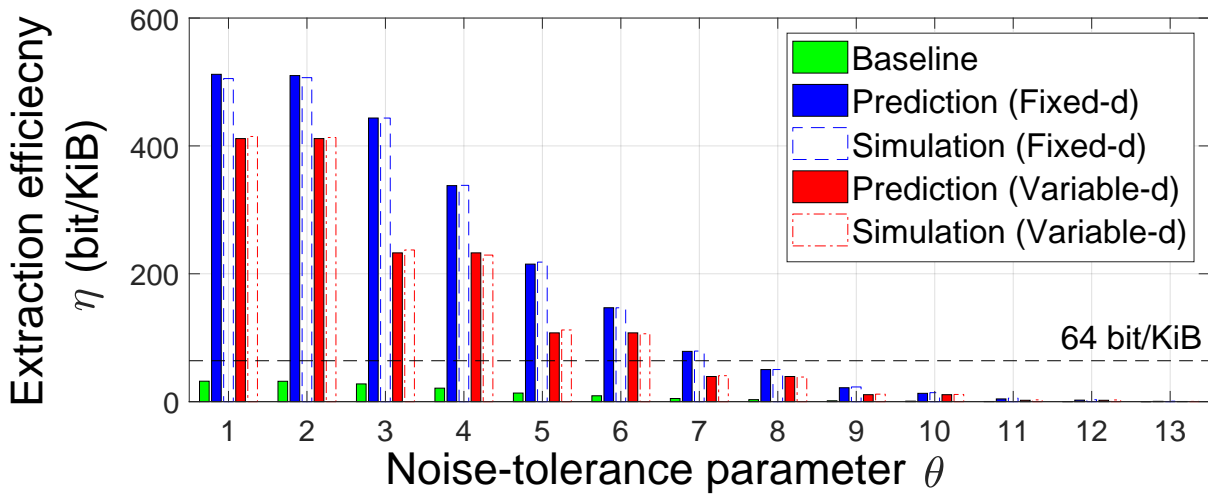


Figure 7.10: Evaluating the extraction efficiency η for different methods involves comparing predictions against empirical results produced by simulations using a synthetic chip model with 2 KiB SRAM. Given the extraction efficiency formulation of the Baseline method has been intensively validated and confirmed in Figure F.3 in Appendix F, here we use only the value predicted by equation (6.11) derived in Section 6.3.2 in Chapter 6. As expected, the Baseline method can not achieve 64 bit/KiB extraction efficiency necessary to generate a 128 bit key from a 2 KiB SRAM memory, regardless of the θ value. Meanwhile, the Fixed-d (predictions from equation (7.7), simulations from using Algorithm 1 on the synthetic chip) and Variable-d (predictions from equation (7.8), simulations from using Algorithm 2 on the synthetic chip) methods both significantly improve extraction efficiency compared to the Baseline. The Fixed-d method outperforms the Variable-d method for all θ values, which reaffirms our position in Section 7.3.2.

regardless of the θ value used. Among the most critical design targets of NoisFre-Lite is optimising the extraction efficiency. The bar chart indicates that both the Fixed-d method and the Variable-d method demonstrates better extraction efficiency than the Baseline method. The Fixed-d method shows better extraction efficiency across all θ values, as predicted in Section 7.3.2. Meanwhile, our derived analytical model can accurately predict the extraction efficiency of the two proposed transformation and selection methods.

In summary, both the Fixed-d method and the Variable-d method can achieve extraction efficiency exceeding 64 bit/KiB. The Fixed-d method performs better across all selection threshold θ values, achieving slightly higher extraction efficiency than the Variable-d method.

7.6.4 Uniformity (Bias) Evaluation

Here, we measure and compare the uniformity (also known as the bias; for more details on the uniformity measure, please refer to Section 2.3.2 in Chapter 2) of the device fingerprints \mathbf{F} extracted using the Baseline, the Fixed-d and the Variable-d methods for the 25 physical chip dataset, as shown in Figure 7.11.

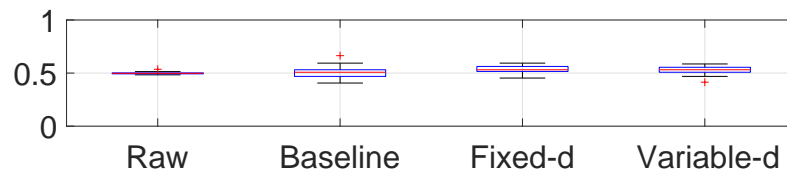


Figure 7.11: Uniformity (bias) evaluation of three methods: Baseline, Fixed-d, and Variable-d. Results from all of the methods closely approximate the ideal value (0.5). Each transformation and selection method is evaluated using the 25 MSP430FR5969 physical chip dataset.

For the 25 physical chips tested, the Baseline method exhibits the ideal mean bias of 0.50, with a standard deviation of 0.06. Neither the Fixed-d method nor the Variable-d method produces any noticeable bias, the mean biases of both methods are around 0.53, with standard deviations below 0.05. In summary, the proposed methods do not lead to a bias in the noise-tolerant fingerprint.

7.6.5 Uniqueness Evaluation

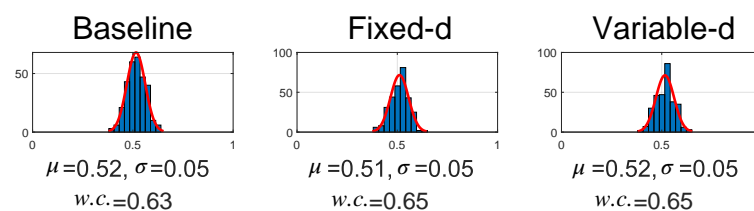


Figure 7.12: Uniqueness performance of the three methods: Baseline, Fixed-d and Variable-d. Each transformation and selection method is evaluated using the 25 MSP430FR5969 physical chip dataset. For each evaluation, the mean value (μ), standard deviation (σ) and worst-case ($w.c.$) are detailed. All three methods maintain a mean value close to 0.5, indicating good and comparable uniqueness performance.

Uniqueness describes a device’s ability to uniquely distinguish itself from a large population. In the context of key generation, the better the uniqueness, the lower the chance of key collision (repetition). For more details on the uniqueness measure, please refer to Section 2.3.2 in Chapter 2. Ideally, the uniqueness should be 0.5. Again, we applied the three methods to the 25 MSP430FR5969 physical chip dataset. The uniqueness evaluations are presented in Figure 7.12. In summary, all three methods maintain mean values close to 0.5, indicating good and comparable uniqueness performance.

7.6.6 Bit-aliasing Evaluation

Bit-aliasing is a common problem for many physical key derivation methods [209]. Bit-aliasing is an undesirable effect, in which case, different hardware instance may produce nearly identical device fingerprints [123]. We examine bit-aliasing by measuring the inter-class bit frequency.

7.6 Experimental Evaluations

The bit-aliasing of k -bit device fingerprints across l hardware instances can be described by equation:

$$\text{Bit-aliasing} = \frac{1}{l} \sum_{i=1}^l F_{i,j} \quad (7.12)$$

Where, $F_{i,j}$ is the j^{th} bit in the \mathbf{F} generated from the i^{th} hardware instance out of a total of l hardware instances. A bit-aliasing close to 0.5 is ideal.

We include the detailed bit-aliasing plot and analysis in Appendix. G.1 and summarise the key statistics in Table 7.2.

Table 7.2: Comparing bit-aliasing between the proposed transformation and selection methods. Data is collected from the 25 MSP430FR5969 described in Section 7.6.1. The mean and standard deviation obtained from the evaluated bit-aliasing results are denoted as μ and σ , respectively. We observe that the Baseline method shows perfect bit-aliasing statistics, while both the Fixed-d and Variable-d show slight deviations from the ideal value.

Method	μ	σ
Baseline	0.50	0.13
Fixed-d	0.53	0.12
Variable-d	0.54	0.13

As Table 7.2 shows, we applied all three methods to the 25 physical chips datasets. For each of the methods, we present the aliasing statistics, which describe the overall bit aliasing across the physical chip population. The aliasing statistics report two quantities: the mean μ and the standard deviation σ . The mean indicates the overall ‘0’/‘1’ preference of all key bits, which effectively resembles bit bias, as detailed in Section 7.6.4. A μ close to 0.5 is ideal. Standard deviation describes the extent of the differences in bit aliasing results across the key bits.

In summary, the mean bit-aliasing values of the proposed methods are slightly deviated from the expected, ideal value. This can be explained from the detailed results in Appendix G, where for instance, the Fixed-d method records a worst-case value of 1.00 for bit-aliasing (see Figure G.2 in Appendix G); this indicates that, for all 25 chips, the corresponding F bit is ‘1’. However, a designer may consider excluding such bits from keys to avoid entropy leakage.

7.6.7 TRE Complexity

Although the TRE mechanism is executed by the resource-rich server, it remains important to maintain TRE execution complexity at a manageable level. Here, we evaluate the

time complexity associated with incorporating the TRE procedure into the Fixed-d and Variable-d methods. Notably, one TRE execution comprises multiple iterations, with one iteration defined as one MAC tag computation and one corresponding MAC tag comparison.

The Fixed-d method with TRE. According to Figure 7.10, using the Fixed-d method, 7 is the largest θ to extract a 128-bit key from a 2 KiB memory space. The predicted key failure rate is 1.12×10^{-2} and the corresponding simulated key failure rate is 5.96×10^{-3} . Neither immediately meets the 10^{-6} key failure rate requirement. Thus, to further improve the key failure rate, we apply the TRE technique in an attempt to restore unmatched bits in re-generated keys. Using equation (7.3), we can estimate the required TRE capability, the upper bound for the number of error bits x that needs to be corrected under the worst-case operating condition to achieve the desired key failure rate of 10^{-6} . In this context, for an expected key failure rate $P_F^{\text{fail}} = 10^{-6}$ with key bit error rate $\text{BER}_F = 8.76 \times 10^{-5}$, obtained using equation (6.9), the necessary TRE capability is $x = 2$. Therefore, the maximum number of trials (or iterations) associated with applying the TRE method to achieve a key failure rate below 10^{-6} can be estimated using equation (7.11):

$$C(\text{TRE}(128, 2)) = \binom{128}{2} \cdot 2^2 = (2 \cdot 128)^2 = 65,536$$

We further validate the complexity using an extensive empirical evaluation; we perform 1,000,000 key generations and record the number of TRE routine iterations or trials for each key generation. According to the formal models, we would not expect a key failure in the 1,000,000 key generations, and *the maximum number of iterations* should fall below the 65,536 computed previously. Even under the worst-case scenario of 65,536 iterations, the TRE process can be completed within 2.86 microseconds on a laptop with a quad-core CPU@1.8 GHZ and 384-core GPU, with $\text{MAC}_{\text{speed}} = 1,911$ Mbps according to the estimate from equation (7.10).

The results from our extensive simulations are illustrated in Figure 7.13. The Fixed-d method achieves a failure rate of 6.46×10^{-3} , producing 6,458 errors over 1,000,000 repeated key generations. However, the TRE procedure is proven capable of correcting all 6,458 errors with a maximum cost of 7,862 iterations, well below the estimated upper bound of 65,536. Consequently, we did not observe a single key failure after the application of the TRE method. In summary, combining the Fixed-d method with the TRE approach can generate a 128-bit key from 2 KiB of SRAM with a failure rate below 10^{-6} , with up to microseconds on-server latency.

The Variable-d method with TRE. We also evaluated the performance of the Variable-d method after incorporating the TRE mechanism. As Figure 7.10 shows, $\theta = 6$

7.6 Experimental Evaluations

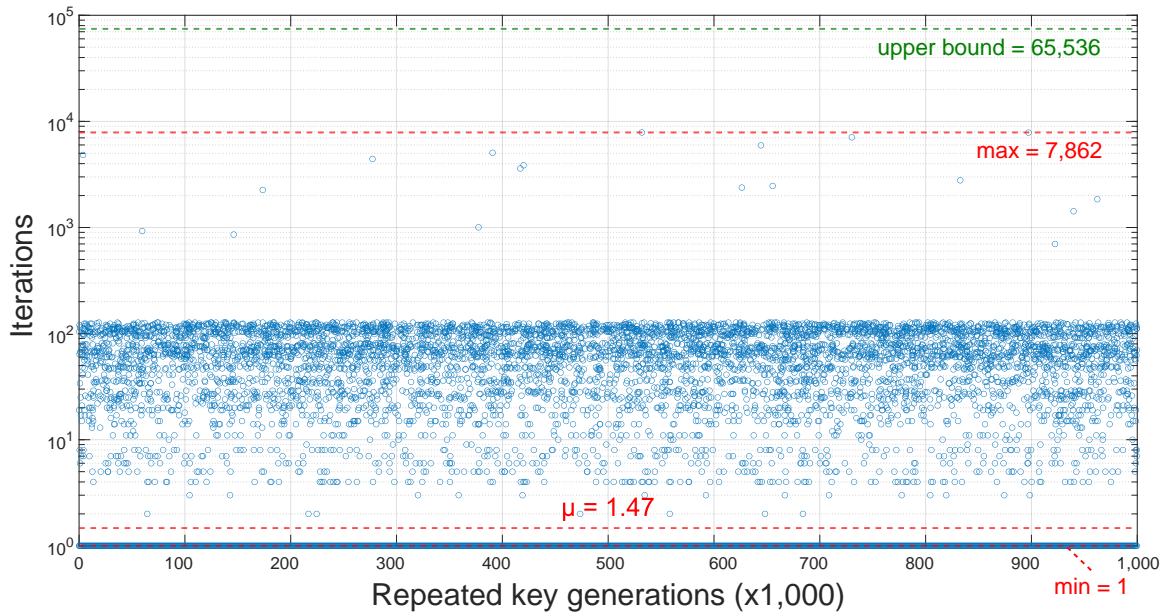


Figure 7.13: Evaluating the TRE mechanism with setting $x = 2$ over 1 million (1,000,000) repeated key generations using the Fixed-d method with $\theta = 7$. In the vast majority of cases, we observe that the TRE procedure only requires a single iteration, suggesting that no error exists in the re-generated noise-tolerant device fingerprint key. On average, key generations require 1.47 TRE iterations. In the worst-case scenario (max), TRE complexity is 7,862 iterations, which remains well below the predicted upper bound of 65,536.

is the largest value the Variable-d method can use to obtain a 128-bit key from a 2 KiB of memory space, with predicted and simulated key failure rates of 6.10×10^{-2} and 3.94×10^{-2} , respectively—as shown in Figure 7.9. Based on $\theta = 6$ and the worst-case $\text{BER}_f = 5.30\%$ we can calculate $\text{BER}_f = 4.92 \times 10^{-4}$ with equation (6.9). Then, equation (7.3) models the expected number of error bits $x = 3$ to be corrected to achieve a key failure rate $< 10^{-6}$. Now, the predicted upper bound for TRE complexity can be calculated as:

$$C(\text{TRE}(128, 3)) = \binom{128}{3} \cdot 2^3 = (2 \cdot 128)^3 = 16,777,216$$

The result presented in Figure 7.14 shows that the vast majority of key generations do not feature error bits. In such cases, the TRE process requires only one iteration. Only 3.40% (33,972 out of 1,000,000) of key generations require the TRE mechanism to intervene. The mean number of TRE iterations is 6.54, and the maximum number of iterations in our tested 1,000,000 key generations is 301,308, almost two order of magnitudes below the estimated upper bound of 16,777,216. According to equation (7.10), a laptop with only a quad-core CPU@1.8 GHz and 384-core GPU can complete the 16,777,216 iterations of TRE computations required under the worst-case condition in 0.73 milliseconds. Therefore, the TRE mechanism can reduce the

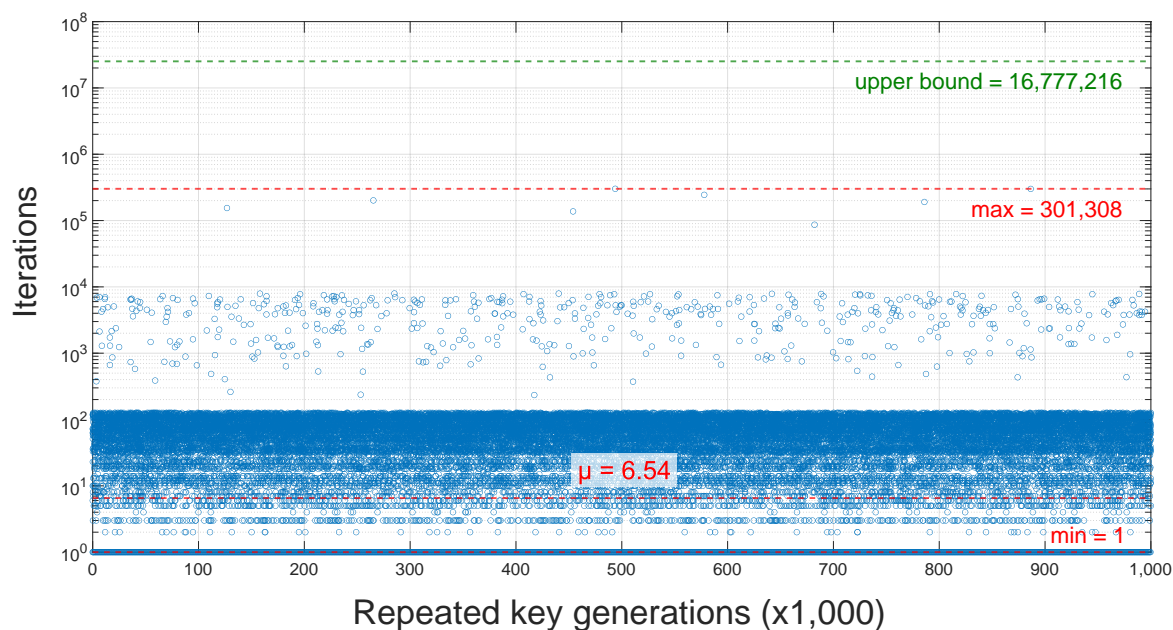


Figure 7.14: Evaluating the TRE approach with the setting $x = 3$ over 1 million (1,000,000) repeated key generations using the Variable-d method with $\theta = 6$. In the vast majority of cases, we observe that the TRE mechanism only requires a single iteration, suggesting that no error exists in the re-generated noise-tolerant device fingerprint key. On average, a key generation requires 6.54 TRE iterations. In the worst-case scenario (max), TRE complexity is 301,308 iterations, more than one order of magnitude below the predicted upper bound of 16,777,216.

key failure rate of the Variable-d key generation method from 3.40% to below 1×10^{-6} with a minimal on-server computational cost.

In summary, we have demonstrated that the TRE mechanism can substantially reduce the key failure rate of NoisFre-Lite by utilising the server's superiority over resources without adding to the execution overhead of the highly resource-limited device. Having demonstrated the efficacy of NoisFre-Lite through the extensive evaluations in this section, the following section will: i) present a practical use case of NoisFre-Lite by considering mechanisms for its application in the derivation of a root key for security functions; and ii) evaluate and compare the implementation cost of a NoisFre-Lite key generation method on a highly resource-limited devices.

7.7 Key Derivation for Security Functions

Here we contemplate a key derivation protocol to demonstrate a use case of the proposed NoisFre-Lite method. The overview of the NoisFre-Lite design presented in Figure 7.1 (d) shows that for a key generator, only a simple fingerprint transformation is needed on the device

7.7 Key Derivation for Security Functions

side to transform \mathbf{f}' into \mathbf{F}' , fundamentally removing all other post-processing from the highly resource-limited device, including, for example, the computationally insensitive encoder of the RFE. The server adopts the TRE technique to reconstitute the re-generated \mathbf{F}' from the enrolled \mathbf{F} , even when the reevaluated bits from the in-field device and the enrolment counterpart in the server differ to some degree. Notably, the device-side secure key generation is identical to that in NoisFre (see Figure 6.17 in Chapter 6), meaning NoisFre-Lite imposes no additional implementation overhead on the highly resource-limited device.

7.7.1 A Method for Realising a NoisFre-Lite-Based Key Generator

We adopt a simple challenge-response-based method for a device and a server to agree on the key derived by the device. The overview of the NoisFre-Lite-based key derivation protocol we constructed is presented in Figure 7.15. The protocol comprises of three stages: i) **On-Server Secure Key Enrolment**; ii) **On-device Dynamic Secure Key Generation**; and iii) **On-Server TRE**. We elaborate on these stage below.

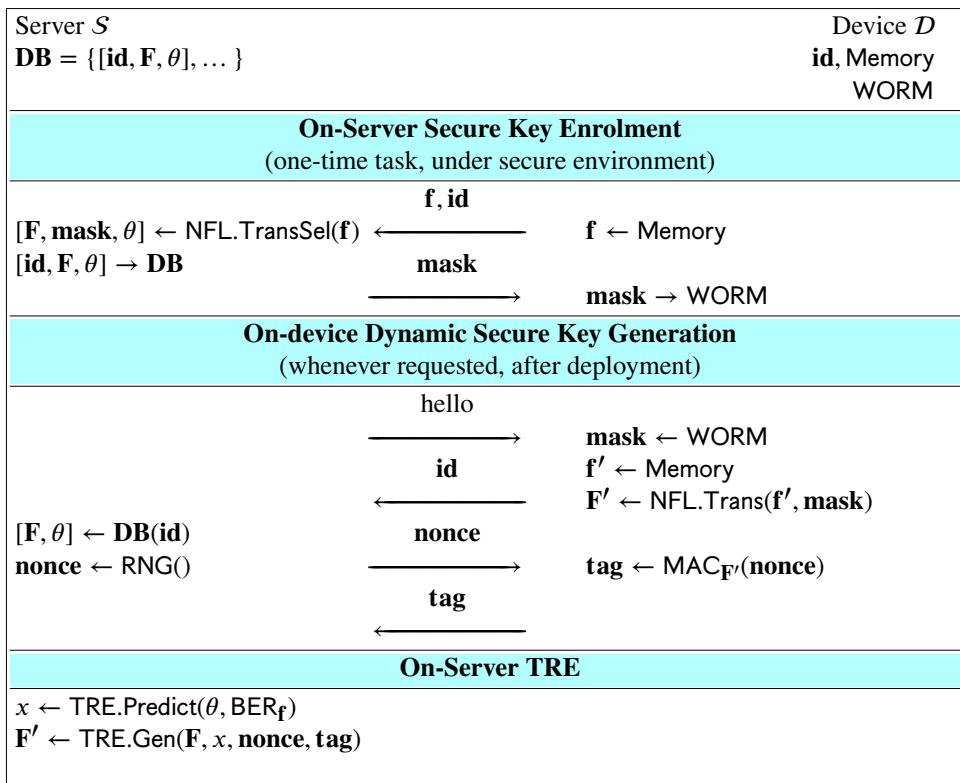


Figure 7.15: The NoisFre-Lite-based key derivation protocol enables lightweight key establishment between a highly resource-limited device and a server. Here, we assume the device implements write-once-read-many memory (WORM), and the \mathbf{mask} is stored in the WORM. In cases where WORM is not supported by the device, the \mathbf{mask} can also be transmitted during the **On-device Dynamic Secure Key Generation** stage, as discussed in Section 6.5.1 in Chapter 6.

On-Server Secure Key Enrolment

This stage is conducted in a secure environment and represents a task that is performed only once across a device's life cycle. The start-up state of the device's SRAM memory (raw fingerprints \mathbf{f}) is extracted, for example, through a wired programming interface, and sent to the server. The raw device memory fingerprint \mathbf{f} is fed into the NoisFre-Lite transformation and selection function, $\text{NFL.TransSel}()$ —as described by Fixed-d and Variable-d Algorithms in Section 7.3 to acquire a noise-tolerant fingerprint \mathbf{F} and a noise-tolerant parameter θ —and the corresponding memory addresses are employed to generate the **mask**.

The triplet comprising \mathbf{id} , \mathbf{F} and θ is inserted into the server's database \mathbf{DB} and indexed with the device's identification number \mathbf{id} . The **mask** is transferred to the device and stored in the device's WORM. Alternatively, the **mask** can also be transmitted during the **On-device Dynamic Secure Key Generation** stage if WORM is not supported by the device, as discussed in Section 6.5.1 in Chapter 6.

On-device Dynamic Secure Key Generation

During this stage, key establishment can be requested at any time after the device is deployed. First, the server sends 'hello' to the device and the device replies with its identification number \mathbf{id} . Meanwhile, the device requests a new noisy raw fingerprint \mathbf{f}' from its internal memory's fingerprint zone $\mathbf{f}' \leftarrow \text{Memory}$. The \mathbf{f}' may differ from the enrolled \mathbf{f} due to noise. The on-device NoisFre-Lite transformation function $\text{NFL.Trans}()$ (identical to the function $\text{NoisFre.Transform}()$ defined in Section 6.5.1 in Chapter 6), produces a \mathbf{F}' from the re-generated noisy fingerprint \mathbf{f}' defined by the **mask**. Notably, the re-generated noise-tolerant fingerprint \mathbf{F}' need not be exactly the same as the \mathbf{F} enrolled at the server—a minor error ϵ may exist.

In this example, the **mask** is stored in the device's WORM memory. In the event that the WORM memory is unavailable, the **mask** can also be transmitted during the **On-device Dynamic Secure Key Generation** stage (as discussed in Section 6.5.1 in Chapter 6).

On reception of the device's response (containing device's \mathbf{id}), the server retrieves the enrolled \mathbf{F} and θ from the \mathbf{DB} indexed with the received \mathbf{id} , before producing a **nonce** using a random number generator, $\text{RNG}()$. Subsequently, the **nonce** is transferred to the device. The device employs the noise-tolerant fingerprint \mathbf{F}' as the key in a message authentication code $\text{MAC}()$ generated over the **nonce** received from the server to produce the **tag** transmitted back to the server.

On-Server TRE

The subsequent on-server TRE phase reconciles the error bit vector ϵ within the re-generated \mathbf{F}' , which distinguishes it from the enrolled version \mathbf{F} and precludes its direct use as a device key. The server determines the necessary TRE capability x (the expected number of error bits in \mathbf{F}') by using the TRE.Predict() function. TRE.Predict() function performs two tasks: i) calculate $\text{BER}_{\mathbf{F}}$ according to the enrolled θ and the worst-case $\text{BER}_{\mathbf{f}}$ using equation (6.9) derived in Chapter 6; and ii) employ analytical equation (7.3) to calculate the required TRE capability x . The server subsequently re-generates \mathbf{F}' using the TRE generation function TRE.Gen() as elaborated in Algorithm 3.

Having elaborated all of the necessary components for the NoisFre-Lite-based key derivation protocol, we will investigate next, the implementation of NoisFre-Lite on a device representative of a highly resource-constrained setting and evaluate the implementation overhead in comparison to the current state-of-the-art RFE-based method for on-device reliable key derivation.

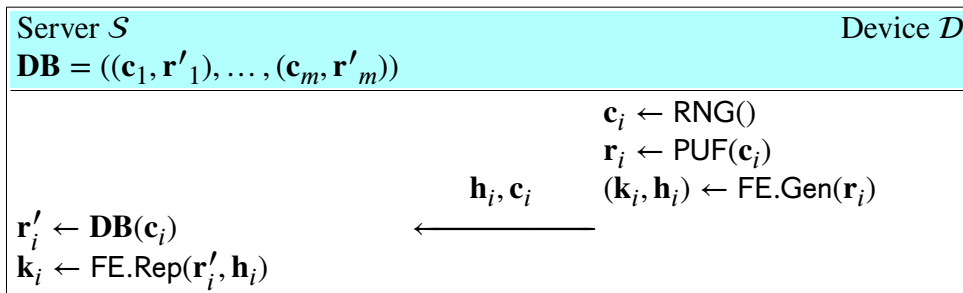


Figure 7.16: The RFE-based lightweight physically obfuscated key derivation method described in Figure 3.2 in Chapter 3. Here, we removed functions not related to key derivation. The RNG() is an on-device random number generator. The function PUF() symbolise the SRAM PUF that takes a challenge \mathbf{c}_i (memory address) as input and reacts with a corresponding response \mathbf{r}_i (SRAM start-up value) as output. The RFE encoding function (or the helper data generation function) is denoted by FE.Gen() which derives a PUF key \mathbf{k}_i and corresponding helper data \mathbf{h}_i . Both the challenge \mathbf{c}_i and the helper data \mathbf{h}_i are transferred to the server S . The server S accesses the enrolled response \mathbf{r}'_i from its database \mathbf{DB} , and the PUF key \mathbf{k}_i is reconstructed based on \mathbf{r}'_i and \mathbf{h}_i through RFE's reproduction function FE.Rep().

7.7.2 On-device Implementation Overhead and Comparisons

To demonstrate the real-world practicality of the proposed NoisFre-Lite-based key generation method, we compare with the lightweight physically obfuscated key derivation mechanism based on an RFE and employed in SecuCode in Chapter 3 to secure the firmware update scheme for *battery-free CRFID devices* [38]. The key derivation process employed therein, built upon a BCH code-based encoding block, is depicted in Figure 7.16.

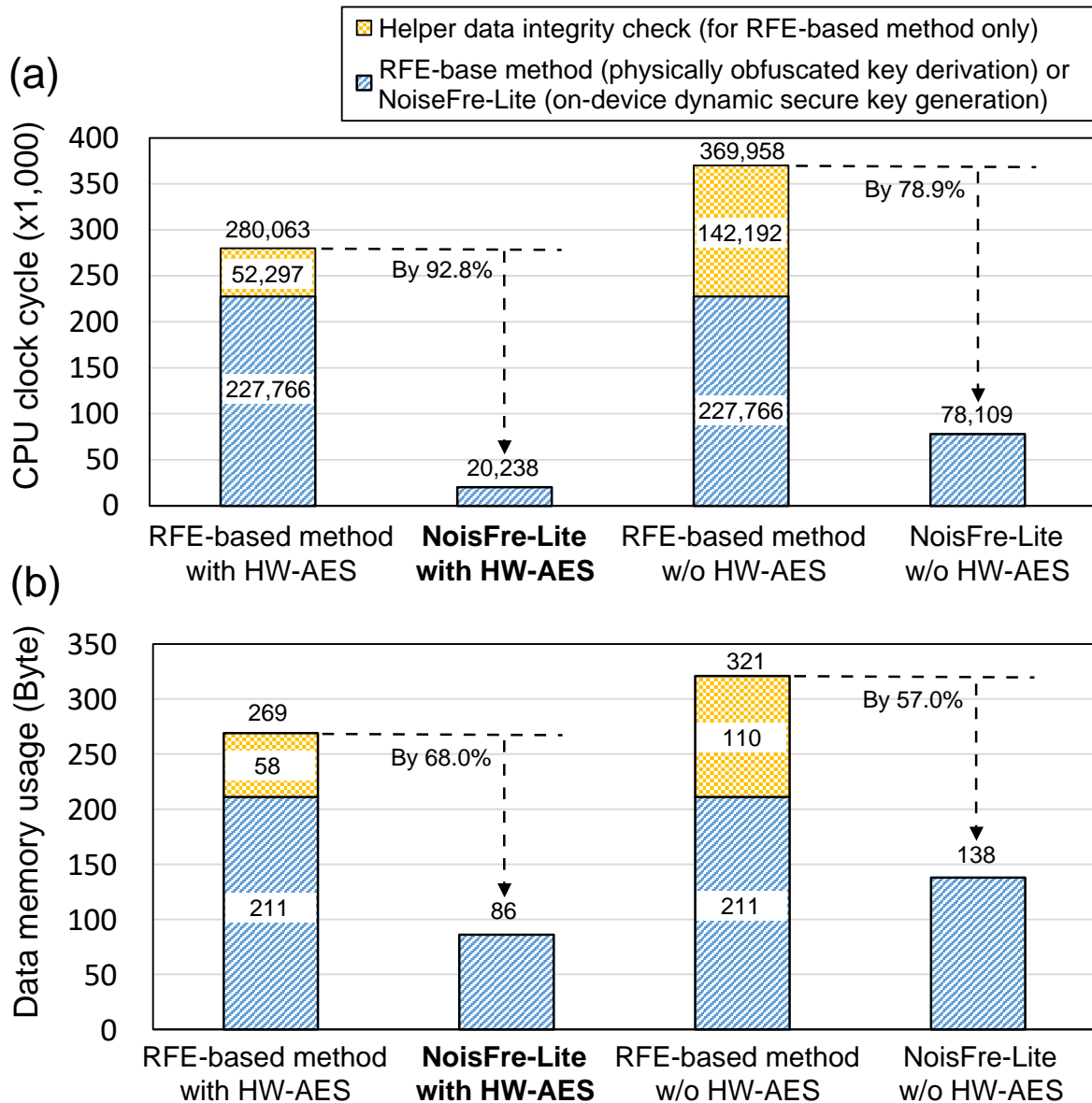


Figure 7.17: A comparison between the implementation overhead of an RFE-based key generator following the lightweight physically obfuscated key derivation method in Chapter 3 and NoisFre-Lite . We consider two situations: i) using a hardware accelerated HWAES-CMAC implementation built with an on-chip AES accelerator (HW-AES); and ii) using a software BLAKE2s implementation of the MAC function in the event that a hardware AES accelerator is unavailable (w/o HW-AES). Overhead is evaluated based on: (a) CPU clock cycles; and (b) data memory (SRAM). Results are acquired from an MSP430FR5969 embedded platform using the Code Composer Studio (CCS) 10.0.0 development environment, with TI v20.2.0.LTS compiler using Optimisation settings: `-O = 3; -opt_for_speed = 5`.

We adopt the methods employed for the lightweight physically obfuscated key derivation method in SecuCode to build an RFE-based key generator. Thus, we employ an encoder comprising of 19 parallel blocks of BCH code parameterised by $(n=63, k=7, t=15)$ and employ a $BER_f = 5.30\%$ within the legal temperature range of $0\text{ }^\circ\text{C}$ to $40\text{ }^\circ\text{C}$ for the target MSP430FR5969 MCU (for

7.7 Key Derivation for Security Functions

more details please refer to Section 3.2.2 in Chapter 3) to generate a 128-bit key with a key failure rate of 2.35×10^{-6} (calculated using equation (5.3) described in Chapter 5). Notably, the proposed method avoids the need for performing helper data integrity checks, although the helper data \mathbf{h}_i is sent over an insecure channel as discussed in Section 3.4.5 in Chapter 3. For generality, we can adopt methods described in [73] to ensure the integrity of the helper data for the RFE-based key derivation method. Hence we also compare with the addition of helper data integrity checks for the RFE-based implementation.

We assume in NoisFre-Lite, the **mask** is stored in device D 's WORM memory and cannot be modified after deployment. Notably, in Section 4.3.1 in Chapter 4 we discuss a number of methods for realising immutable memory regions that can be employed for ensuring the **mask** is not altered. However, NoisFre-Lite still requires an on-device MAC function to provide important information for the on-server TRE as detailed in Figure 7.15. Similar to most modern MCUs, the very low cost MSP430FR5969 MCU we studied is equipped with a hardware AES accelerator. Therefore, to implement the on-device MAC() we use the HWAES-CMAC developed in SecuCode (for more details, please refer to Section 3.3.4 in Chapter 3). But, other platforms may not be equipped with a hardware AES accelerator (e.g., the ATmega328p, widely used in transport ticketing systems [235] and smart cards [236]). Therefore, we also consider and evaluate a software implementation of the lightweight cryptographic hash function BLAKE2s [237]—for further options, comparisons and benchmarks for software hash implementations please refer to [238].

The results of the overhead evaluation are reported in Figure 7.17. Although our main focus is the CPU clock cycles consumed by the key derivation processes, we also consider the run-time data memory (SRAM) usage and discuss the results below.

Clock Cycles. As depicted in Figure 7.17 (a), the NoisFre-Lite only consumes 20,238 clock cycles to produce a 128-bit key to meet the 10^{-6} key failure rate requirement supported by the on-server TRE mechanism as detailed in Section 7.4. In contrast, the state-of-the-art *lightweight* RFE-based method requires 280,063 clock cycles (including 227,766 clock cycles to generate a 128-bit key with 2.35×10^{-6} key failure rate, plus 52,297 clock cycles for the helper data integrity check). Thus, the proposed NoisFre-Lite method provides similar key reliability but with a 92.8% reduction in the required CPU clock cycles.

Even without the availability of a hardware AES accelerator, using the software implementation of the BLAKE2s, NoisFre-Lite (w/o HW-AES) consumes only 78,109 clock cycles; this is still a 78.9% reduction over the RFE-based method which consumes 369,958 clock cycles.

Data Memory (SRAM) Usage. As shown in Figure 7.17 (b), the data memory (for run-time data storage) usage for NoisFre-Lite (with HW-AES) is 86 bytes, which is 183 bytes (or 68.0%) less than the 269 bytes consumed by the RFE-based lightweight method (211 bytes for physically obfuscated key derivation and 58 bytes for helper data integrity check). In addition, when using the software implementation of BLAKE2s, the data memory usage of NoisFre-Lite (w/o HW-AES) is 138 bytes, a decrease of 57.0% compared to the 321 bytes needed for the RFE-based method (consisting of 211 bytes for physically obfuscated key derivation and 110 bytes for checking the integrity of helper data).

Summary

In summary, the proposed NoisFre-Lite-based key generator achieves phenomenal improvements to the computational overhead in terms of clock cycles, with fewer clock cycles reducing the possibility of power loss for intermittently powered devices (e.g., CRFID tokens). As an additional benefit, the reduced data memory occupation releases more system resources for other on-device applications. However, despite the NoisFre-Lite-based key generator's outstanding implementation footprint, it remains unclear just how secure the approach is. Accordingly, the next section investigates associated security issues.

7.7.3 Security Analysis

NoisFre-Lite is built upon the transformation concepts employed in NoisFre, proposed in Chapter 6, the assumptions and analysis presented in therein remain relevant. Further, the experimental validations with physical chip datasets confirm fingerprint quality, such as uniqueness, uniformity and bit aliasing as analysed in Section 7.6. Therefore, the security analysis presented in Section 6.7.3 in Chapter 6 is still valid for the NoisFre-Lite approach. Consequently, we only consider attacks involving the TRE mechanism, introduced specifically as part of the NoisFre-Lite approach.

For the reader's convenience, it is worth re-introducing some concepts used in our analysis in Chapter 6. First, we assume the adversary \mathcal{A} cannot directly access the fingerprints of the SRAM memory, a common premise when fingerprints are used to derive keys. Second, we allow the adversary \mathcal{A} to read code and constants (including the **mask**) stored in the non-volatile memory region of the device \mathcal{D} . Third, we allow the adversary \mathcal{A} to eavesdrop on the communication channel, isolate the device \mathcal{D} from the system and conduct a man-in-the-middle attack allowing the adversary to forward tampered information from the server \mathcal{S} to the device \mathcal{D} and vice versa.

Attack Against the TRE Method

As described in Section 7.7.1, the NoisFre-Lite approach relies on the TRE mechanism to generate highly reliable device keys. We recognise that using a TRE-based method will increase the success rate of brute-force attacks.

An adversary \mathcal{A} can attempt to impersonate the device D and try to spoof the server S , by brute-force guessing of \mathbf{F} . We follow the definition in [234] and describe the probability of correctly guessing a k -bit by:

$$P = (\max(b, 1 - b))^k \quad (7.13)$$

In our setting, the b is the bias of the transformed noise-tolerant fingerprint \mathbf{F} . As shown in Section 7.6.4 both the Fixed-d and Variable-d methods evaluated in this chapter show bias, b , values around 0.53. Hence, we can see that when k is sufficiently large, for example $k = 128$ bits, even with a slight bit bias, the brute attack success rate is extremely small, that is approximately $(\frac{1}{2})^{128}$. But, when the TRE technique is applied, x bits in a k -bit noise-tolerant fingerprint could be wrong. Hence, the number of bits to guess are effectively reduced by the adopted TRE capability. Then, the probability of correctly guessing the noise-tolerant key becomes:

$$P = (\max(b, 1 - b))^{k-x} \quad (7.14)$$

To impersonate a device D , an adversary \mathcal{A} must correctly guess $k - x$ bits to fool the server S . For example, suppose $k = 128$ bits and $x = 3$ bits—based on the setting from Variable-d with TRE in Section 7.6.7—to ensure a key failure rate below 10^{-6} . Then, the expected brute-force attack success rate is approximately $(\frac{1}{2})^{125}$. Despite the slight increase in brute-force attack success rate induced by the TRE technology, such an attack still remains computationally infeasible in practice.

7.8 Chapter Summary

This chapter presents NoisFre-Lite for generating cryptographic keys from device memory under two practical constraints that are common in highly resource-constrained devices: i) limited memory size; and ii) limited computing resources. NoisFre-Lite first transforms raw and noisy SRAM fingerprints \mathbf{f} into noise-tolerant fingerprints \mathbf{F} and selects noise-tolerant bits by trading some bit reliability to achieve significantly improved extraction efficiency to satisfy key bit

requirements of security functions. We have evaluated two new transformation and selection algorithms for the task and provided not only systematic formulations but extensive experimental validations.

Subsequently, in use cases, some noise-tolerant fingerprints F bits can remain below the level of reliability required of a highly reliable key generator—to compensate the increased key failure rate due to the reduction of bit reliability—the proposed TRE approach restores the minor errors in F bits that may exist in repeated re-generations. Importantly, the TRE approach only utilises server-side computational power, thus avoiding imposing any computational costs to the device. The overall power of NoisFre-Lite has been demonstrated using a battery-less CRFID device with a 2 KiB SRAM memory by demonstrating that the approach achieves a 92.8% reduction in computational overhead (measured in clock cycles) than the state-of-the-art RFE-based approach to key generation.

The conclusion of this chapter signals the conclusion of this thesis' investigations of outstanding issues identified and the comprehensive proposal of new solutions to all of those problems. The next chapter will briefly review the problems explored and conclude the thesis, highlighting its myriad contributions to the field and outlining possible future research directions emanating from this dissertation.

Chapter 8

Conclusion and Outlook



This chapter concludes the thesis and suggests several fertile future research directions.

8.1 Summary Preamble

This dissertation has considered the provision of security services to and their implementation on resource limited devices exemplified by CRFID devices. A key focus of the thesis has been the provision of a secure key generator to derive a root key for security functions under resource constraints typical of low-end devices, such as those found in the Internet of Things.

8.2 Summary of Original Contributions

The thesis first investigated the challenging problem of realising a secure and wireless code update mechanism compliant with current communication protocols under resource constraints and intermittent power supply without additional hardware components. Those investigations and their findings were presented in Chapter 3 and Chapter 4:

- Chapter 3 introduced SecuCode, a novel secure wireless firmware update scheme for resource constrained and intermittently powered CRFID devices. We generated a dynamic key on-demand that was discarded after use to eliminate the problem of permanent secure key storage in NVM. Our SecuCode protocol only allows authorised parties to perform wireless firmware updates and is standards compliant without requiring hardware modifications. As noted in [62], cryptographic engineering of a protocol must consider the complex environment for physical devices, including noise and energy constraints, performance and the cost of protocol instantiations. To this end, we successfully addressed security and implementation challenges, realised an end-to-end SecuCode implementation on the popular CRFID transponder and extensively evaluated the cost and performance of our actualisation, including an application case study for which complete code and experimental data were made public.
- Chapter 4 proposed and implemented the first secure and simultaneous wireless firmware update for many RF-powered devices to feature remote attestation of code installation. We explored highly limited resource contexts and innovated to resolve security engineering challenges to implement Wisecr. The scheme prevents malicious code injection, IP theft and incomplete code installation threats while remaining compliant with standard hardware and protocols. The performance of Wisecr and comparisons with state-of-the-art approaches via an extensive experimental regime validated the efficacy and practicality of the design, and the end-to-end implementation source code was released to facilitate further improvements by practitioners and the academic community.

Given the lack of security related features and protections on resource limited devices, we then considered the question of exploiting ubiquitously available memory fingerprints for security functions on resource-limited devices, investigating this challenge in Chapter 5, Chapter 6 and Chapter 7:

- Chapter 5 developed the MRR approach to significantly reduce the overheads associated with RFE and FE implementations, proposing MR^3FE and MR^2FE for lightweight key generation. We validated our approach using a class of ultra-low-power MCUs employed by a CRFID transponder (WISP5.1LGR) as an exemplary resource constrained device. Our extensive experimental results and analysis, which included SRAM PUF data from 23 ultra-lower-power MCUs, demonstrated that regardless of the response-enrolment approach, (R)FE with MRR will always greatly outperform the conventional single-reference-response(R)FE. Enrolling more reference responses under fine-grained operating conditions can further reduce a token's overhead, especially in the MR^3FE case, where the overhead is independent of the number of enrolled MRR. The proposed MRR applies to not only the studied SRAM PUF but also other silicon PUF types.
- Chapter 6 exploited the ubiquitously embedded memory within commodity computing devices. The proposed NoisFre approach constructively extracts the transformed memory fingerprints embodied with a high tolerance to noise affecting fingerprint generation. With a simple, single, one-off fingerprint enrolment measurement, NoisFre can judiciously identify highly reliable transformed fingerprints serving as hardware root keys or trust roots to directly support various security functions for a wide range of COTS electronic devices. Besides formalising two specific S-Norm and D-Norm fingerprint transformation methods and introducing extensive empirical validations of SRAM, Flash and EEPROM memory using 119 total physical chips, we have conducted a case study featuring the end-to-end implementation of a remote attestation security service employing NoisFre fingerprints to significantly reduce overheads in comparison to the state-of-the-art RFE method for constructing reliable fingerprints for key generation. We also demonstrate how SRAM fingerprints can be generated at run-time by utilising individual memory-bank power control features on MCUs. Ultimately, NoisFre is a *simple but practical* method, especially for existing low-end commodity electronic devices.
- Chapter 7 proposed NoisFre-Lite, which can generate cryptographic keys from SRAM memory under the constraints of limited memory and limited computing resources

8.3 Future Research Opportunities

common to many low-end devices. NoisFre-Lite transforms raw SRAM fingerprints into new noise-tolerant fingerprints \mathbf{F} with improved extraction efficiency whilst sacrificing some bit reliability. We propose two new transformation and selection methods for the task. Consequently, several \mathbf{F} bits may remain erroneous upon re-generations. But, the TRE approach proposed constructively restores the mismatched F bits. TRE is undertaken by the server, with abundant computational resources, and, thus, avoids adding any computation costs to the device. We propose formal models for the two new transformation and selection methods to support their use in practice and conduct extensive experimental validations. The chapter documented the overall capability and efficacy of NoisFre-Lite by an implementation to derive a 128-bit key with a key failure rate $< 10^{-6}$ on a battery-less CRFID device with only 2 KiB SRAM memory. The overhead assessment demonstrated a 92.8% reduction of computation overhead (measured in CPU clock cycles) compared to state-of-the-art reverse fuzzy extractor-based methods.

8.3 Future Research Opportunities

During the time spent pursuing the PhD, numerous exciting notions presented themselves. Time and resource limitations precluded these from developing into mature thesis chapters. This subsection documents these possibilities for the benefit of my personal future career aspirations and to guide other interested researchers.

8.3.1 Fingerprinting Emerging COTS Device Memory Types

- Ferroelectric Random Access Memory (FRAM) already exists in numerous COTS devices, including the MSP430FR5969 MCU that powers the CRFID device in Chapter 3. The datasheet for the MSP430FR5969 MCU [163] indicates minimal write time to the MCU's internal FRAM. If the main clock exceeds 8 MHz, the firmware code should include a wait state, a requirement suggesting a non-instantaneous FRAM write operation. Failing to meet the rated write time may produce a partial programmed state. Our initial experiments were encouraging and demonstrated that chips with FRAM memory show different behaviour when the rated write time was violated. Thus, investigating FRAM based key generators is a worthy future research direction.
- When collecting the memory PUF dataset for Chapter 6, we observed an interesting memory type. In late 2008, Cypress launched the world's first commercial nvSRAM. The

nvSRAM represents an instance of non-volatile random access memory (NVRAM). As observed in Chapter 2, SRAM supports fast and random access, allowing one to directly read and overwrite any bit in the SRAM array without the demands of pre-charge or block erasure. However, SRAM requires a stable power supply to maintain the state. Upon powering down, it will lose its memory contents. Cypress developed nvSRAM by adding one QuantumTrap to each SRAM cell. Upon losing power, a backup capacitor keeps the nvSRAM live for a short time period, and the nvSRAM immediately copies all of the data in its SRAM cells to the QuantumTrap. When power is restored, data are copied back from the QuantumTrap to the SRAM array, with all procedures handled completely automatically by the nvSRAM's internal controller. The QuantumTrap implements a silicon oxide nitride oxide silicon (SONOS) floating gate transistor, according to the datasheet. The data backup typically takes 8 ms. If no capacitor is connected, the data will be corrupted.

We believe it is possible to construct a key generator from fingerprinting the nvSRAM using a backup capacitor with attributes below the recommended level, which produces a partial programmed state. Given QuantumTraps are expected to have different program latency, when applying the same programming time, some may get programmed and some may not, depending on manufacturing variations. Thus, investigating nvSRAM based key generators is a worthy pursuit.

8.3.2 Securing Firmware Broadcasting for Fast Moving Passively Powered Devices

- The study in Chapter 4 is not without limitations. Although Wisecr reasonably assumes (as with the non-secure update methods in *R³* [32] and *Stork* [22]) that the devices are at a fixed distance from a reader antenna during the short firmware update process (approximately 16 seconds in application demonstrations), this assumption may occasionally not hold. For example, if the tags were on a conveyor belt being re-programmed, it would be desirable to avoid delaying the movement of the tags and enable re-programming while distances change. It may be possible to consider the dynamic pilot election mechanism to support such settings. Developing a solution to the problem in the context of resource-limited devices operating over a highly constrained air interface protocol represents a research problem worth investigating in the future.

8.3 Future Research Opportunities

- The problem of broadcast reliability is also noteworthy. One potential avenue involves considering the observers to actively participate during a firmware broadcast. For example, if an observer detects a CRC-16 mismatch, this observer can reply with a negative acknowledgement to request a re-transmission.

8.3.3 Investigating Task Scheduling Mechanisms for Executing Security Functions

- As an initial study, the PAM method was developed based on the assumption that the device consumes power at a constant rate during the active mode. However, different tasks have distinct power consumption profiles, making it worth considering a more adaptive PAM in future work. One potential avenue for future research is the investigation of compiler assisted power consumption characterisations for dynamic power scheduling during run-time.

8.3.4 Investigating The Multiple Reference Concept Beyond Silicon-Based PUFs

- Chapter 5 evaluated the MRR approach using SRAM fingerprints or PUFs. Although Section 5.5.5 discussed its generalisability to silicon-based PUF types, in which the expected bit error rate increases outside of the reference operating conditions, the MRR approach may not provide implementation efficiency for PUF types that do not exhibit this behaviour. We believe that investigating the applicability of MRR concept to other PUF types represents an interesting research question that can be investigated in the future.
- Chapter 5 focused on hard-decision decoding and does not consider the impact of MRR approach on soft-decision decoding. We believe that using the MRR approach with a soft-decision decoding strategy can reduce the implementation overhead. Consider, for example, the following two approaches: i) soft-in-soft-out [114], [196]; and ii) hard-in-soft-out [206]. In these cases, data concerning bit-specific reliability is employed to help identify reliable responses and improve PUF key generator gains. Rather than enrolling bit-specific reliability and the response itself under a single reference operating condition, bit-specific reliability and response can be deployed under multiple reference operating conditions. This demonstrates the potential of the MRR approach to

further lower the failure rate and, consequently, reduce the footprint of the needed key generator implementation.

Given that it is apparent that soft-in-soft-out decision decoding in [114] and decoding strategy based on hard-in-soft-out proposed in [206] are vulnerable to helper data manipulation attacks proposed in [74], future research could probe the extent to which the MRR approach can benefit soft-decision decoding.

- In Chapter 5, we limit the evaluation of the key failure rate P_r^{fail} given in equation (5.4) to $\min\{P_{2j}\}$, $j \in \{1, \dots, J\}$ to demonstrate the MRR's efficacy in a very conservative setting. But, future research could investigate a tighter bound for P_r^{fail} by considering the correlations between the reference responses.

8.3.5 Developing Theoretical Bounds for Transformation and Selection

- In Chapter 7, we developed two new D-Norm-based noise-tolerant fingerprint transformation and selection algorithms and formulated an analytical model for each algorithm to describe its extraction efficiency. Simulations and measurements demonstrated that these new methods outperform the original method (proposed in Chapter 6). However, the potential to further improve extraction efficiency remains an open question. It would be desirable to theoretically prove an upper bound for the highest extraction efficiency possible for a D-Norm transformation and selection method.

8.3.6 Investigating TRE Algorithms with Lower Computational Complexity

- Although Chapter 7 demonstrated that the worst-case TRE complexity is approximately 300,000 iterations (and the worst-case scenario occurs very rarely, twice per million in our tests), demanding only limited effort from a resourceful server, the need to improve scalability make it desirable to further investigate TRE efficiency. One identified problem is that the current TRE method always performs trials on each key-bit, sequentially. One interesting mitigation strategy involves employing Bayesian statistics[239] to consider historic results and assign higher priority to the bits detected to be erroneous to further reduce the computational load on the server across repeated requests.

8.3 Future Research Opportunities

- The TRE developed in Chapter 7 assumes that each bit in the device memory fingerprint is equally reliable. We consider this assumption too conservative for the Variable-d method, in which some F have a higher bit error rate than others. This makes it reasonable to develop a TRE algorithm that allocates greater TRE ‘strength’ to bits with a higher error rate whilst skipping over bits that are highly reliable. Such interesting direction to further develop the concepts in therein are left for future research.

Appendices

Appendix A

Read-Rate Based Dynamic Execution Scheduling for Intermittent RF-Powered Devices

A.1 State of Problem

The power that a CRFID device receives from a radiating RFID reader antenna reduces as distance increases [96]–[98], as illustrated in Figure A.1 (a), due to path loss, shadowing, scattering and radio wave absorbing effects from obstacles [240]. Further, power harvesting devices typically use burst-charge cycles based on a charge pump to deliver packets of energy to power the devices. These bursts become intermittent at larger operating distances. Consequently, these devices operate over a sequence of short-term bursts and work in so-called intermittent powering conditions. Therefore, CRFID type devices may not always be able to support long-run execution of tasks, as shown in Figure A.1 (b).

Recent studies have considered methods for operating under intermittent powering conditions to support the long-run execution of tasks, such as cryptographic algorithms. However, existing methods normally require specific hardware support [13], [162] or introduce additional execution overhead [38], [160]. In this appendix chapter, we investigate the following problem:

- How can CRFID devices make efficient use of harvested power with minimal cost to the device, in terms of hardware requirements and implementation overhead, to achieve long run execution of tasks?

Our main contributions from our efforts to address the above problem are as follows:

- We formally analysed the relationship between the number of successful interrogating rounds (*read-rate*) and the time required for a CRFID device to harvest adequate energy

A.2 Related work

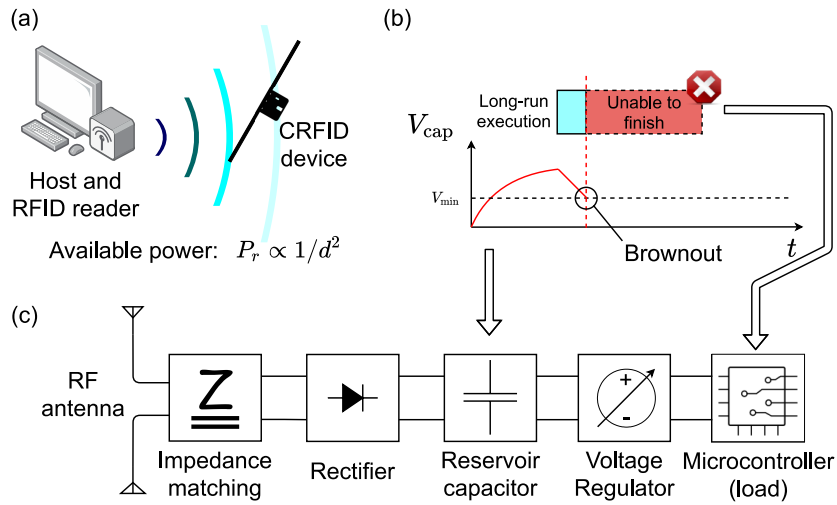


Figure A.1: RF energy propagation, harvest and brownout: (a) Due to the properties of RF energy propagation, at a certain distance away from the reader (b) CRFID devices may be unable to finish the long-run execution of a task due to a brownout event. Here, the voltage across the reservoir capacitor V_{cap} drops below the minimal operational voltage of the MCU V_{min} . (c) Shows the architecture of a typical passively powered device operating on harvested RF energy.

(charge time) and subsequently formulated an analytical expression linking these two quantities.

- To the best of our knowledge, this work is the first to attempt to achieve a reader-driven *dynamic* execution scheduling method for intermittent RF powered devices utilising the simplicity of read-rate measurements and with minimal implementation cost to the device.
- We designed, implemented and evaluated a reader-driven read-rate based dynamic execution scheduling scheme for long-run executions suited for intermittent RF powered devices.

The rest of this appendix chapter is organised as follows: We first review related works in Section A.2, then provide the necessary background knowledge to understand an intermittently powered system in Section A.3. We discuss the principle of our proposed method in Section A.4, followed by the detail design of the *ReaDmE* scheme. We implement and evaluate *ReaDmE* in Section A.5 and Section A.6.

A.2 Related work

The intermittent powering condition for CRFID is first considered in the original WISP design. In [13], Sample *et al.* identified that, the harvested energy may not always be sufficient to

accomplish the desired computational task. To overcome this issue, a dedicated hardware voltage supervisor is employed to monitor the harvested energy level. The voltage supervisor is a comparator with an internal voltage reference V_{ref} . The device's microcontroller unit (MCU) may poll the voltage supervisor before running any power-intensive tasks, such as complex computations or sensor sampling. If the harvested voltage level is below the threshold, the MCU puts itself into a deep sleep state to accumulate energy. Once the voltage level reaches V_{ref} , the voltage supervisor sends an interrupt to wake-up the MCU. The CRFID device can then safely launch the power-intensive task. Since the CRFID wakes up at a fixed voltage, via the hardware voltage supervisor, this method is referred to as *HwFixed* [13]. A limitation of this setup is that it relies on the dedicated hardware voltage supervisor available only on WISP 4.1DL. Notably, this supervisor is absent in the more recent version of the WISP[9].

Buettner *et al.* proposed the Dewdrop [162] execution model to *prevent* a brownout event due to unreliable powering; the method monitors available harvested power and executes tasks only when they are likely to succeed. Dewdrop utilises a dynamic on-device task scheduling method but requires the overhead of collecting samples of the harvester voltage and task scheduling by the device's application code. Further, Dewdrop is only suitable for CRFID devices equipped with a passive charge pump, such as WISP 4.1 [13] since Dewdrop requires an analog to digital converter (ADC) to directly measure the charging rate of the reservoir capacitor (the charge storage element). In the follow-up, WISP version 5.1, the passive charge pump is replaced with an active charge pump (S-882z) and the reservoir capacitor is only connected to the load when V_{cap} developed across the capacitor exceeds the reference voltage of $2.4 \text{ V } V_{\text{ref}}$. Consequently, in WISP 5.1, the voltage delivered to the microcontroller is a sharp step-up, rather than a ramp-up function related to harvested power. Therefore, the charging-up process cannot be directly monitored using the technique in the Dewdrop model.

Su *et al.* [38] proposed the intermittent execution model (IEM) as a part of the SecuCode executable code dissemination scheme. IEM is designed to prevent brownout by fragmenting the power-intensive tasks into small sub-tasks and sandwiching them with low power sleep modes (LPM), thus allowing the CRFID devices to recharge regularly. IEM makes use of the MCU's internal timer to wake the system up from LPM. Although SecuCode is designed to perform firmware updates, securely, the IEM execution schedule is hard-coded as a part of the immutable bootloader. IEM cannot dynamically adjust the schedule once the CRFID device is deployed. Consequently, the fixed-term LPM states could be incurred unnecessarily, with resulting time overheads, despite the availability of a good powering channel.

A.3 Background

Checkpointing is another well-studied technique to handle unpredictable intermittent powering conditions. By regularly saving the run-time data into a device’s non-volatile memory (NVM)[31], [159], [161] or uploading to a server after encryption [160] (i.e., the checkpoint), the system can resume from the last intact checkpoint whenever a power failure occurs. However, writing to NVM is energy intensive [160] and raises security concerns [34], [160]. Off-device checkpointing methods [160] requires coordination between the reader and the device, as well as time and energy overheads (e.g., for data transmissions and encryption functions).

Notably, received signal strength indicator (RSSI) has been investigated in numerous studies [241], [242] for communication channel characterisation, but no studies to date have considered using RSSI for task scheduling.

Summary. The existing methods for dealing with power loss are compared to in Table A.1. In contrast to current approaches, we propose using remote measurements from the transceiver (RFID reader in case of reading and writing to CRFID devices) to characterise the available power at the CRFID device and allow the transceiver to determine, at run time, the dynamic execution schedule for a device.

Table A.1: Comparing with related works.

Study	Scheduling method	Power measurement	Scheduling action	Hardware requirements
WISP 4.1 (HwFixed) [13]	Programmatic	On-device voltage	Pre-task decision	On-device voltage supervisor
Dewdrop[162]	Dynamic	On-device voltage	In-task decision	On-device ADC and passive voltage multiplier
IEM [38]	Programmatic	None	In-task sleep	None
Check-pointing[31], [159]–[161]	None	None	In-task check-pointing	On-device NVM or reader coordination
ReaDmE (<i>this work</i>)	Dynamic	Reader-driven	In-task sleep	None

A.3 Background

In this sections, we briefly describe the background information pertinent to our study.

A.3.1 Typical Energy-harvesting Device Architecture

A generic power supply design for an RF energy harvester architecture is depicted in Figure A.1 (c). The RF antenna is driven by the electromagnetic waves, starting from the left-hand side, and pushes charges into the impedance-matching network. The impedance matching technique maximises energy harvesting efficiency at a specific frequency. The rectifier ensures charges only flow along one direction to produce a direct current (DC). The generated charges are buffered into an energy storage component, normally a reservoir capacitor. The

buffered electricity is regulated to the desired voltage before delivering to the load, such as the MCU of a CRFID device.

A.3.2 Execution Model in HwFixed Scheme

Normally, CRFID devices are under intermittent powering conditions as there is no guarantee that the energy harvester can always supply the load with the required power. Figure A.2 describes the underlying execution model adopted in WISP version 4.1DL and the resulting execution cycles from intermittent powering. On startup, the CRFID device first charges its reservoir capacitor. We denote the time elapsed for the capacitor to be charged as T_c . A conceptual plot of voltage at the reservoir capacitor V_{cap} versus time is illustrated in Figure A.2 (b). Once the voltage at the reservoir capacitor V_{cap} reaches a threshold $V_{charged}$ —see Figure A.2 (b)—the MCU is booted up. The MCU may execute code, sample sensors and prepare the up-link packet. This process may take time T_a to finish, with ‘a’ denoting that the MCU is in active mode. Subsequently, the CRFID enters the ‘Wait for query’ state until the reader instructs it to backscatter the data packet generated earlier. The waiting and backscattering fall within the time frame T_t .

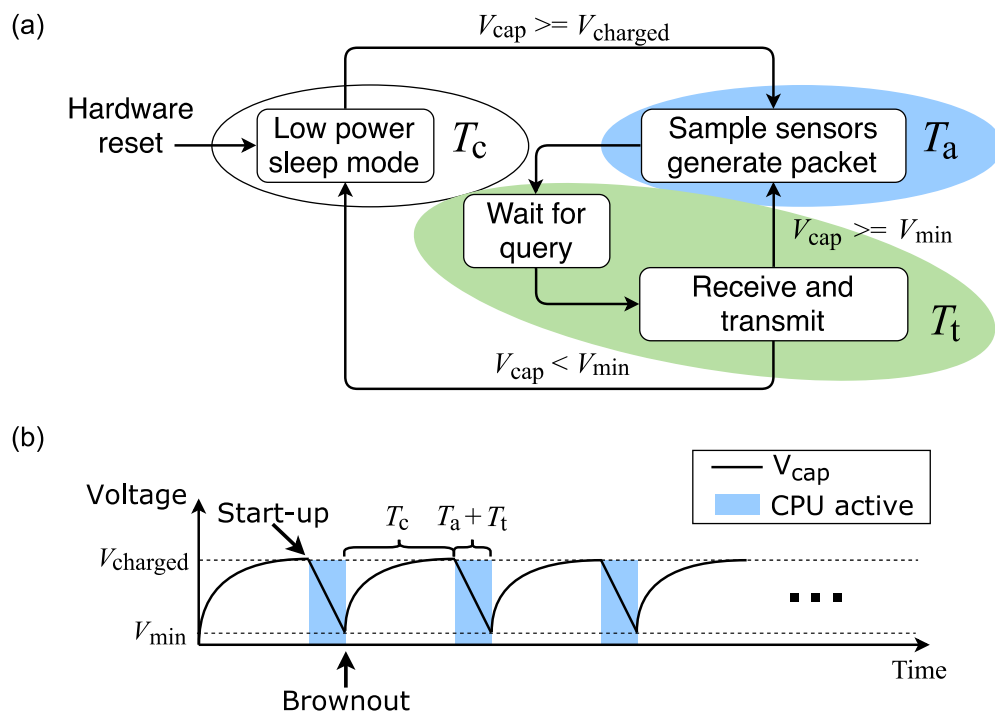


Figure A.2: CRFID (a) Operational power cycle of a CRFID device; and (b) charge burst operational mode.

A.4 Read-Rate Based Dynamic Execution Scheduling Method

Notably, V_{cap} is periodically charged and discharged. If nothing unexpected happens, the period of one charge-discharge cycle, the intermittent power cycle (IPC) following the definition in [38], is dominated by the three previously mentioned time variables: T_c , T_a and T_t .

A.4 Read-Rate Based Dynamic Execution Scheduling Method

We recognise that it is possible for a Server that controls the RFID reader in our setting, to estimate the time to harvest adequate energy at a CRFID device and hence, determine the length of the IPC dynamically. Therefore, in contrast to an a priori chosen intermittent operating setting, as in [38], we propose the dynamic selection of IPC settings by the Server to reduce unnecessary delays in task execution.

A.4.1 Theory and Proof

Observation. The number of successful CRFID device interrogations per second [243], commonly known as the read-rate R_{read} , reduces as the interrogation range d increases.

Proposition. R_{read} can be employed to infer information related to available harvested power by a CRFID device at a given distance. At increasing distances, where harvesting RF power is increasingly difficult, the R_{read} is largely determined by the IPC. We assume the time spent executing code and communicating over the RFID channel is invariant at different powering conditions. Therefore, R_{read} will be dominated by the charging time T_c .

Therefore, we expect a CRFID devices charging time can be expressed as a function f of its read-rate: $T_c = f(R_{\text{read}})$.

Proof. Consider an ideal radio wave propagation environment. The relationship between the transmitted power P_t by an RFID reader antenna and the received power P_r by a CRFID device is expressed by the Friis transmission equation [97], illustrated in equation Equation A.1. The available power at the receiver P_r is determined by transmitted power P_t , transmitter antenna gain G_t , receiver antenna's gain G_r , the radio frequency wavelength λ and the distance from the transmitter to the receiver d .

$$P_r = \frac{P_t G_t G_r \lambda^2}{(4\pi d)^2} \quad (\text{A.1})$$

We can see that the received power P_r is inversely proportional to the square of distance d , assuming all other parameters are constant.

Considering losses at the RF front-end and burst-charge pump, $P_{\text{charge}} = \eta P_r$, where η is the efficiency factor of the energy harvester. Now we can express the energy stored during a charging cycle E_{stored} as:

$$E_{\text{stored}} = \frac{1}{2}C(V_{\text{charged}}^2 - V_{\text{min}}^2) = P_{\text{charge}} \times T_c \quad (\text{A.2})$$

As discussed in Section A.3.2, the length of a charging cycle is T_c , C is the capacitance of the reservoir capacitor, V_{charged} indicates the voltage at the end of the charging cycle when the MCU starts up, V_{min} is the minimum voltage a CRFID device requires for normal operation. Since C , V_{charged} and V_{min} are constants, $T_c \propto \frac{1}{P_r} \propto d^2$. This explains our observation that a CRFID device takes a longer time charge and respond to interrogation at longer operating distances.

During T_c , a CRFID device's power consumption is non-zero, although the MCU is in the LPM; further, the reservoir capacitor has leakage power. We introduce a new term, P_{sleep} , to represent power losses during T_c . On the other hand, P_{charge} would still be present when the MCU is active T_a . Therefore we formulate T_c as in Equation A.3; if the charge power P_{charge} is greater than the system power consumption during LPM P_{sleep} , the charge time T_c is given by the ratio of the energy required to fully charge the storage component E_{storage} over the difference between the charge power P_{charge} and the system power consumption under LPM P_{sleep} . Otherwise, if P_{charge} is lower than the P_{sleep} , voltage across the reservoir capacitor C can never build up, and thus T_c is infinite. The time T_a that the MCU (and thus the CRFID device) is active is given by Equation A.5; the ratio of energy stored over the difference between the system power consumption of active mode P_{active} and the charge power P_{charge} , if P_{charge} is below P_{active} . Otherwise, T_a is the time elapsed to execute the CRFID's application code (case in Equation A.6).

$$T_c = \begin{cases} \frac{E_{\text{stored}}}{P_{\text{charge}} - P_{\text{sleep}}} & , \text{ if } P_{\text{charge}} > P_{\text{sleep}} \\ \infty & , \text{ if } P_{\text{charge}} \leq P_{\text{sleep}} \end{cases} \quad (\text{A.3})$$

$$T_a = \begin{cases} \frac{E_{\text{stored}}}{P_{\text{active}} - P_{\text{charge}}} & , \text{ if } P_{\text{active}} > P_{\text{charge}} \\ T_{\text{execute}} & , \text{ if } P_{\text{active}} \leq P_{\text{charge}} \end{cases} \quad (\text{A.5})$$

As expressed in Equation A.7, the R_{read} is 0 if $P_{\text{charge}} < P_{\text{sleep}}$ as the reservoir capacitor takes an infinite time to be charged. Consequently, the CRFID never responds to an RFID reader's query. Alternatively, the CRFID device may respond at $R_{\text{read}} = \frac{1}{T_c + T_a + T_t}$. That is one closed loop consisting of charge, activation and transmission as illustrated in Figure A.2 (c). For a successful RFID inventory, T_a and T_t will be constant; the only variable is the charging time T_c .

$$R_{\text{read}} = \begin{cases} 0 & , \text{ if } P_{\text{charge}} < P_{\text{sleep}} \\ \frac{1}{T_c + T_a + T_t} & , \text{ if } P_{\text{charge}} \geq P_{\text{sleep}} \end{cases} \quad (\text{A.7})$$

We can easily express T_c as a function of R_{read} , shown in Equation A.9, while assuming $K = T_a + T_t$ is a constant.

$$T_c = (R_{\text{read}})^{-1} - K \quad (\text{A.9})$$

Hence, we can express the charging time T_c as a function of read-rate R_{read} . ■

A.4.2 *ReaDmE* Design

We develop our *ReaDmE* method by building upon the IEM scheme in [38] since it is the only candidate requiring no specific hardware modifications and is realised with the popular CRFID device, WISP 5.1 LRG[9]. IEM is part of the SecuCode bootloader as discussed in Section A.2. Our proposed *ReaDmE* is distinguished from the continuous operation model (CEM) and IEM in Table A.1 and Figure A.3.

As shown in Figure A.3 (a), the CEM tries to finish a long-run execution in a single burst, but it fails due to the brownout, where the reservoir capacitor voltage V_{cap} cannot hold the system's minimal voltage requirement.

The IEM (Figure A.3 (b)) fragments the long-run execution into multiple small sub-tasks and sandwiches them with LPM sleep with a fixed duration. Therefore V_{cap} can be maintained above

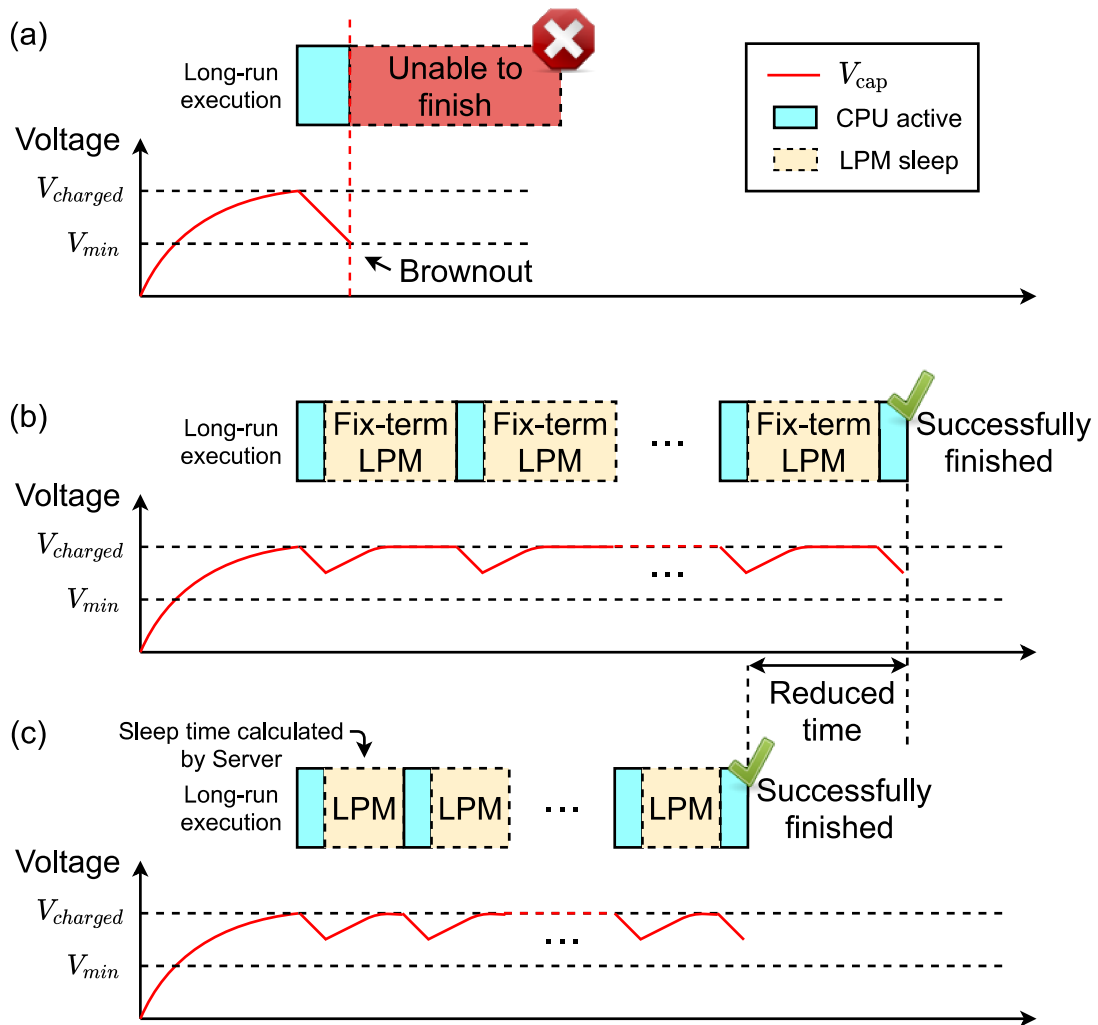


Figure A.3: Comparing: (a) CEM (continuous execution model); (b) IEM (intermittent execution model); and (c) proposed *ReadDmE*.

the V_{min} until the long-run execution successfully finishes. However, since the duration of LPM is fixed, this may incur unnecessary time overhead.

Our proposed *ReadDmE* (Figure A.3 (c)) dynamically adjusts the duration of LPM sleep according to the R_{read} reported by the RFID reader. Also, if the powering condition is extremely poor *ReadDmE* could assign longer LPM sleep time to allow for CRFID recharge.

A.5 *ReadDmE* Implementation

As shown in Figure A.4, we visualise the implementation of *ReadDmE* over the *EPC Gen2* protocol in a sequence diagram. We can identify three stages:

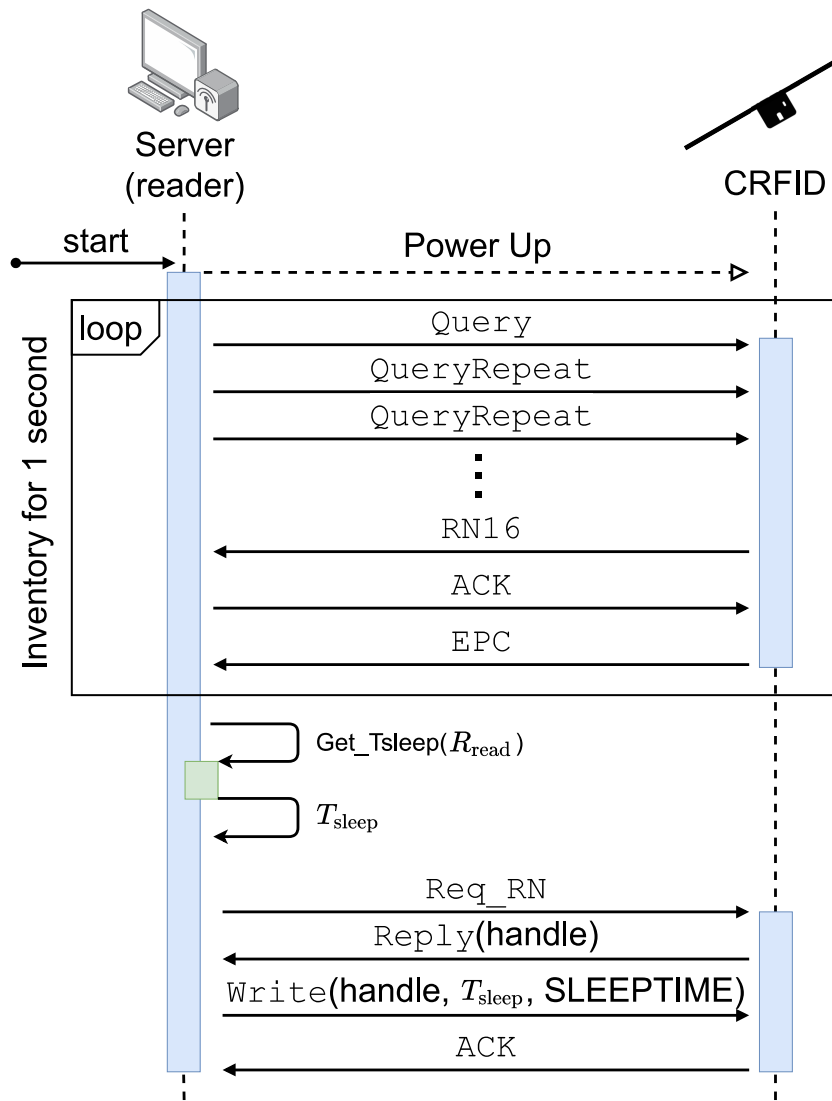


Figure A.4: The implementation of *ReaDmE* over the *EPC Gen2* protocol.

Stage 1. The reader powers up the CRFID device, and counts the R_{read} over a fixed period, for example, one second. This is done by repeatedly performing the *EPC Gen2* Inventory session (as enclosed in the ‘loop’ box in Figure A.4). R_{read} is, therefore, the total number of successful inventories over the time window.

Stage 2. Calculate T_c using Equation A.9. To increase fault tolerance, we let the calculated sleep time $T_{\text{sleep}} = \tau \times T_c$, for example $\tau = 1.1$ gives a 10% margin. In the sequence diagram, we denote this function as $\text{Get_Tsleep}(R_{\text{read}})$.

Stage 3. Calculated T_{sleep} is downloaded to the CRFID device via a standard *EPC Gen2* Write command. On the CRFID device, the received T_{sleep} is stored in a volatile register SLEEPTIME, and the IEM adjusts its LPM according to the received T_{sleep} .

We employed the open source code release from SecuCode [38] at [165] to integrate *ReaDmE* within the IEM model described therein. In order to integrate *ReaDmE*, two minor change needed to be made to the IEM implementation:

- Server shall compute T_{sleep} based on the R_{read} and download it to the CRFID device, via a `Write` command to the SLEEPTIME register.
- The CRFID device shall handle the `Write` command addressing the new SLEEPTIME register and update the IEM settings according to the received data.

A.6 Experimental Validation

We conducted the following three experiments to: i) validate our model regarding the relationship between the R_{read} and T_c (Section A.6.1); ii) show that *ReaDmE* is accurate (Section A.6.2); and iii) practical (Section A.6.3).

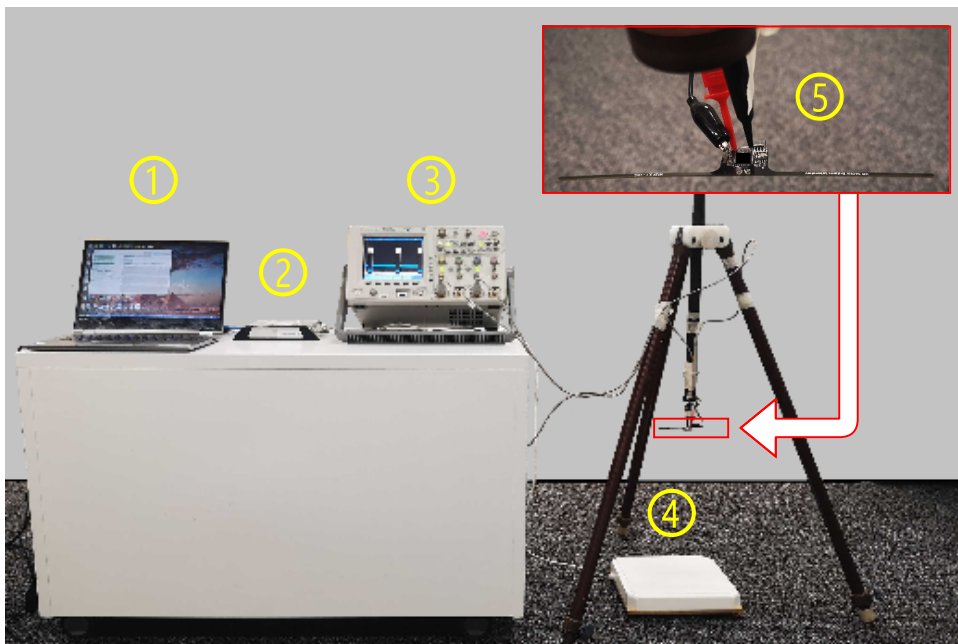


Figure A.5: The experimental setup. ① a host PC running the Server Tool, ② an RFID reader (Impinj R420 RFID), ③ a digital storage oscilloscope (DSO), ④ a reader antenna and ⑤ a CRFID device suspended from a wooden tripod.

All the experiments were conducted using the setup depicted in Figure A.5. We had a laptop execute the Server tool, a modified version of the SecuCode Server tool at [165]) that included the functions to count the read-rate and calculate the suitable T_{sleep} as described in Section A.4.2.

A.6 Experimental Validation

The laptop was connected to an Impinj R420 RFID reader, which drives a 9 dBic panel antenna. The CRFID sensor device being tested was suspended from a height adjustable wooden tripod. Instead of using the Joint Test Action Group (JTAG) debugging interface, we monitored a specific general-purpose input/output (GPIO) ping of the CRFID device with a digital storage oscilloscope (DSO) to measure the internal timing state. Because the JTAG provides power to the target device, it hinders us from investigating the CRFID device's behaviour under a passively powered condition. It also inserts extra debug-related code that interferes with our timing analysis. Hence, we programmed the CRFID device with a special firmware; when the CRFID device switches between its internal states—as shown in Figure A.2 (b)—the firmware flips a specific GPIO pin corresponding to the current machine state.

A.6.1 Validating Our Model

The first experiment was conducted to collect and analyse all quantities mentioned in Section A.4. We sought to identify whether the R_{read} is a meaningful parameter reflecting CRFID device's behaviour, and further validate our theory and assumptions developed in Section A.4.

We recorded the read-rate $R_{\text{read}}(\text{Reader})$ as the successful Inventory sessions per second from the reader side, averaged over 20 seconds. From the CRFID device side, we also measured timing information, such as charge time T_c , active time T_a and transmit time T_t .

The results are presented in Figure A.6, with a white bar representing T_c , a blue bar representing T_a and a green bar representing T_t . We can see a trend representing an exponential increase in T_c , while T_a and T_t remain approximately constant; in accordance with our assumptions described in Section A.4.1.

Using Equation A.8, we can calculate the CRFID device's read-rate $R_{\text{read}}(\text{Device})$, which is plotted with a blue dashed line in Figure A.6. The $R_{\text{read}}(\text{Device})$ is a good approximation of $R_{\text{read}}(\text{Reader})$ —plotted by a red line in Figure A.6. Notably, both the $R_{\text{read}}(\text{Device})$ and $R_{\text{read}}(\text{Reader})$ show a saturation for operation range between 10 and 30 cm. We can explain this observation by recognising that the harvested power at this close operating range is able to support the CRFID transponder to run continuously. Thus, the read-rate is dominated by T_a and T_t . As the operational range increases, from 40 cm—see Figure A.6 (b) for a clearer view— T_c follows an exponential trend. Meanwhile, T_a and T_t remain stable until the operational range reaches 80 cm. At this point, T_t becomes unstable, presumably due to power failures during the RFID backscatter under poor powering conditions.

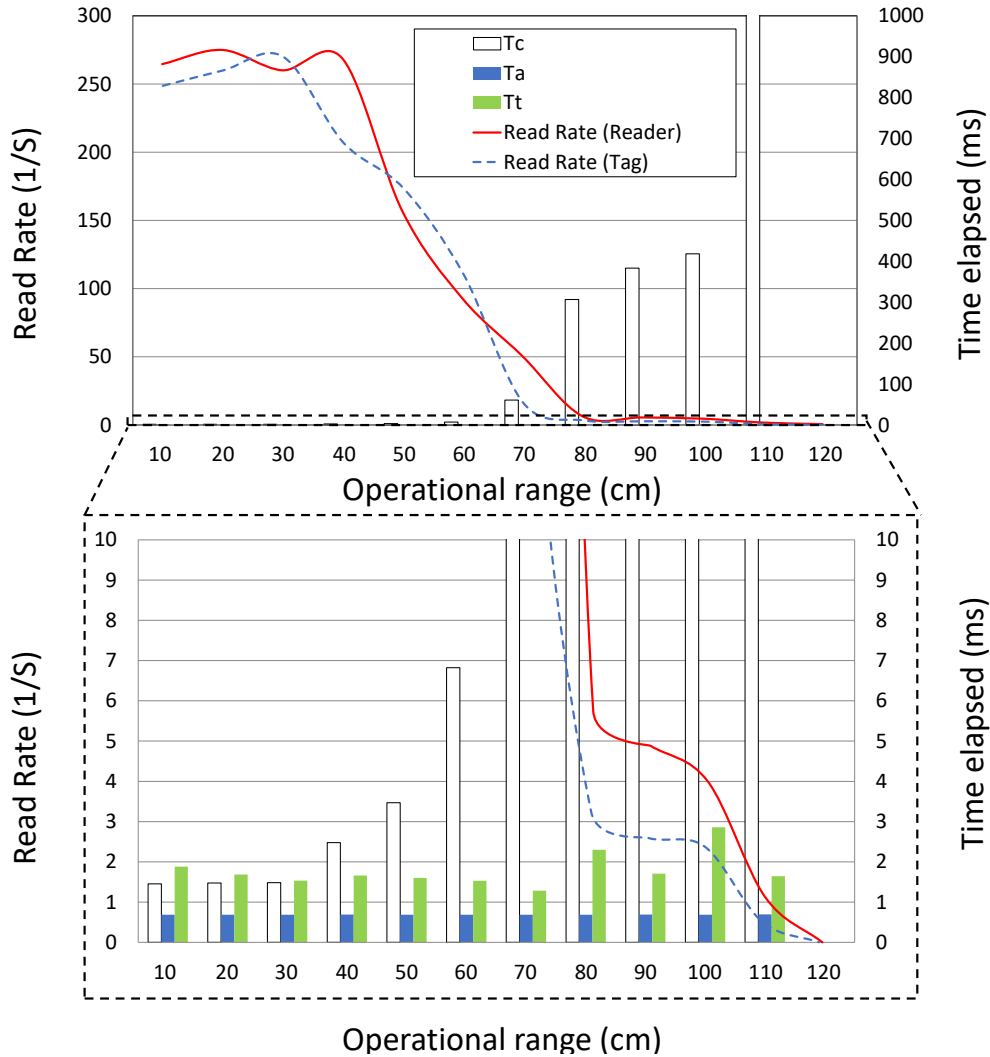


Figure A.6: The time elapsed (right y-axis) for charging up T_c , executing the program T_a , transmitting data T_t . On the left y-axis is the read-rate R_{read} reported from the RFID reader and calculated from on-device measurements using equation (A.8). Results showing: (a) T_c increases as operational range (x-axis) becomes larger; and (b) a magnified view to better illustrate the measurements.

The read-rate reported by the RFID reader closely follows the read-rate calculated by Equation A.8 based on T_c , T_a and T_t measured directly on-device. We can confidently say that our model derived in Section A.4.1 is valid and the read-rate is a reliable reflection of the CRFID device's internal state.

Table A.2: Measured values for Equation (A.9).

	mean (μ)	standard deviation (σ)
CPU active time (T_a)	0.630	0.190
Data transmission time (T_t)	1.641	0.635
$K = T_a + T_t$	2.271	0.792

A.6 Experimental Validation

From this experiment, we can also find the constant K in Equation A.9, as $K = T_a + T_t$. The average over all tested operational ranges from Figure A.7 are summarised in Table A.2. This reaffirms our assumption in Section A.4 that T_a and T_t are invariant over different powering conditions.

A.6.2 The Accuracy of Read-rate-based Charging Time Predictions

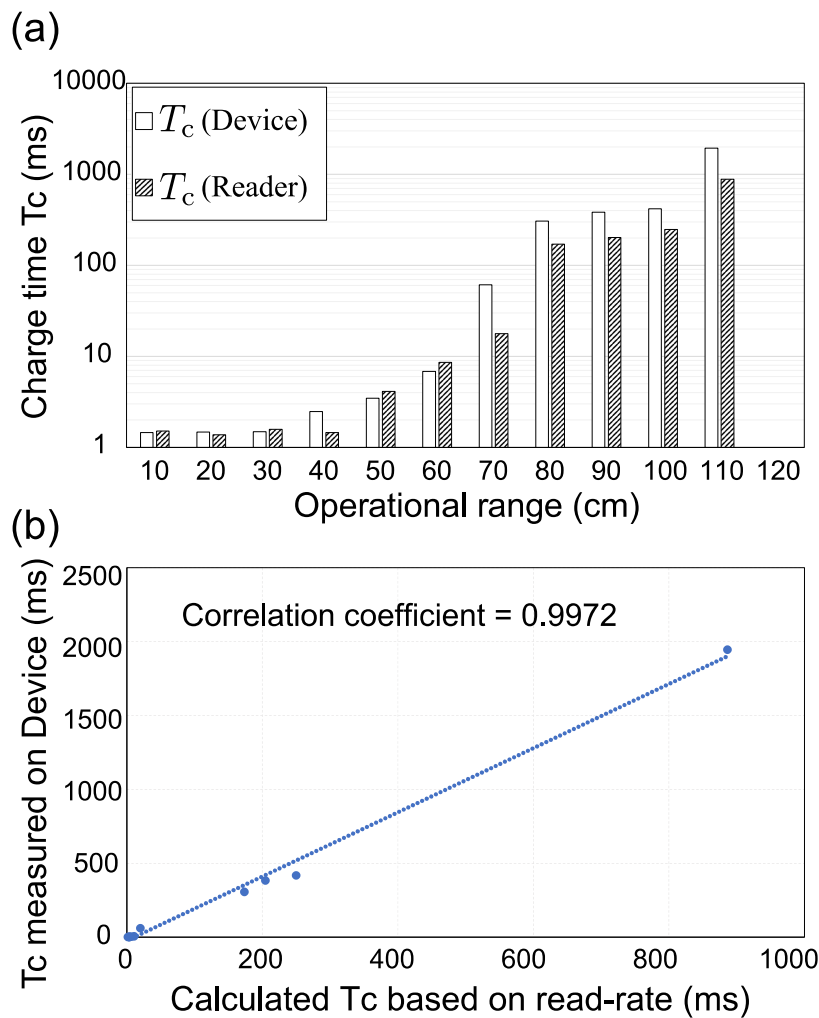


Figure A.7: Comparing: (a) the actual charging time measured from the device T_c (Device) and the charging time calculated based on the R_{read} T_c (Reader) with Equation A.9; and (b) the correlation between the T_c (Device) and the T_c (Reader) evaluations.

From Figure A.7, we can see that the T_c measured directly from the device closely follows that evaluated using Equation A.9 using the R_{read} reported by the RFID reader. However, a systematic analysis is still required to determine whether R_{read} is an accurate measurement of T_c .

To examine the accuracy of the read-rate based powering condition measurement, we apply the Pearson correlation test [244]. We use the T_c measured from the device as one tested variable and use the T_c calculated based on R_{read} as the other tested variable. The result is plotted in Figure A.7 (b). We can see that all measured points are located very close to the trend line. The Pearson correlation coefficient of 0.9972 clearly shows that there is a solid correlation between the T_c directly measured from the device and the T_c calculated based on R_{read} from the RFID reader using Equation A.9.

A.6.3 Application Case Study with a Cryptographic Algorithm

To demonstrate the effectiveness of our *ReaDmE* method, we followed the same setup as in [38] and performed a message authentication code (MAC) over a 1280-byte random string, and compared against CEM and IEM with hard-coded 30 ms T_{sleep} . We repeated each measurement ten times and report the success rate (successfully calculate the MAC tag) and latency (average time taken). Notably, data was only collected when MAC computation commenced, where the DSO was triggered by a GPIO event. Failures in underlying RFID communications were ignored—notably, *ReaDmE* cannot be applied to RFID communications, as the injected LPM can violate the strict RFID communication timing requirements.

The experimental results are summarised in Figure A.8. On the one hand, in terms of latency, *ReaDmE* demonstrated very close performance to the CEM when power is plentiful at 20 cm and 40 cm, while IEM suffered from large unnecessary time overheads due to the hardcoded 30 ms dead weight. On the other hand, *ReaDmE* achieved improved success rate due to the dynamically generated LPM duration based on an accurate assessment of the power available to the CRFID device. It is worth pointing out that *ReaDmE* could achieve a 100% success rate at a 60 cm operational range while IEM fails on 20% of its trials at this range. *ReaDmE* was also 20% more successful than the IEM under the poor powering condition at an 80 cm operational range. Notably, as shown in Figure A.8, the CEM's success rate dropped to 20% at a 60 cm range and to 0% at 80 cm range.

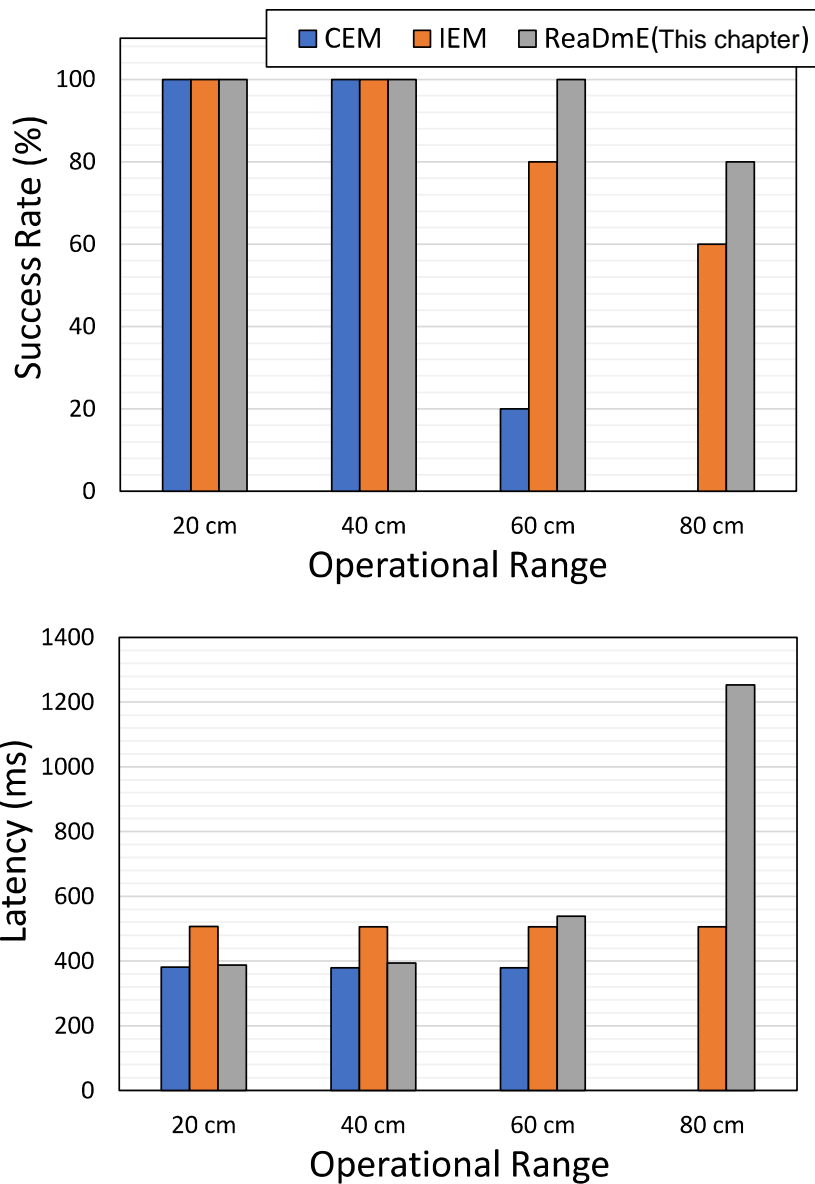


Figure A.8: Experiment results comparing the success rate and the latency across the continuous operational model (CEM), the intermittent operational model (IEM), and our proposed *ReaDmE*.

Appendix B

Memory Bus Snooping and Open Debug Interface Exploits

B.1 State of Problem

A considerable amount of IoT devices have been adopted by consumers and industries over the last decades; the strong growth momentum is set to continue. In 2020, the number of Internet-connected smart devices reached 11.7 billion and is expected to increase by more than five-fold by 2025 [6]. Apart from the popular consumer electronics, health care, and manufacturing, the automotive industry is the next promising market, followed by retail, logistics, agriculture, and animal husbandry sectors [7]. Despite the rapid growth, there are still many challenges related to security, especially for devices involved personal and sensitive data. By 2025, the amount of data gathered by those devices are expected to reach 73.1 zettabytes [6]. The widespread existence and access to sensitive data render Internet-connected devices a lucrative attack target for malicious actors [245].

Despite user privacy concerns, it is difficult to choose products with adequate protections without an in-depth knowledge on security and privacy [246]. Meanwhile, manufacturers have generally focused on reducing time-to-market delays [247], improving cost-effectiveness and quality of service [248] but skipping over some necessary security measures [249].

Our Focus and Contributions. This appendix focus on investigating the vulnerabilities arising from common debug interfaces left open and exposed memory buses in IoT (Internet of Things) devices. Notably, unlike software that can be patched, we focus on exploiting vulnerabilities related to the hardware design that are difficult to be fixed after products are released to the market. To demonstrate the ease with which snooping³⁴ on exposed memory buses and exploiting open debug interfaces can be, we apply techniques that are simple in practice, only require inexpensive gadgets, and do not impose irrecoverable or noticeable damage to the device. In particular, in this appendix chapter, we:

³⁴Snooping: unauthorised access, similar to eavesdropping but not limited to data collection during transmissions.

B.2 Related Attacks Against IoT Devices

- Investigate the practical threat posed by: i) debug interfaces left open; and ii) exposed buses to off-chip memory commonly used for storing data and secrets in COTS (commercial off-the-shelf) devices.
- Demonstrate the relatively low cost and the level of skills required to extract sensitive data from devices.
- Conduct and describe two case studies, in *the style of a tutorial*, to demonstrate the realistic threat by extracting executable code and secrets from COTS electronic devices. A demonstration video of gaining remote access to a portable Internet connected camera after an exploit is available at:

<https://www.youtube.com/watch?v=fnIn9QugrXI>



scan to watch

Organisation. The rest of this appendix chapter is organised as follows. In Section B.2, we summarise previous work and useful resources related to physical attacks on IoT devices. Section B.3 presents the threat model. Exploitable debug interfaces and memory buses are discussed in Section B.4 followed by two case studies using popular IoT devices. Section B.7 provides concluding remarks.

B.2 Related Attacks Against IoT Devices

Software attacks. Attacks against software exploit security vulnerabilities of the communication protocols, cryptographic algorithms, or software implementation of the product [250]–[252]. Software attacks are possible without physical access to the victim device. However, the manufacturer could easily fix the software vulnerabilities through system updates [75], [253]. In this appendix, we would like to focus on exploiting vulnerabilities related to the hardware design that are difficult to be fixed after products are released to the market.

Invasive physical attacks. These attacks require direct access, for example, dissect a smart card and the use of microprobes to read the secure storage of cryptographic keys [254] from its internal components [250], [255]. We consider methods that obviate invasive attacks since it generally requires expensive equipment (e.g., a US\$5,000+ microprobe workstation) and, inevitably, leaves traces of tampering with the device [256].

Non-invasive physical attacks. Vasile et al. [257] summarised three major firmware extraction techniques: i) Debug interfaces; ii) Raw Flash Dump; and iii) Software Methods. The authors

also examined 24 popular COTS smart devices in 2018, and found all of them to be vulnerable to at least one of the techniques. This work also proposed countermeasures for each of the exploits. Although the study is inspiring and covers a wide spread of target devices, the focus of the study was the techniques to dump the firmware from the devices—leading to intellectual property theft. However, exploitation or what useful information the dumped firmware could provide, time and monetary cost of an attack or the articulation of a practical adversary model extracting the firmware and the consequence of having access to that firmware was unclear. Krishnan and Schaumont, in [34] investigated exploiting the JTAG (Joint Test Action Group) interface in an intermittent computing system. They demonstrated the extraction of the AES (Advanced Encryption Standard) secret key from checkpoints stored in on-chip NVM (non-volatile memory). In their adversary model, an active attacker has unrestricted physical access and can corrupt and modify the target device’s memory content.

B.3 Threat Model

We build our threat model on the basic assumption that without technical knowledge, ordinary users would not treat IoT devices without obvious mark of tampering as malicious and warrant further investigation. The attacker’s goal is to extract sensitive information from the IoT device hardware, including but not restricted to the user’s personal information, usage data, device configuration and executable code.

B.3.1 Victim Devices

We classify the victim IoT devices into: i) **Class-I**: The device implements debugging or programming interfaces, and such interfaces were left open in the final product; and ii) **Class-II**: The devices store sensitive data in off-chip NVM, and the memory buses are exposed at the top/bottom layer of the PCB (Printed circuit board) in the final product.

B.3.2 Attacker Capabilities

- Has physical procession to the victim device for a restricted time window—a few minutes to half an hour—such a short time generally does not attract the device owner’s attention. Notably, this setting renders it impractical for physical attacks that require a longer time [255] for specialised laboratory setups and data collection.

B.3 Threat Model

- Has a basic understanding of electronics, computer security, knows how debug interfaces and serial buses function. It is important to highlight that it is easy to acquire the simple knowledge needed from materials freely available online, such as [258] and [259].
- Has access to inexpensive tools as exemplified in Section B.3.3 and a computer with open-source decryption and binary file analysis programs.

Our attacker capabilities are reasoned based on [34], with following changes to consider more generic threats: 1) the duration of attacker's physical access is restricted; 2) an attack does not leave visible traces (e.g., modify the configuration or data) to avoid drawing a user's attention; and 3) we obviated attacks that requires expert knowledge or expensive equipment.

B.3.3 Resources and Costs

We assume an attacker has access to the tools listed in Section B.3.4 to facilitate memory bus snooping and open debug interface exploits. The reference prices were obtained from www.ebay.com. For the attacks we explore, the total cost of a memory bus exploit attack is US\$15, and the cost of an open JTAG interface exploit attack is US\$45. We can observe that an attack under this threat model is inexpensive, and tools can be easily acquired to facilitate memory bus snooping and open debug interface exploits. Notably, an attack under the settings does *not* leave irreversible or visible damage to the device [256] or attempt to modify firmware (as that could permanently brick the device [253], [260]).

B.3.4 Attacker Tools

- **Multimeter:** An instrument that can measure basic electrical properties, such as voltage, current, resistance, and so on. We recommend choosing a multimeter with diode forward biasing and wire connectivity function. The model we used in this work is a US\$33 Stanley STHT77364, an US\$10 alternative Gator XL830L from could be an alternative.
- **Logic analyzer:** An instrument that can measure fast varying digital signals, records those signals over time domain, and performs analysis to discover the information encoded. In this work, we did not use a logic analyzer, in future work we will use a US\$300 Digilent Analog Discover 2 to demonstrate the analysis of U-Boot entry point by monitoring the off-chip Flash memory bus traffic. For such kind of task a 24MHZ 8 Channel open-source logic analyzer priced US\$15 is adequate.

- **Flash memory programmer:** A low-cost (US\$15) device to read out the Flash (generally also supports EEPROM) memory content from or write image files to Flash memory chips. A model with a test clip for fast and clean hooking up the exposed memory bus is more useful. The CH341A Pro Flash memory programmer used in this work is priced US\$10 on ebay.com.
- **JTAG programmer/debugger:** JTAG is also known as the IEEE 1149.1 standard for deploying and debugging firmware on the chip and also offers low-cost and time-saving testing for all components in a system through boundary-scan. JTAG is widely used in industry. Different system architectures may require different JTAG programmers, the one we used to extract electronic lock programming code from MSP430G2433 is an US\$150 MSP-FET430UIF. Compatible MSP430 JTAG programmer from a third party is around US\$35. An universal JTAG programmer supports ARM, MIPS and RISC-V also priced at US\$35 on ebay.com.
- **Embedded system development board:** Such as Raspberry Pi Zero (US\$10), Arduino UNO R3 (US\$22) or STM32 Bluebell (US\$15). Those development boards are useful, for example, to deploy JTAGenum³⁵, an open-source program for identifying JTAG pin-out definitions.

B.4 Exploitable Debug Interfaces and Exposed Memory Buses

This section will introduce exploitable debug interfaces and memory buses in COTS IoT devices and demonstrate how we can easily identify these interfaces for exploitation.

B.4.1 Open Debug Interfaces

JTAG is a commonly used debug interface for embedded systems. We may find one or two rows of jumper pins with a printed JTAG, JT x or J x label, where x might be a number as in Figure B.1 (a). Occasionally, we can also find individual pins labelled with TDI, TDO, TMS or TCK as seen in Figure B.1 (b). In some products, JTAG headers are there without any label—Figure B.1 (c)—or even hidden from sight as seen in Figure B.1 (d).

³⁵<https://github.com/cyphunk/JTAGenum>

B.4 Exploitable Debug Interfaces and Exposed Memory Buses

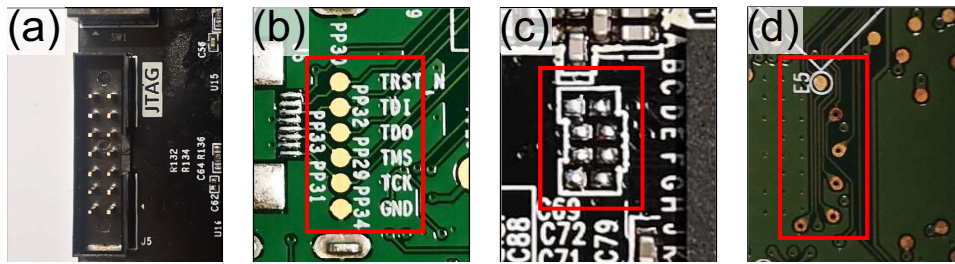


Figure B.1: JTAG debug interface found in IoT devices: (a) labeled with JTAG silk print; (b) labeled with JTAG pin-outs (e.g., TDI, TDO, TMS or TCK); (c) one or two row of test points nearby the MCU; and (d) hidden JTAG.

Table B.1: Basic JTAG signals. A JTAG interface may also have V_{supply} , V_{tref} , $RTCK$ and $TRACE$ signals—as described in the user manual of a JTAG debugger.

Signal name	Direction	Description
TRST_N	Programmer → Device	JTAG reset (active low)
TDI	Programmer → Device	JTAG scan-chain input
TDO	Programmer ← Device	JTAG scan-chain output
TMS	Programmer → Device	JTAG mode selection
TCK	Programmer → Device	JTAG clock
GND	Programmer ↔ Device	Common ground

To use the basic functionality of the JTAG debug interface, the five signals shown in Table B.1 need to be wired up correctly to a JTAG Debugger—see Figure B.6. There are no concrete definitions of the pin-out order for the JTAG header. A manufacturer can place the JTAG pins in any order. To determine the JTAG pin-definition, we can use the manual inspection method in Section B.5 or use the open-source JTAGenum listed in attacker tools in Section B.3.4.

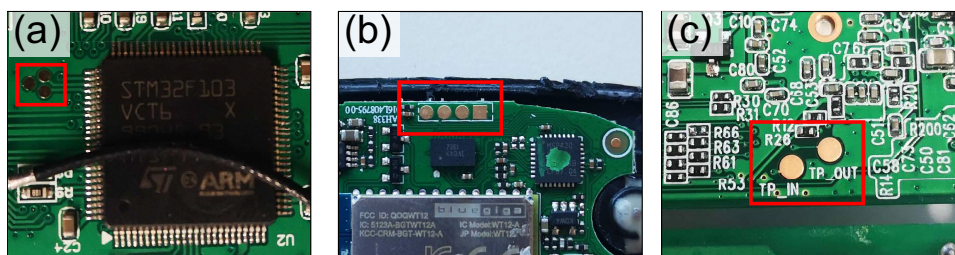


Figure B.2: Two-wire debug interfaces found in IoT devices: (a) ARM SWD debug interface; (b) TI SBW debug interface; and (c) manufacturer defined proprietary two-wire debug interface.

Other than JTAG, manufacturers may implement different debug interfaces, such as ARM SWD (Serial Wire Debug in Figure B.2 (a)) and TI SBW (Spy-Bi-Wire in Figure B.2 (b)). IoT device manufacturers can also implement a proprietary two-wire debug interface (e.g., the that found in a TP-Link Wi-Fi access point Figure B.2 (c)). The two-wire programming interfaces typically consist of one bi-directional data wire and one clock wire, sometimes including GND (ground) and V_{cc} (common collector voltage) for potential reference and power supply.

Two-wire debug interfaces benefit from the reduced pin design and are suitable for IoT devices with small form factors.

B.4.2 Exposed Memory Buses

Since the on-chip NVM is limited and expensive [261], many IoT devices use off-chip Flash or EEPROM (electrically erasable programmable read-only memory) to store firmware images, settings, and secrets.

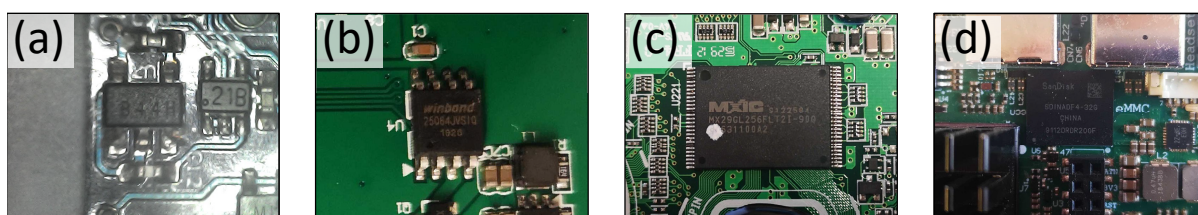


Figure B.3: Different NVM chips in IoT devices: (a) EEPROM in SOT23-5 package; (b) Flash in SOP-8 package; (c) Flash in TSOP-56 package; and (d) Flash in eMMC-153b package.

Figure B.3 lists a number of off-chip memory in IoT devices. EEPROM in SOT12-5 package in Figure B.3 (a) often appears in tiny devices, such as a smartwatch, due to its small footprint. Flash or EEPROM chips in the SOP-8 package are common off-chip storage in many IoT devices; such memory communicates with the microcontroller via a serial bus such as I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface), as exemplified in Figure B.3 (b). IoT devices requiring relatively higher storage space and faster access speeds may employ Flash chips with parallel buses, such as those shown in Figure B.3 (c) and (d)—parallel buses employ more wires to transfer data and address signals. We focus on memory chips using serial buses, as parallel buses are generally difficult to access and require professional and costly equipment. In addition, memory chips with parallel buses can use a BGA (ball grid array) package where all signal pins are hidden behind the package; therefore, it is more difficult to access [257].

B.5 Open JTAG Interface Exploit Case study

Post identification of interfaces for possible access, this section investigates the potential threat posed by demonstrating the extraction of memory contents in IoT devices in two example case studies under the practical threat model we consider in Section B.3.

Scenario. Consider the following scenario for our first case study: *Elizabeth is an 87-year-old widow, she lives alone in an apartment equipped with an electronic lock. One day the electronic*

B.5 Open JTAG Interface Exploit Case study

lock runs out of battery. Elizabeth asked one of her neighbours to replace the battery for the lock. However, the battery in the lock is an unusual model. The neighbour suggests bringing the lock to the hardware store to match the correct battery model. Half an hour later, the neighbour returned with the battery and re-installs the electronic lock for Elizabeth. Could the neighbour extract the key code for the electronic lock within the 30-minute time window to gain unauthorised access to Elizabeth's house?

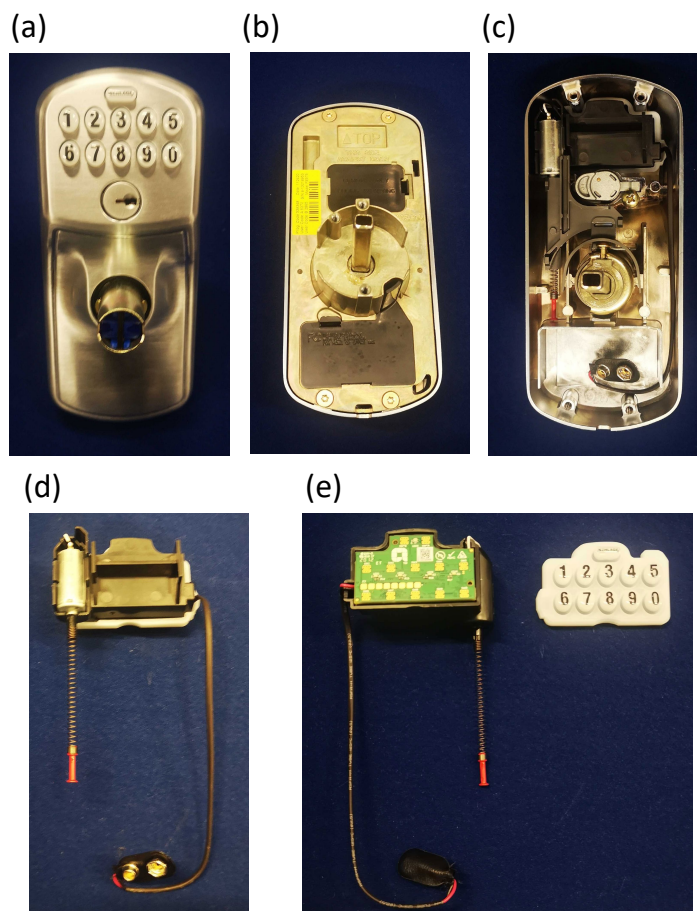


Figure B.4: Tear-down of a Schlage electronic lock FE575 electronic lock: (a) front side; (b) back side; (c) with back panel removed; (d) the electronic assembly; and (e) the circuit board is visible after removing the waterproof silicone rubber keyboard.

Attack. We use a Schlage electronic lock FE575 as an example. The tear-down of the electronic lock is summarised in Figure B.4. The mainboard of the Schlage electronic lock FE575 is shown in Figure B.5. Its circuit layout is uncomplicated. The black square in the middle is a buzzer. It makes a sound when the keyboard is pressed. At the left bottom corner of the buzzer is an MSP430G2433 microcontroller— the electronic lock's brain. Other components are mostly responsible for powering, motor operation—the circuit needs to drive a DC motor to release

the lock when a correct password is pressed. More importantly, there are seven pins in a row labelled with JT1, which is highly like a JTAG interface.

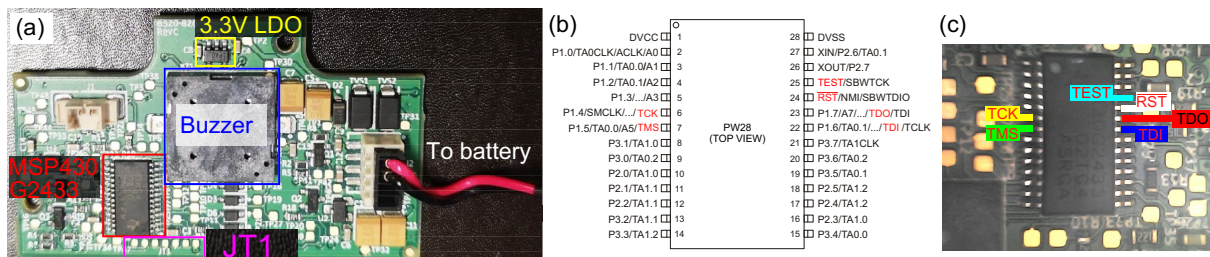



Figure B.5: The main board (a) of the electronic lock. The embedded microcontroller in the lock is a MSP430G2433, and JT1 is highly likely to be the JTAG interface. Highlighted JTAG signals on (b) are pin definitions reproduced from MSP430G2433 datasheet [262]; and (c) the microcontroller on the main board.

Unfortunately, the pin definitions for JT1 are not labelled on the PCB; hence a strategy is needed to identify the correct order of JTAG signals. Fortunately, as seen in Figure B.5, the MSP430G2433 microcontroller is in an SOP-28 package, all pins are easily accessible. Hence a US\$10 multimeter is adequate to identify pin definitions of JT1 without using professional tools such as the US\$200 JTAGulator.

Our technique to find correct JTAG pins is to use the diode and continuity mode of a multimeter. The diode and continuity mode is commonly labelled with a diode and a sound wave symbol (e.g., ). This mode measures the forward bias of a diode (if there is one between the two probes of the multimeter). If there is a direct wire connection (or short circuit), the multimeter buzzer will sound. Depending on the multimeter manufacturer, different probe polarities in diode test mode may be used. For our model Stanley STHT77364, the red probe is connected to the cathode, and the black probe is connected to the anode of the tested diode (although, typically, red is the anode and black is the cathode, refer to the multimeter model used for probe polarities). We will start with seeking for the GND, which can be easily accessed at: i) The negative terminal of the battery; ii) any EMI (Electromagnetic interference) shielding used; iii) any pin connected to the ground copper pour; iv) metal case of connectors, such as USB; v) the GND or V_{SS} pin of a known IC.

Plug the probes into the correct receptacle of the multimeter, select the diode and continuity mode. Attach the red multimeter probe to any of the exposed GND listed above. We choose to use the negative terminal of the buzzer. As this terminal is relatively large, it is easy to attach the probe and is directly connected to the ground copper pour. Then use the black multimeter probe to explore each of the seven pins in JT1. As summarised in Table B.2 under the column GND, the 5th pin of the JT1 has forward-biased voltage drop $V_F = 0$, which implies the JT1.5 is the GND pin in this JTAG connector.

B.5 Open JTAG Interface Exploit Case study

Other JTAG signals can be determined using a similar method. By looking up the datasheet of the MSP430G2433 microcontroller, we have highlighted the six JTAG signal pins in Figure B.5. Now attach the multimeter's black probe to each of the five highlighted microcontroller leads, and measure the V_F between them and each JT1 pin. The results are summarised in Table B.2.

Table B.2: The forward-biased voltage drop (in mV) measured between each JTAG signal and JT1 pins. Here, OL stands for open loop.

	GND	TRST_N	TDI	TDO	TMS	TCK	TEST
JT1.1	687	OL	OL	OL	OL	0	OL
JT1.2	714	OL	OL	OL	0	OL	OL
JT1.3	711	OL	0	OL	OL	OL	OL
JT1.4	714	OL	OL	0	OL	OL	OL
JT1.5	0	715	OL	OL	478	450	430
JT1.6	716	0	OL	OL	OL	OL	OL
JT1.7	523	OL	OL	OL	OL	OL	0

Table B.3: The estimated JT1 pins definitions.

JT1.1	JT1.2	JT1.3	JT1.4	JT1.5	JT1.6	JT1.7
TCK	TMS	TDI	TDO	GND	TRST_N	TEST

Based on the results in Table B.2, we can conclude that JT1.1 is the TCK signal, as there is a direct wire connection between the two points. Similarly, JT1.2 is TMS; JT1.3 is TDI; JT1.4 is TDO; JT1.5 is GND and JT1.6 is TRST_N. The JT1.7 is directly connected to the 25th pin of the MSP430G2433. According to the datasheet, this is the TEST pin connected to the internal device protection fuse. The product manufacturer should blow the internal fuse by applying a 6 V, 100 mA current to prevent further JTAG access. In addition, V_{CC} is absent in JT1; in this case, the board is powered from a battery while debugging instead of the JTAG debugger. The estimated JT1 pin definitions are summarised in Table B.3.

Next, we use a TI MSP430 JTAG debugger (or a lower cost, compatible model from a third party) and connect to the pins identified above. We opted to solder a connector to the PCB and use jumper wires to connect to the JTAG debugger as shown in Figure B.6. Soldering is not compulsory, a data repair tool (typically costing US\$20) can be used instead.

To access the internal memory contents, we used Texas Instrument's UniFlash software available at: <https://www.ti.com/tool/download/UNIFLASH>. If a suitable JTAG debugger is selected to match the target system, the connections are correct, and the JTAG fuse was not blown by the manufacturer, UniFlash will automatically detect the chip model. As shown

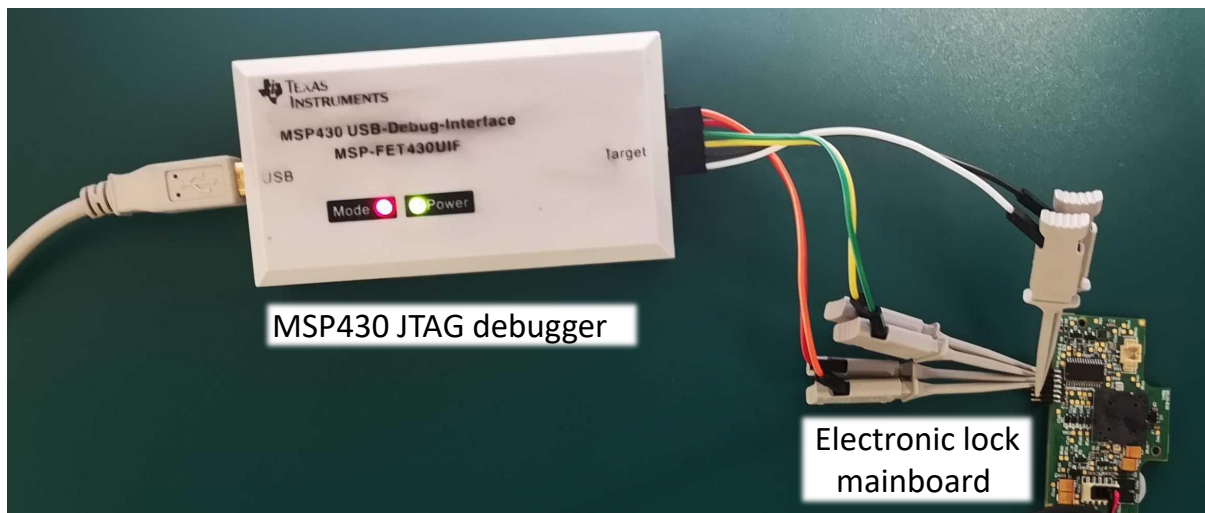


Figure B.6: JTAG debugger attached to the electronic lock main board.

(a)	Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	00000000	<u>53</u>	<u>93</u>	<u>48</u>	05	<u>28</u>	<u>65</u>	ff	0a	<u>53</u>	<u>70</u>	ff	0e	ff	ff	ff	10
	00000010	ff	ff	ff	11	ff	ff	ff	10	ff	ff	ff	11	ff	ff	ff	20
	00000020													f	ff	ff	30
	00000030													f	ff	ff	40
	00000040													f	ff	ff	50
	00000050													f	ff	ff	ff
	00000060													f	ff	ff	ff
	00000070													f	ff	ff	ff
	00000080													f	02	52	02
	00000090													f	ff	ff	ff

(b)	Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
	00000000	<u>17</u>	<u>07</u>	<u>12</u>	05	28	65	ff	0a	53	70	ff	0e	<u>50</u>	<u>15</u>	ff	12

Figure B.7: The memory content read from the electronic lock, (a) contains secret programming code ('539348') and user codes ('5370' and '2865'), in plain-text; (b) after the user change the programming code to '170712' and added a new user code '5015', the new codes can still be read out.

in Figure B.7 (a), the memory content read out from information memory (0x1000-0x10FF according to [262]), contains the lock's programming code, and user codes. All stored in plaintext. Those codes match that printed on the user's manual of the electronic lock and should be kept secret. Even if the user has changed the default codes, an adversary can still extract the new value from the same address. With the user code, one can gain entry without letting the homeowner know. This raises important questions regarding the security of such devices to insider attacks, and in a more practical setting, leaving the house unattended and in the company of individuals with lower degrees of trust—e.g. short term rental settings. Simple mitigation would be to change the default programming code and blow the fuse by putting a 6 V by 100 mA current to the TEST pin of JT1 to prevent further JTAG access.

B.6 Memory Bus Snooping Case study

In the second case study, we consider the following.

Scenario. *Emiko is an international student who lives in a shared house with a few housemates. She bought a Wi-Fi IP camera to monitor her room when she was out. One day, her suburb experienced power outages, and Emiko decided to stay at the University until power was restored. We demonstrate the risk of a malicious actor, under our treat model, sneaking into Emiko’s room (when the camera will not raise any potential alarms, given the power outage) and access the Wi-Fi IP camera to gain control and potentially facilitate peeping whilst leaving no visible marks of tampering and a fully operational device.*

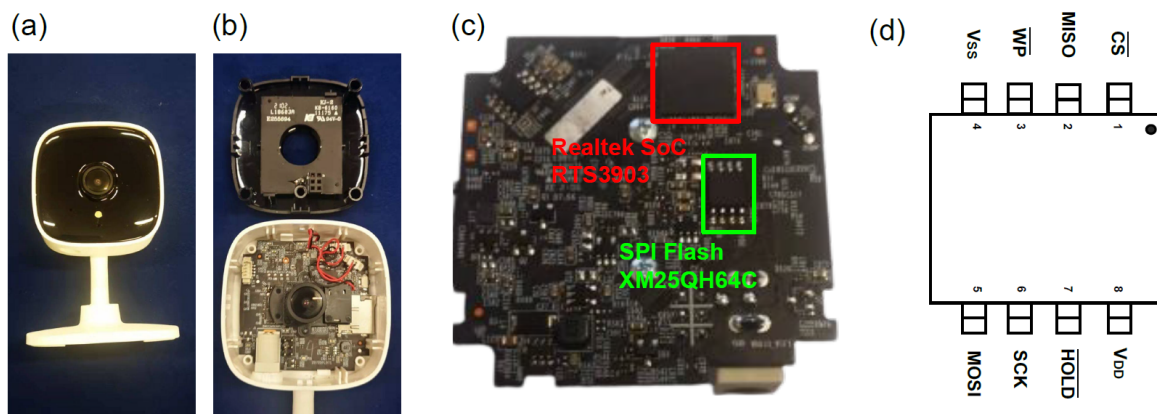


Figure B.8: The tear-down of the TP-link Tapo C100 IP camera: (a) the front side view; (b) with the front panel removed; (c) the back side of the main board; and (d) the pin definition of the MX25QH64 SPI Flash chip.

Attack. We employed a TP-link Tapo C100 IP camera as shown in Figure B.8 (a) to illustrate an attack that exploits an exposed memory bus. The casing of the camera is held together by snap-fit joints; we could disassemble it with a simple lever without leaving irrecoverable damage as shown in Figure B.8 (b). Most of the important logic components are located at the backside of the mainboard, as shown in Figure B.8 (c). The brain of the IP camera is the Realtek SoC (System-on-chip) RTS3903. Unfortunately, its datasheet is not publicly available. Besides the RTS3903, there is an 8 MiB SPI Flash chip XM25QH64C. In this case study, we target extracting information stored in this Flash chip. The XM25QH64C Flash chip is in an SOP-8 package. All its pins are exposed, pin definitions, reproduced from [263], are shown in Figure B.8 (d). As we demonstrate, we can easily use low-cost tools to access the exposed memory bus.

To snoop on the Flash memory chip, the easiest way is to use a US\$10 CH341A programmer with a test clip as shown in Figure B.9. The programmer will power the Flash chip and override

the SPI bus to send access commands, even when the camera is powered off. The memory contents can be dumped out using freely available software³⁶.

The IP camera we studied has the Flash chip and the Realtek SoC powered from the same power rail. When the test clip powers the Flash chip, the Realtek SoC will also start up and attempt to access the Flash chip. This will interfere with our memory readout. The easiest way to prevent the interference is to keep the Realtek SoC in a reset state. We spotted four unpopulated connectors near the SoC by inspecting the IP camera circuit board. Using the multimeter, we can conclude pin 1 and pin 4 are GND and 3.3 V V_{CC} , respectively. Pin 3 is pulled up to 3.3 V via a 4.7 K Ω resistor. We suspect it is either the reset pin or the TCLK signal of the cJTAG (Compact JTAG designed by MISP company) debugging interface. By trying to short pin 3 and pin 1, we observed the IP camera is reset, so we can conclude that pin 3 is the reset pin of the Realtek SoC. During the entire readout process, we need to short the reset pin, and the GND pin with tweezers to keep the SoC inoperative, as illustrated in Figure B.9 (b).

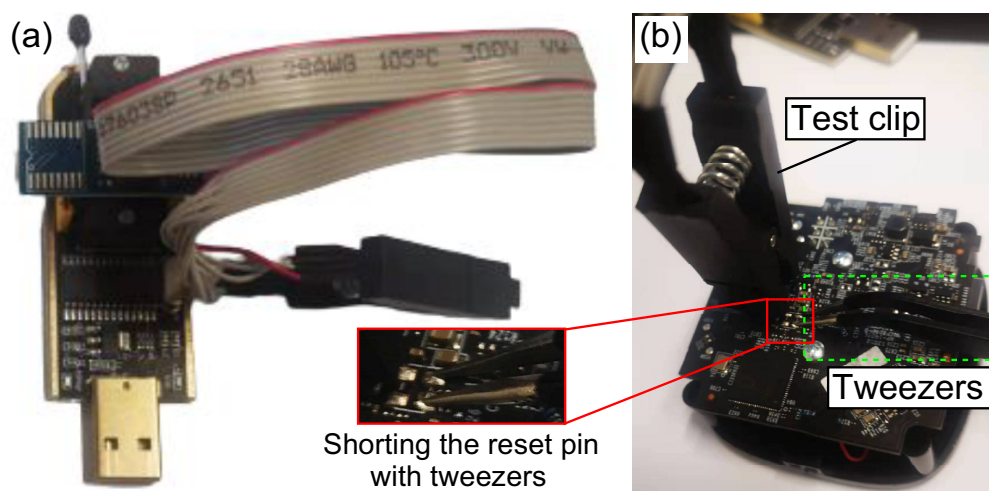


Figure B.9: (a) the CH341A USB Flash memory programmer; and (b) the test clip of the memory programmer attached to the MX25QH64 SPI Flash chip.

At this point, the binary image is dumped from the Flash chip, and the victim device can be re-assembled. No further physical access is required. Next, we follow the Blog post at <https://drmnsmoliu.github.io/> to extract the sensitive information. The user configurations (password, Wi-Fi SSID, and passphrase) are located at memory address 0x40000-0x50000 in the dumped file. However, this partition is compressed with `zlib` and encrypted with DES (data encryption standard).

The camera must first decrypt the configurations at start-up and read the Wi-Fi password before connecting to the Internet. Hence the DES key must be stored on-device. According to the

³⁶Available: <https://www.instructables.com/CH341A-Programmer/>

B.6 Memory Bus Snooping Case study

blog post, the key is derived from a string ‘C200 1.0’ (corresponding to the model number and hardware version) at address 0x600c0. However, at this address, in the memory image we dumped, there is a meaningless string (0x06 0x68 0x7a 0x88 0xa8 0xa7 0x01 0x97). We know the model number of our IP camera is C100, and the hardware version is 2.0 according to the nameplate. Therefore, it is natural to seek to search for ‘C100 2.0’ in the dumped file. A matched string appears at memory address 0x700c0. The sting ‘C100 2.0’ is the model-specific key material. To derive the correct DES key, a hash function³⁷ extracted from IP camera’s firmware is used. By replacing the key material ‘C200 1.0’ with ‘C100 2.0’, a 64-bit DES key ‘249c6923’ can be derived. Since OpenSSL takes hex strings instead of character strings, we must convert the key into hexadecimal value 0x3234396336393233. Subsequently, we can employ the following command to decrypt the dumped memory image:

```
# openssl enc -d -des-ecb -nopad -K [DES key] -in [dumped image] -out [out file]
```

This command specified using the encrypting function ‘enc’ in the OpenSSL toolbox. The parameter ‘-des-ecb’ specifies selecting the DES cipher and operating it in ECB (electronic codebook) mode. No padding is used as ‘-nopad’. The DES key 3234396336393233 should be filed after the keying flag ‘-K’. The dumped memory image is passed in, following the ‘-in’ flag. In the end, use the ‘-out’ flag to specify a location to save the decrypted file. Once the dumped memory image is decrypted, we can use `binwalk` command to decompress it:

```
# binwalk -e [decrypted file]
```

`binwalk` is an open-source toolbox for firmware image analysis. Flag ‘-e’ indicates to extract known file types automatically. The extracted user configuration will be placed in a new folder. If successful, a readable user configuration should be available.

Inside the extracted user configuration file, we can see the IP address, supported network protocols, user name, and passwords. Instead of storing a clear text password, the IP camera stores a hash value of the password. As a hash function, it is undesirable to be able to invert a hash value to its plaintext form (a.k.a., pre-image attack [252]). One promising attack to revert the hash is to use a rainbow table. Rainbow table is a pre-computed mapping table from chosen plaintext to hash values and vice versa. We have used the online rainbow table website <https://crackstation.net/> to successfully revert the password of one shared account in our IP camera and used this information to successfully gain access to the RTSP (real-time streaming protocol) from the IP camera.

³⁷Available: <https://drmsamoliu.github.io/assets/code/key.c>

We demonstrate obtaining unauthorised access to a video stream using information extracted from the Wi-Fi IP camera's memory dump in:

<https://www.youtube.com/watch?v=fnIn9QugrXI>



scan to watch

Importantly, the entire attack process takes *less than 25 minutes*. Only the *first 8 minutes require physical access to the target camera*. The firmware analysis and cracking of the dumped firmware can be done offline, using freely available tools.

Summary. The data stored in off-chip Flash memory can be easily read out through the exposed memory bus even when the system is powered off. The manufacturer has employed multiple techniques to enhance the security, such as encrypting the user configuration partition and storing hash value instead of the original password. The shared password can be reverted from hash using a rainbow table in our demo. Password salting could effectively mitigate such attacks.

B.7 Conclusion and Discussion

This appendix considered the dangers of open debug interfaces and exposed memory buses in commercial IoT devices. With two case studies, we showed the simplicity and the low cost of attacks by a person with entry-level knowledge on embedded systems—notably, the first attack only required less than 30 minutes and the second, only requires less than 8 minutes of access to the device. Evidently, security of IoT devices still require further emphasis from device manufacturers. Strategies such as disabling the debug interface supported by SoCs, securing the exposed memory buses by encrypting sensitive memory partitions and salting passwords are minimal to no additional cost step to improve current state-of-practice. However, secure on-device key derivation remains a challenging problem where memory fingerprint based methods can provide secure alternatives [38], [124], [264]. We hope our work will help support development of mitigation methods, inform threat models, and increase awareness of a different threat dimension posed by electronic devices employed in everyday life.

C.1 EPC Global C1G2v2 Command Encoding

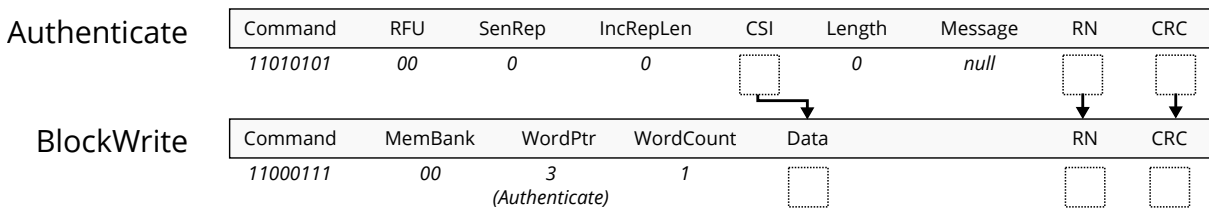


Figure C.2: Authenticate command encoding. We use MemBank = 0 (Reserved) in the BlockWrite command to indicate that the message is not a regular BlockWrite and WordPtr = 3 to indicate that the command should be processed as an Authenticate command. The 8-bit CSI field is placed in the 16-bit Data field of the BlockWrite command, and the RN and CRC fields are as specified for the BlockWrite command.

In order to mitigate the possible network delays in using a networked RFID reader connected to host machine to determine the delays, we employed two oscilloscope probes directly connected to the CRFID device to measure reservoir capacitor voltage and backscatter events to determine the time at which the backscatter event is initiated as well as to determine if the backscatter event immediately followed a cold start.

In order to independently evaluate key derivation and hash function execution, we created a second bootloader instance where the physically obfuscated key derivation code was replaced with the execution of a selected hash function. This allowed us to use the same method described to ascertain the success rate of the hash function execution under wireless powering conditions.

The success rate and latency of both lightweight physically obfuscated key derivation method and hash function executions are evaluated using the experimental setup shown in Figure C.3. Two probes are connected to the CRFID circuit board where the common ground (GND) is coloured in black in the figure. The backscatter signal probe lead is coloured in red, the regulated voltage (V_{reg}) is coloured in green. We define the time latency as the interval between the V_{reg} reaching 2.0 V (t_0) and the backscatter event (t_b). An operation is successful if a backscattering event occurs before V_{reg} drops below 1.8 V—minimum operating voltage of the MCU.

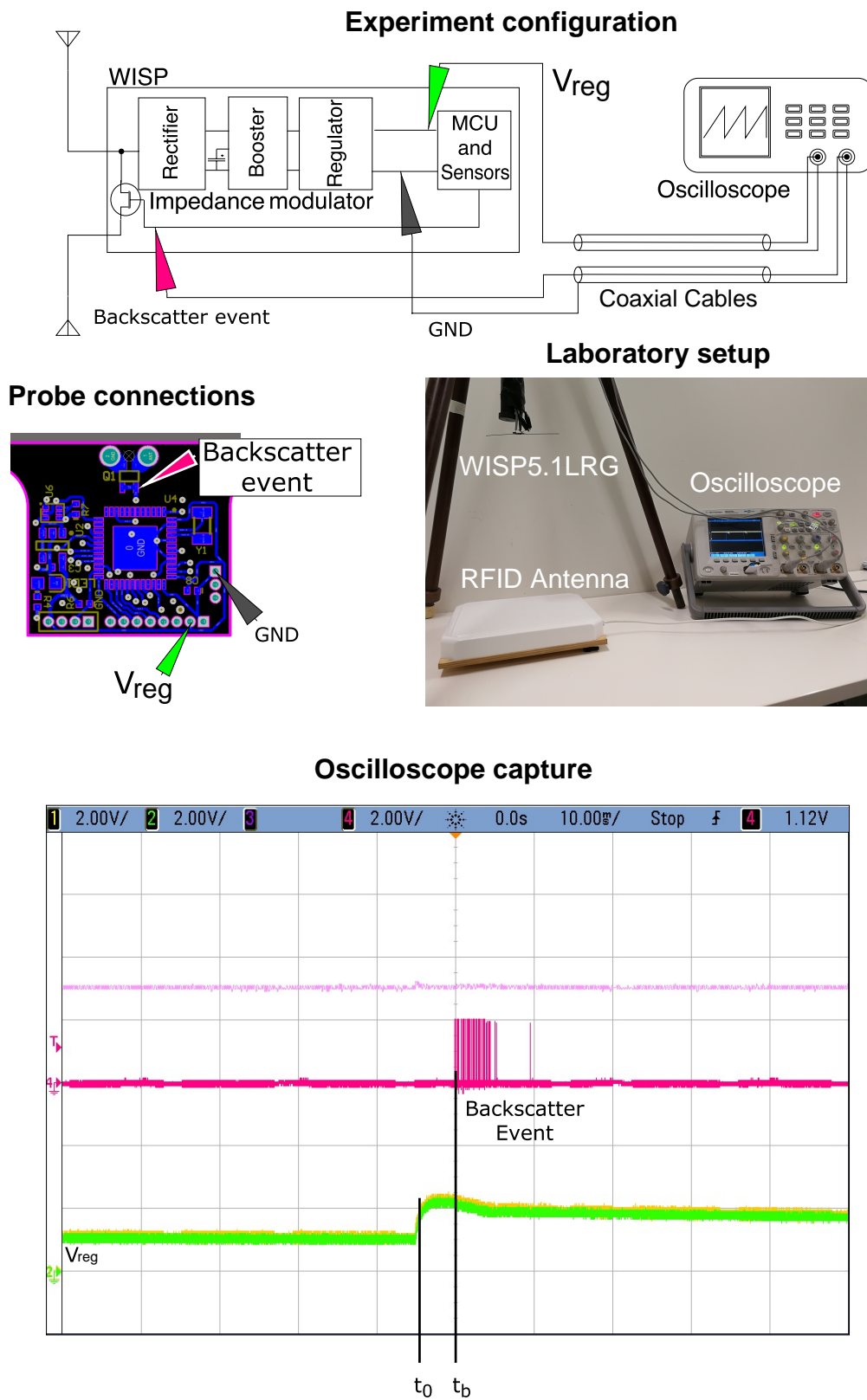


Figure C.3: Experiment setup for measuring latency and success rate for PUF key derivation and MAC function executions.

Appendix D

Chapter 4 Appendix

D.1 Memory Management Comparison

For implementing the Secure Storage component, several different mechanisms can be used:

1. **Isolated segments (e.g. using IP encapsulation hardware).** *Requirements and limitations:* It requires hardware features (such as IP encapsulation) to implement.
2. **Volatile secret keys** (see, for example, SRAM PUF [265]). *Requirements and limitations:* It might be computationally expensive to derive and erase the secret key, need a mechanism to ‘restore’ the volatile area on device restart, and a different mechanism is required to ensure the bootloader is immutable.
3. **Execute only memory (e.g. using MPU segments at compile time)** In this scheme secure memory is encoded as instructions (e.g. MOV) in execute only memory (XOM) (see [142]). *Requirements and limitations:* Implementation can be complex, since any code in this region must ensure that it does not leak data (e.g. in CPU registers or volatile memory), including from arbitrary jumps into the code. Additionally, since the bootloader is also unable to write to this region, it cannot be used to protect dynamic secrets (e.g. the broadcast session key).
4. **Runtime access protection (e.g. using MPU segments at runtime)** In this scheme, secure storage is available on device boot-up, but is locked (until next boot-up) by the bootloader before any application code is executed. We selected this method in Wisecr. *Requirements and limitations:* Method requires MPU hardware with runtime configuration and locking, and the implementation must ensure that the MPU is always configured correctly before any application code is executed.

D.2 Detailed Wisecr Update Scheme

Server Toolkit. Our Server Toolkit can be executed on a host with network connectivity to an RFID reader. The App loads in the ELF file generated from the compilation process, parses and slices it into MSPBoot specified 128-bit-long commands suitable for the RFID reader. The App then uses LLRP commands to construct `AccessSpecs` and `ROSpecs`. These encode *EPC C1Gen2* protocol commands such as `BlockWrite` and `SecureComm` commands to discover, engage and configure a networked RFID reader to execute the Wisecr protocol. Notably, we follow the same protocols for Host-to-RFID-reader and Reader-to-CRFID-device communications as in [22], [38] and detail implementation of Wisecr over *EPC Gen2* in the Figure 2.1.

Wisecr Update Scheme Over *EPC Gen2*. As described in Figure 7.15, Wisecr enables the ability to distribute and update firmware of multiple CRFID tokens, simultaneously. Given that a Server S communicates with an RFID device using *EPC Gen2*, a practicable, scalable and secure code dissemination scheme must be implemented over *EPC Gen2*. This requires communication between three separate entities: i) the host machine; ii) the reader; and iii) CRFID transponders—see main text Figure 2.1. Our scheme description here focuses on the communication between CRFID devices—tokens \mathcal{T} —and the RFID reader—the Server S over the *EPC Gen2* protocol as illustrated in Figure D.1—our open source code base provides a complete description (<https://github.com/AdelaideAuto-IDLab/Wisecr>).

Authenticating the Acknowledgement in the Validation Stage. It is theoretically possible to compute and include a MAC tag in an acknowledgement message at the end of the Validation stage but the implementation of this in practice is not possible. Our current acknowledgement signal is based on exploiting the last message in a singulation session—see RFID singulation phase in the Validation stage in Figure D.1. This last message returns the unique device identifier or EPC (electronic product code). We piggy back the current version number \mathbf{ver}_i in the data field part of this message as our *soft* validation signal. This is possible because at power-up (whether it be software or hardware reset) the device executes the bootloader and during its execution copies the \mathbf{id}_i and \mathbf{ver}_i to a global SRAM memory region for use after the MPU (memory protection unit) prevents access to this content stored in the secure storage area (see memory protection unit segmentation diagram in Figure 4.8).

Notably, even if a MAC tag was computed by the bootloader after power cycling (reboot) in the Validation stage, and stored in global memory for access after MPU protections to be used during the singulation phase that follows, the data field in the last EPC message is limited to 96 bits. Thus, there is inadequate room to piggyback a 128-bit MAC tag. Further, this would be an

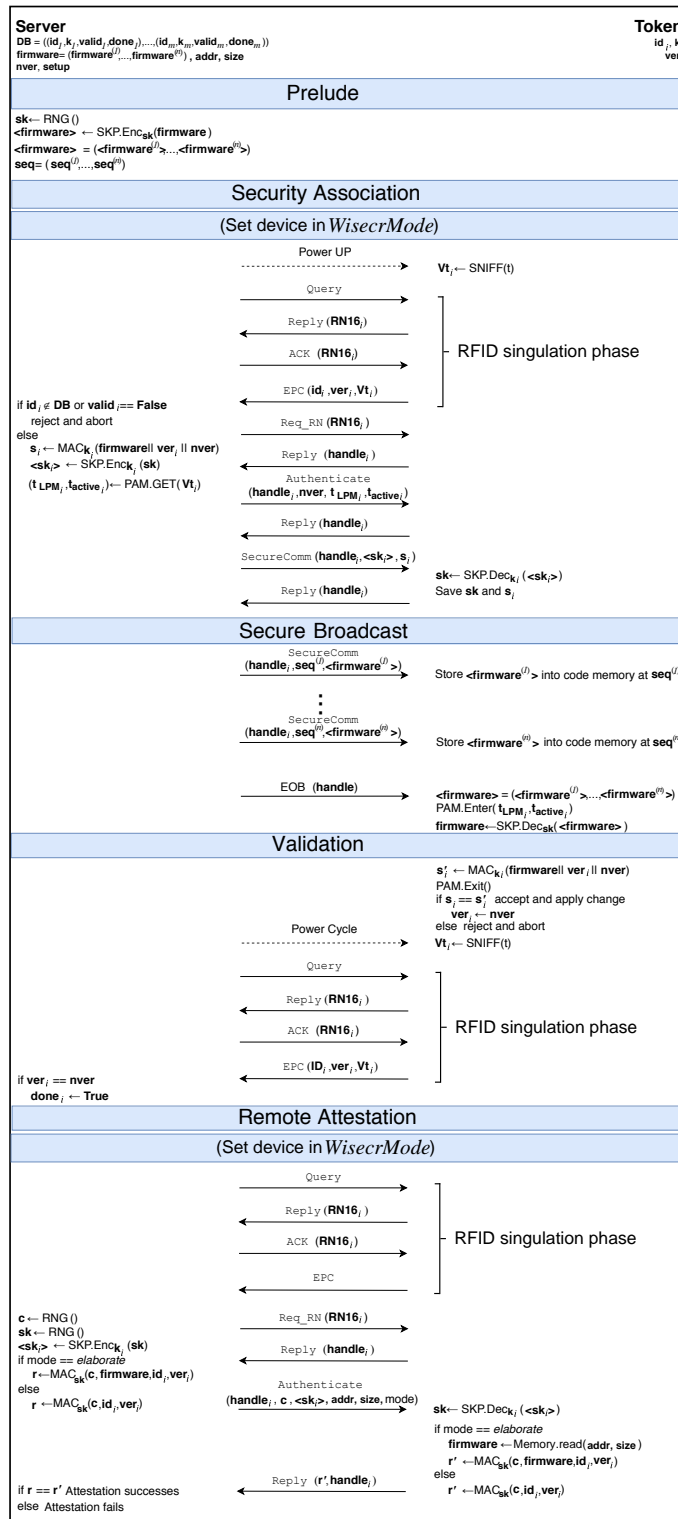


Figure D.1: Wisecr protocol implemented over the *EPC Gen2* protocol. To reduce complexity, we only provide the protocol sessions after a device is placed in the *WisecrMode*—as described in Figure 4.7—and highlighted here.

added overhead since the MAC computation would need to occur each time a CRFID device is powered (booted), irrespective of whether the MAC is needed or not.

D.3 Low Overhead Execution Scheduling

Thus, it is better to compute a MAC tag after reboot, on demand and when necessary by: i) singulating a token first; ii) instructing the token to compute the MAC tag; and iii) requesting the MAC tag. This is essentially the method we employ in the **Remote Attestation** stage that follows the **Validation** stage.

D.3 Low Overhead Execution Scheduling

Buettner, *et al.* proposed Dewdrop [162] execution model to *prevent* a brownout event under unreliable powering by executing tasks only when they are likely to succeed by monitoring the available harvested power. Dewdrop explores a dynamic on-device task scheduling method, however, requires the overhead of collecting samples of the harvester voltage and task scheduling by the device's application code.

In addition, Dewdrop is only suitable for CRFID devices equipped with a passive charge pump, such as WISP4.1 [13] since Dewdrop requires directly measuring the charging rate of the reservoir capacitor—charge storage element. In the follow-up, WISP version 5.1, the passive charge pump is replaced with a S-882z active charge pump and the reservoir capacitor is only connected to the load when V_{cap} developed across the capacitor exceeds the reference voltage of 2.4 V (V_{ref}). Consequently, in WISP5.1LRG the voltage delivered to the microcontroller is a sharp step-up, rather than a ramp-up function related to harvested power. Therefore, the charging-up process cannot be directly monitored by the technique in Dewdrop.

D.4 Powering Channel State Measurement and Power Aware Execution Model (PAM)

Observation. Generally, increasing distance of the token from a powering source lowers the harvester output power, RSSI and Read Rate of a given token.

Proposition. Measure powering channel state from the token is the most reliable measure of power available at a device.

Validation.

$$RSSI = P_t G_t^2 G_{path}^2 K, \text{ where } G_{path} = \left(\frac{\lambda}{4\pi d_o} \right)^2 |H|^2 \quad (D.1)$$

$$H = 1 + \sum_{i=1}^N g_t^i g_i^i \Gamma_i \frac{d_o}{d_i} e^{-jk(d_i - d_e)} \quad (D.2)$$

Although RSSI or received message rate (Read Rate) [29] measured by an RFID reader (Server) could provide a simple method to measure the powering channel state at a CRFID transponder, we observed these measures to be highly unreliable. This can be mostly understood by considering the complexity of the signal propagation model [98]—see equations (D.1) to (D.2) for details. We can see that RSSI depends not only on the transmit power P_t , the transmitter antenna gain G_t and backscatter coefficient K , but also the path gain G_{path} . However, G_{path} depends on signal wavelength λ , line of sight distance d_0 and the multi-path factor H ; where H is a complex function of angle alignment of the transmitter g_t^i and the device g^i , angle-dependent reflection coefficient Γ_i of i -th object, i -th path length for a total of N multi-paths—and the influences from the random access nature³⁸ of the media access control protocol used by the RFID air interface affecting RSSI and read rate measurements.

Therefore, the powering channel state is best estimated by the field deployed CRFID device harvesting available power at the device.

Thus, we introduce a power aware execution model where:

- The powering channel state at a CRFID tag is measured by the device using a single measurement of harvested power at boot-up (called a *Power Sniff* denoted as $\mathbf{Vt}_i \leftarrow \text{SNIFF}(t)$ in the update scheme). The voltage measure, \mathbf{Vt}_i , is used to estimate the power that can be harvested by a given CRFID device in the field; and
- The workload of scheduling execution of on-device tasks is determined by the resourceful Server (RFID reader and network infrastructure) as opposed to the resource limited CRFID device based on the channel state measurements from a CRFID.

We consider a harvesting device operating under the commonly used charge-burst mode where charge is first accumulated in a storage element—often a capacitor—and charge is released for useful work when an adequate amount of charge (measured in terms of the voltage) is reached at the charge storage element. Our power aware execution model (PAM) requires the Server (RFID reader and/or host) involved in the update to derive two parameters: i) time estimation to charge to a set voltage (t_c); ii) time estimation to a brown-out (t_b) based on the CRFID reported measurement \mathbf{Vt}_i . *Here, we reason that the distance between the antenna and the given CRFID device does not change during a given update session. Therefore, the power estimated at the beginning of the session is valid during the entire session.*

³⁸Notably, the RFID air interface relies on a slotted ALOHA media access control protocol.

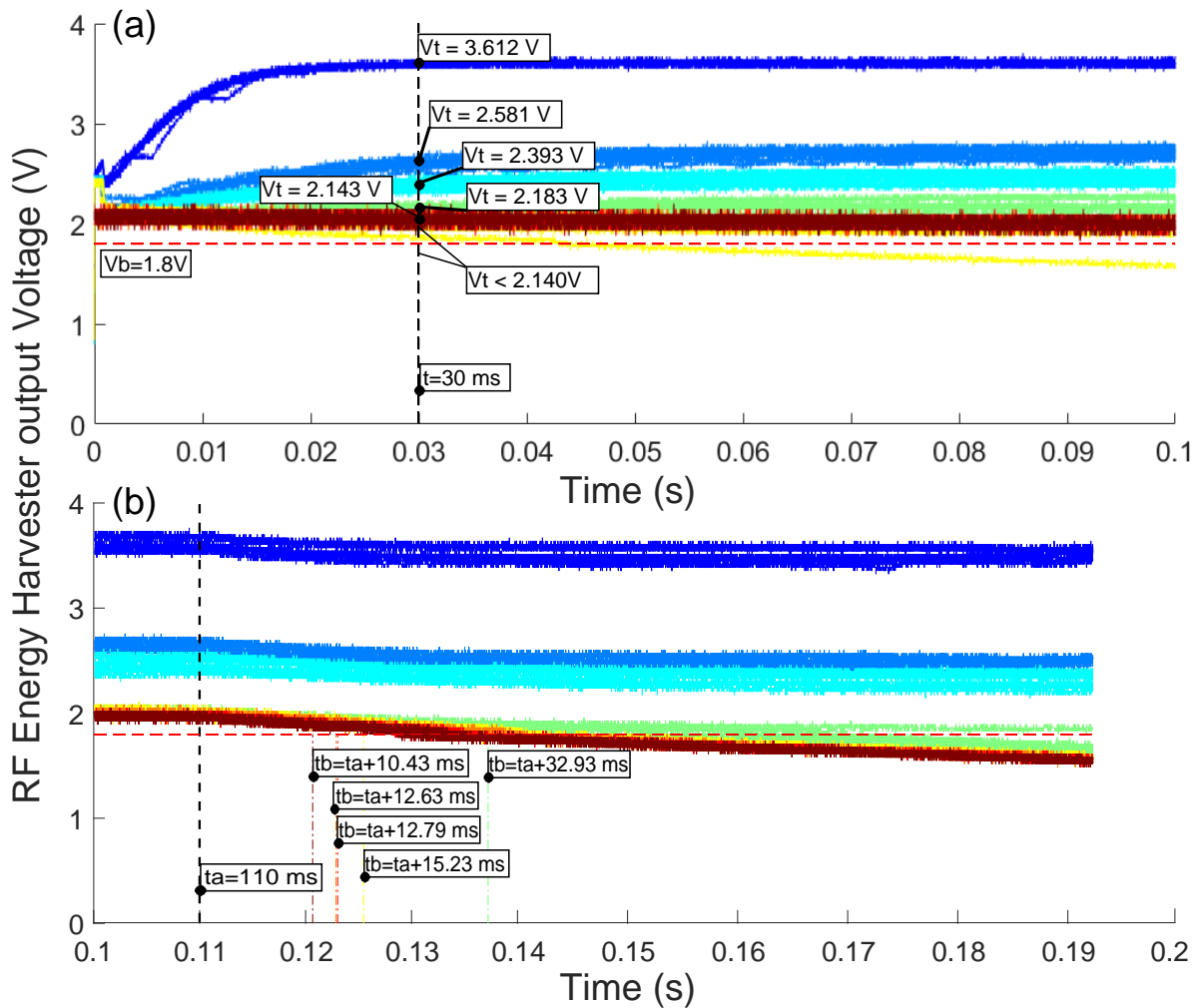


Figure D.2: Power measurements: (a) charging while the MCU is in LPM mode; and (b) discharging when the MCU is in active mode (we employed a MAC computation for the task). The charge and discharge experiments at each fixed distance were repeated 10 times to obtain a mean response.

Unfortunately, the RF energy harvester on a CRFID device is a non-linear circuit component whose output voltage changes over time as input RF power level varies. Therefore, it is non-trivial to model the RF energy harvester [162]. Therefore, instead of relying on an analytical model, we adopt an experimental method to derive the parameters for PAM empirically. We measure, in repeated measurements, the output voltage generated by the power-harvesting network on the WISP CRFID. We also measure the expected workload we can obtain from a CRFID device, before a brownout causes power loss. We extrapolate from these measurements to construct an empirical model for the PAM() function employed by the Server to determine the best set of execution parameters from the reported voltage V_t at runtime.

As depicted in Figure D.2 (a), the output voltage from the charge pump grows at different rates and as a function of received power. The traces do not intersect, therefore, if we measure the

charge pump output voltage at an early time, e.g, 30 ms as labelled in the picture, we can obtain a voltage across the reservoir capacitor V_{cap} , with which we can predict the time required for the reservoir capacitor to be charged to a certain voltage level. The charging rate of a capacitor becomes slower following a logarithm trend. It is too conservative to use the time to charge to nearly 100% saturated voltage; we empirically determined an adequate LPM time. We use the time to charge to approximately 63% of the saturated voltage, then compared to waiting for a nearly fully charged capacitor, we can reduce charge time by 75% and still accumulate adequate energy to execute the chunk of task under the active time period t_{active} .

The time before brownout is also a function of received power. We can observe in Figure D.2 (b), when the powering condition is good, the CRFID device can execute the MAC computation we employed for the load, continuously; starting from $t_a = 110$ ms without power failure. However, the device starts to fail or brown-out causing a power loss at $t = 142.92$ ms, giving a time to brown out of $t_b = t_a + 32.93$ ms for Vt below 2.183 V. As expected, we can see that t_b decreases as the received power decreases.

PAM function formulation. From our results, we can conclude: i) for $Vt \geq 2.393$ V, the CRFID token may continuously operate with no power loss. In such cases, $t_{LPM} = 0$, and $t_{\text{active}} = \infty$ is used (execution does not need to be altered; ii) for Vt within 2.183 V and 2.393 V, we can employ $t_{LPM} = 10$ ms and $t_{\text{active}} = 29$ ms (90% the measured t_b) for a conservative approach to prevent power failures; iii) for Vt within 2.143 V and 2.183 V, use $t_{LPM} = 15$ ms and $t_{\text{active}} = 14$ ms; iv) for Vt within 2.140 V and 2.143 V, use $t_{LPM} = 25$ ms and $t_{\text{active}} = 11$ ms; and v) if $Vt \leq 2.140$ V, then the CRFID device cannot accumulate adequate energy under such a condition to complete a computation intensive task, and we strongly suggest to not perform a code update in this case. We describe $(t_{\text{active}}, t_{LPM}) \leftarrow \text{PAM.Get}(Vt)$ function in equation (D.3).

$$\text{PAM.Get}(Vt) = \begin{cases} [\infty, 0] & , \quad Vt \geq 2.393 V \\ [29 \text{ ms}, 10 \text{ ms}] & , \quad 2.393 V > Vt \geq 2.183 V \\ [14 \text{ ms}, 15 \text{ ms}] & , \quad 2.183 V > Vt \geq 2.143 V \\ [11 \text{ ms}, 25 \text{ ms}] & , \quad 2.143 V > Vt \geq 2.140 V \\ [9 \text{ ms}, 30 \text{ ms}] & , \quad Vt < 2.140 V \end{cases} \quad (\text{D.3})$$

D.5 Transponder Modes & Broadcast Channel

Stork [22] proposed exploiting promiscuous listening by choosing one in-field token (CRFID) to stay in the non-overhearing mode and the rest in overhearing (observer or promiscuous listening) mode to create a logical broadcast channel. This method overcomes the unicast media access

D.6 Pilot Election Experiments

Table D.1: Comparing the settings of tokens.

Method	Pilot/Non-Overhearing role selection	Transponder mode	Ignore handle in SecureComm/BlockWrite	SecureComm/BlockWrite response
Stork [22] (insecure protocol)	First seen by Host	Overhearing	✓	✗
Wisecr (Our secure protocol)	The <i>lowest</i> V_t Token	non-Overhearing	✗	✓
		Observer	✓	✗
		Pilot	✗	✓

layer protocol to facilitate wireless code dissemination to multiple CRFID devices. Building upon Stork, our Pilot-Observer mode (Section 4.3.2) makes a key improvement. In contrast to Stork [22], for the token under Non-overhearing/Pilot mode, instead of selecting the first device responding to an interrogation signal from the Server, we elect the device with the lowest reported voltage (V_t) as the pilot token. The similarities and differences between our work and the Stork are summarised in Table D.1.

This method forces the broadcast session to be driven by the device with the lowest available energy, and the highest probability to brownout; thus, increasing the chance of all non responding tags remaining in synchronicity with the processing of the broadcasted firmware, simply because these devices are able to harvest more power than the pilot token whilst also not having to respond to the Server.

D.6 Pilot Election Experiments

We have collected experiments for 20 cm, 30 cm, 40 cm and 50 cm as shown in Figure D.3. Generally, at 20 cm and 30 cm, all methods can succeed, with little differences in terms of the number of attempts and latency. In comparison, all methods failed to update all four tokens in 10 attempts at 50 cm; *although several devices were often updated, no attempt resulted in all four tokens being updated at this powering level*, hence the success rate is reported as zero. However, in the regions where devices are likely to operate at the threshold of powering, seen at 40 cm in our experiments, our proposed pilot election method performs best. Further, the proposed method is also seen to perform more consistently; indicated by the consistent success rate across different powering conditions achieved with different distances.

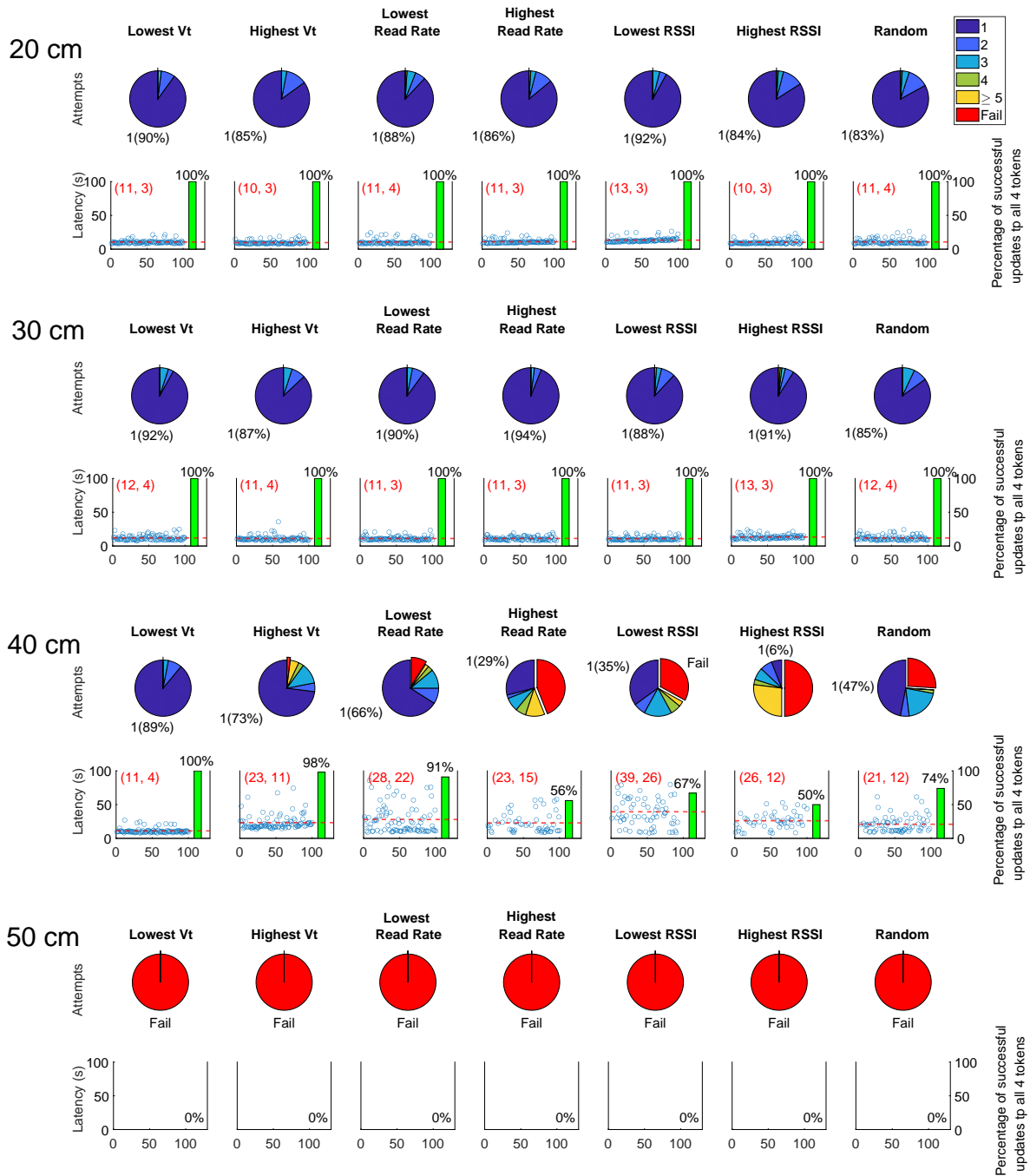


Figure D.3: Evaluation of the pilot token selection strategies we propose. Four tokens were placed at 20 cm, 30 cm, 40 cm and 50 cm above a reader antenna. The pie chart shows the number of attempts to have all 4 CRFID devices or tokens updated. The scatter plot shows the corresponding latency for successful updates. The results are obtained over 100 repeated measurements, where each measurement included 10 attempts to update all four CRFID devices. The bar graph denotes the number of successful updates—defined as all four tokens being updated over the 100 repeated measurements. Here (mean, standard deviation) latency statistics are given in red text.

D.7 Execution Overhead for Receiving Broadcast Packets

It is non-trivial to analyze the clock cycles for receiving a packet and sending a reply, as opposed to other tasks, as they are executed under a constant clock speed. Notably, the CRFID tokens do not have a hardware implementation of the wireless communication protocol. The RFID communication protocol is implemented in software. Since RFID communications require strict timing requirements, the CPU clock is dynamically configured and the relevant source code is written in assembly language to meet the strict timing requirements. For example, when the device is receiving a packet, the CPU works at 16 MHz. When the device is sending a reply by modulating its antenna impedance, the CPU works at 12 MHz. So it is difficult to employ our previous method to predict clock cycles by monitoring GPIO pins.

To overcome the above challenge, we used debugger tools to read the CRFID device’s internal states by inserting three breakpoints: Breakpoint 1: before the receiving routine is called; Breakpoint 2: in between the receiving routine and the reply routine; Breakpoint 3: after the reply routine (as illustrated in Figure D.4). Clock cycles for receiving and replying can be acquired by looking at the clock profile counter at each breakpoint.

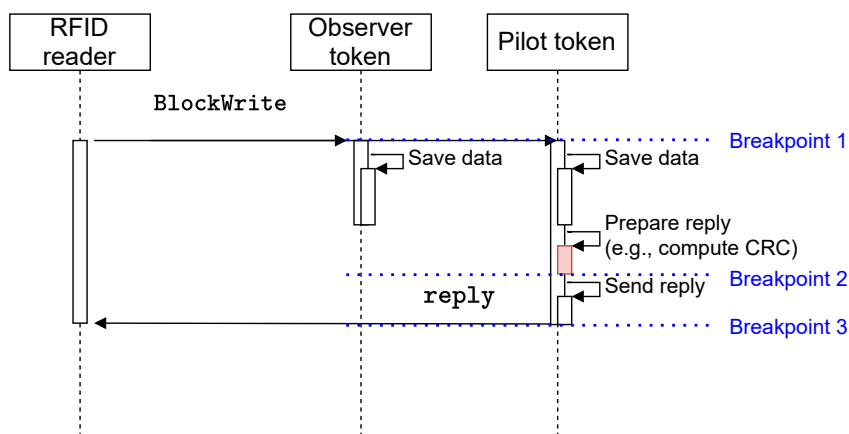


Figure D.4: Method to measure clock cycles for receiving a packet and acknowledging a received packet.

The `SecureComm` command in the *EPC Gen2* specification is yet to be widely supported in commercial RFID hardware. Therefore, as we mentioned in Section 4.3.2, we implement this command over the existing `BlockWrite` specified to support variable payload size as described in [181]. However, the Impinj R420 RFID reader we used in experiments implements it in a distinct manner. Irrespective of the specified payload size, the Impinj R420 reader (software version 5.12.3.240) always splits the payload into multiple `BlockWrite` commands, each command carrying a payload of only 2 Bytes [266]. Further, busy waiting is used while

receiving a command from the RFID reader (for example, at <https://git.io/J1108>). Hence, clock cycle results can vary from measurement to measurement. Hence, in our experimental results, we report the average clock cycles for such `BlockWrite` commands over 100 repeated measurements. We summarise the results below:

- Pilot token's cycles to receive a `BlockWrite` packet (with 2 Byte payload): 23,082
- Pilot token's cycles to send a reply to the `BlockWrite` packet: 1,131
- Observer token's cycles to receive a `BlockWrite` packet (with 2 Byte payload): 22,002

The clock cycles for an observer token to receive a `BlockWrite` packet is 1,080 smaller than the pilot token (this number is obtained by calculating the difference between the pilot and observer tokens' clock cycle counter at Breakpoint 2, over the average obtained from 100 measurements). This is because the observer does not need to prepare the reply packet (ACK), which requires a CRC-6 calculation, as illustrated in Figure D.4.

The total number of packets required in an update can be computed with:

$$N_{\text{packets}} = \frac{\text{firmware size (in Bytes)}}{2 \text{ Bytes per BlockWrite command}}$$

The number predicted using the above equation is in an ideal case, the actual number may be higher considering retransmissions, for example, due to communications errors identified using the CRC.

D.8 Experiment Setup to Access Token's Internal State

Some of our experiments require to measure the internal states of the token. For example the impact of four key device tasks on power-loss and the evaluation of our proposed power PAM method in Section 4.3.2. However, we do not have a precisely controlled RF environment (i.e., anechoic chamber) to remove the impact of the multipath signals constructively and destructively interfering with the RF powering of a device.

The measurement processes for Figure 4.2 and Figure 4.4 were different and complicated by the probes and wires that we need to attach to the device, the Digital Storage Oscilloscope (DSO)

D.8 Experiment Setup to Access Token's Internal State

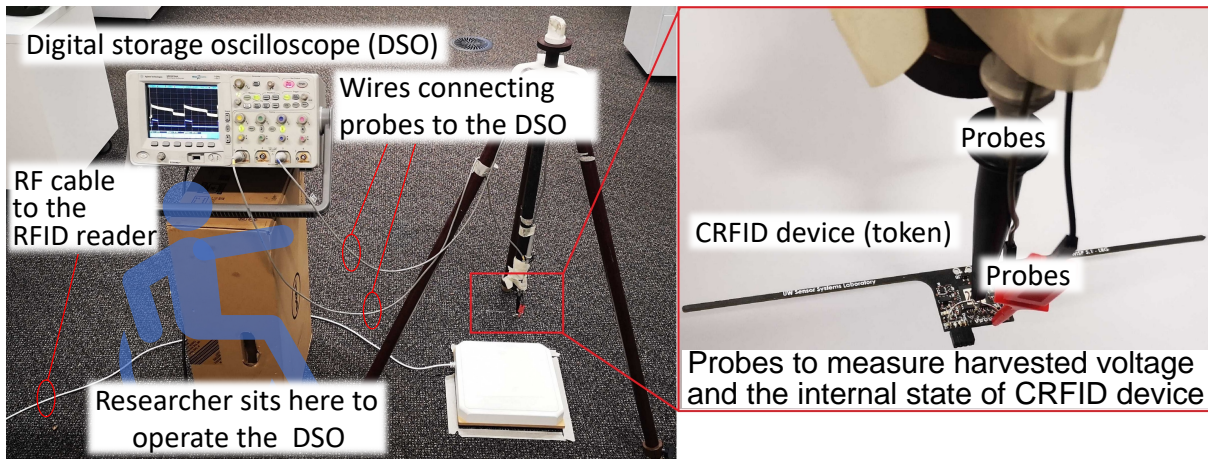


Figure D.5: Experiment setup and instrumented CRFID device used for the experiments summarised in Figure 4.2 and Figure 4.4. Notably conducting the experiments require a researcher to be seated next to the DSO. The probes, the changing positions of the wires leading from the probes, especially at different distances, and the orientation of the researcher significantly impacted these measurements.

we need to keep near the setup as well as the close proximal presence of the researcher to operate the DSO to enable taking the measurement as shown in Figure D.5.

To try to mitigate the influence of factors discussed above, we conducted these experiments following the method we describe below:

- Ensure we used the same CRFID device for both experiments.
- Try our best to keep the multipath environment the same across the two experiments. Hence, instead of adjusting the distance (which suffers from different multipath reflection and interference) and the changing positions of the probes and wires, we have the CRFID device fixed at 20 cm above the reader antenna, and *adjusted the transmit power of the RFID reader following the method in [95]*.

We can employ this method because, according to the free-space path loss equation [97] given below, adjusting the transmit power P_t of the RFID reader has a similar impact on the received power P_r as changing the distance d . Because, the RF wavelength λ is relatively constant (notably RFID employs frequency hopping regulations) whilst the RFID reader antenna gain G_t and CRFID device antenna gain G_r are fixed.

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi d} \right)^2$$

Notably, in the evaluation of PAM, what we are interested in is the available harvested power, distance is just one factor we can use to control the available power at the device in our

experiments. Adjusting the transmit power endows us with a more accurate form of control over the available power as that can be done using the RFID reader software, programmatically, and without interfering with the devices setup (such as changing distances and the arrangement of the probe wires and the multi-path environment).

D.9 Firmware Update to Mobile Tokens

We tested firmware updates to mobile CRFID devices. We rotated the rotor by hand, with a rotation speed of approximately 1 RPM (revolutions per minute) and conducted 100 repeated firmware updates to the four CRFID devices in the rotor blades (as illustrated in the video <https://youtu.be/AVrf0rNM0z8>). Here, as in other experiments, for each firmware update, the Host makes 10 attempts to update all 4 devices. The result in Figure D.6 shows that 3 updates resulted in disseminating firmware to all four tokens, and in 51 updates, at least one token is updated. We can conclude that updating firmware to mobile CRFID devices is possible but the success rate can be expected to reduce dramatically.

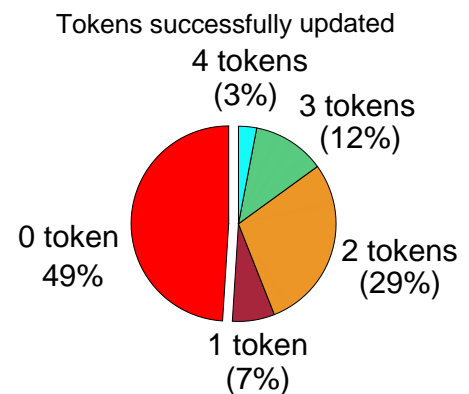


Figure D.6: Attempt to update firmware while the UAV rotor is rotated by hand at approximately 1 RPM. The pie chart shows the number of CRFID devices updated in the 100 repeated firmware update executions.

Appendix E

Chapter 5 Appendix

Table E.1: The implementation overhead of different Hash functions (message size= 240 bytes).

name	digest size	clock cycles	FRAM usage	SRAM usage
DM-SPECK64	64 bits	178,448	1,566 bytes	52 bytes
BMW-256	256 bits	150,046	11,398 bytes	215 bytes
SHA1	160 bits	159,969	12,442 bytes	94 bytes
BLAKE2s-256	256 bits	106,482	4,964 bytes	238 bytes
BLAKE2s-128	128 bits	104,723	4,961 bytes	238 bytes
SHA3-256	256 bits	584,126	3,652 bytes	472 bytes

Table E.2: BCH code encoding overhead.

(n_1, k_1, t_1)	clock cycles	FRAM usage	SRAM usage
(63,18,10)	53,944	858 bytes	114 bytes
(63,16,11)	51,003	738 bytes	117 bytes
(63,7,15)	31,577	842 bytes	126 bytes
(127,64,10)	238,671	858 bytes	197 bytes
(127,57,11)	248,433	1,002 bytes	204 bytes
(127,50,13)	235,005	1,034 bytes	211 bytes
(127,43,14)	219,095	1,050 bytes	218 bytes
(127,36,15)	198,758	1,044 bytes	225 bytes
(127,29,21)	181,438	1,058 bytes	232 bytes
(127,22,23)	167,509	1,072 bytes	239 bytes
(127,15,27)	111,335	1,054 bytes	246 bytes
(255,123,19)	930,093	1,370 bytes	394 bytes
(255,63,30)	680,087	1,418 bytes	454 bytes
(255,47,42)	583,024	1,446 bytes	470 bytes
(255,37,45)	476,744	1,492 bytes	480 bytes
(255,29,47)	377,220	1,506 bytes	488 bytes
(255,21,55)	294,783	1,520 bytes	496 bytes
(255,13,59)	201,535	1,418 bytes	504 bytes

Table E.3: BCH code decoding overhead.

(n_1, k_1, t_1)	clock cycles	FRAM usage	SRAM usage
(63,18,10)	393,836	1,882 bytes	1,226 bytes
(63,16,11)	435,027	2,022 bytes	1,168 bytes
(63,7,15)	626,881	2,572 bytes	1,154 bytes
(127,64,10)	670,622	3,676 bytes	1,226 bytes
(127,57,11)	748,933	3,942 bytes	1,228 bytes
(127,50,13)	1,030,444	4,466 bytes	1,228 bytes
(127,43,14)	978,775	4,728 bytes	1,228 bytes
(127,36,15)	1,057,415	5,006 bytes	1,218 bytes
(127,29,21)	1,574,426	6,602 bytes	1,228 bytes
(127,22,23)	1,742,276	7,134 bytes	1,226 bytes
(127,15,27)	2,102,222	8,198 bytes	1,228 bytes
(255,123,19)	2,515,163	11,958 bytes	1,228 bytes
(255,63,30)	4,116,796	17,700 bytes	1,228 bytes
(255,47,42)	6,102,010	23,964 bytes	1,228 bytes
(255,37,45)	6,582,507	25,530 bytes	1,226 bytes
(255,29,47)	6,976,341	26,574 bytes	1,228 bytes
(255,21,55)	8,345,992	30,750 bytes	1,226 bytes
(255,13,59)	8,528,363	34,566 bytes	1,298 bytes

Appendix F

Chapter 6 Appendix

F.1 Derivation of Performance Metric Models

In this section, we detail the derivation of equations in Section 6.3, where the following ideal chip model is adopted.

F.1.1 Synthetic Chip Model

Due to the size limitation of physical chips and difficulty of taking a massive number of repeated measurements from physical chips, we adopt a synthetic chip model to evaluate our analytic predictions. The synthetic chip model used is from [126] and is built under the following settings:

1. Each bit has a 50% chance being logic ‘1’ or ‘0’ during the enrolment phase. Each initial bit value is randomly generated during chip initialisation.
2. Each bit has an equal probability, BER_f , of being flipped during a regeneration.
3. The values of bits are independent and identically distributed (i.i.d.); hence, we assume no spatial or temporal correlations.

F.1.2 Unreliability Formalisation of S-Norm Transformation

As described in Section 6.2.2, all possible cases of noise-tolerant S-Norm transformed bit F are shown below.

$$F \leftarrow \mathcal{T}_{\text{SNorm}}(\mathbf{f}^{n \times 1}, \theta) = \begin{cases} 1, & \|\mathbf{f}\|_1 \geq \lceil \frac{n}{2} \rceil + \theta \\ 0, & \|\mathbf{f}\|_1 \leq \lfloor \frac{n}{2} \rfloor - \theta \\ \perp, & \text{Otherwise} \end{cases}$$

The formalisation is visualised in Figure F.1. Recall that a F bit can be transformed from n raw bits and the ℓ_1 -Norm of the F is between $[0, n]$. To assess the worst-case BER_F , we consider the

F.1 Derivation of Performance Metric Models

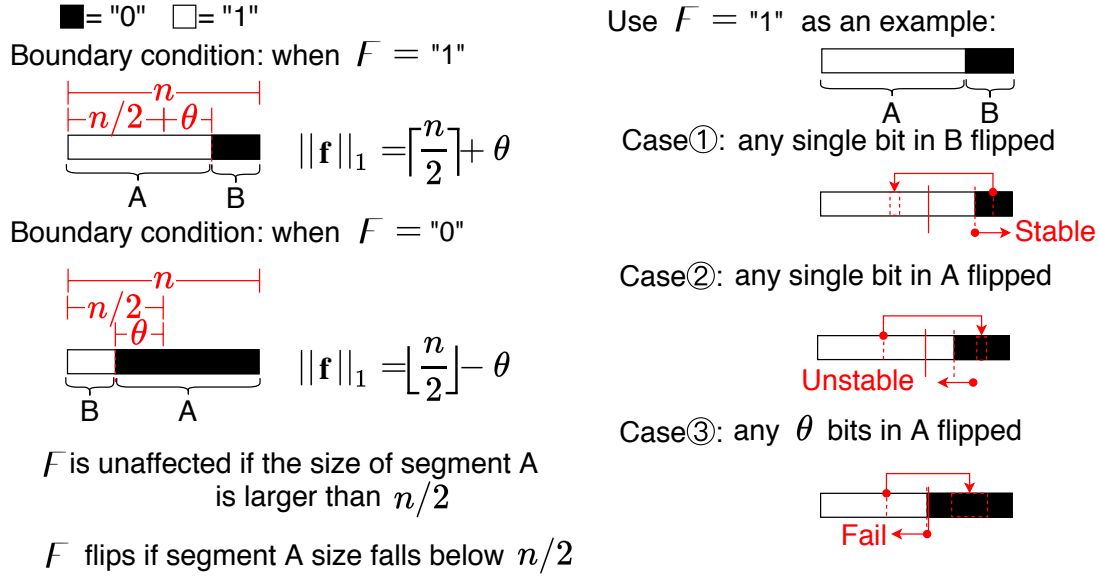


Figure F.1: S-Norm-based NoisFre. Two boundary conditions are illustrated: when $F = '1'$ where $\|f\|_1 = \lfloor \frac{n}{2} \rfloor + \theta$, and when $F = '0'$ where $\|f\|_1 = \lfloor \frac{n}{2} \rfloor - \theta$. Here we use $F = '1'$ as an example to demonstrate the influence of flipped raw bits on their transformed 1-bit F . The generated ℓ_1 -Norm is partitioned into two segments: A and B. Consider three representative cases: ① any single raw bit flip in B will enhance the reliability of the transformed bit F ; otherwise ② any single raw bits flip in segment A will deteriorate reliability of the F ; ③ the F will fail/flip if there are θ or more raw bits flipped in segment A.

condition where the selected word's ℓ_1 -Norm is exactly equal to $\lfloor \frac{n}{2} \rfloor + \theta$, as shown in boundary condition $F = '1'$ in Figure F.1. Here, θ is a threshold to select highly reliable F bits.

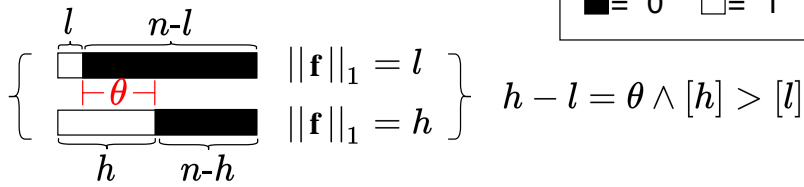
Each raw bit is with BER_f probability to be flipped under reevaluation. Using boundary condition $F = '1'$ as an example, on the one hand, ①, if there are raw bits of '0' (marked as segment B) flipping, it will increase the tolerance of the number of *raw bits of '1'* that allows being flipped (in segment A) *without influencing F bit*. In contrast, ② flipping raw bits of '1' (marked as segment A) will potentially result in an error to the F bit. Further, ③, supposing that raw bits of '0' (marked as segment B) remain unchanged, if more than θ raw bits of '1' flip, the F will exhibit an error—flipping from '1' to '0'. To be precise, the transformed F bit *will not exhibit* error unless more than $\theta + i$ raw bits of '1' flipping.

Overall, bit flipping within raw bits of '0' (marked as segment B) increases the reliability of extracted \mathbf{F} . In contrast, bit flipping within raw bits of '1' (marked as segment A) decreases the reliability of extracted \mathbf{F} . The boundary condition $F = '0'$ is logically equivalent to the case $F = '1'$ but, only inverts F 's '0'/'1' value rather than its BER_f .

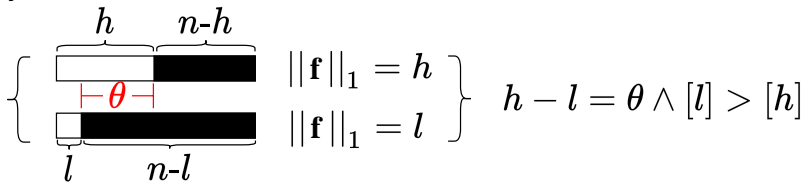
Without losing generality, we focus on one case shown in ① in Figure F.1. $\|f_i\|_1 = \lfloor \frac{n}{2} \rfloor + \theta$, the probability of having exact x error bits in segment A can be expressed as $\Pr_{|x| \in A}^{\text{flip}} = \text{binopdf}(x, \frac{n}{2} + \theta, \text{BER}_f)$, given that each raw bit has a BER_f probability of flipping. Similarly,

(a)

Boundary condition: when $F = "0"$

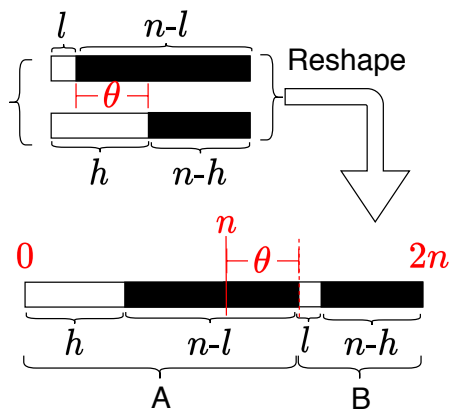


Boundary condition: when $F = "1"$

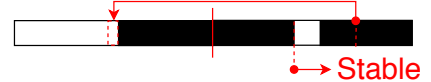


(b)

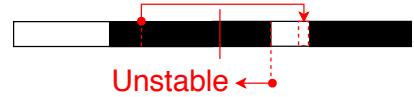
Use $F = "0"$ as an example:



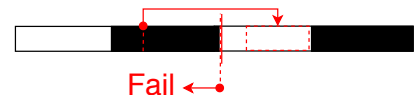
Case ①: any single bit in B flipped



Case ②: any single bit in A flipped



Case ③: any θ bits in A flipped



F is unaffected if the size of segment A is larger than n F flips if segment A size falls below n

Figure F.2: D-Norm-based NoisFre: (a) Two boundary conditions are illustrated: when $F = '0'$ where $(h - l = \theta) \wedge ([h] > [l])$; and when $F = '1'$ where $(h - l = \theta) \wedge ([l] > [h])$; (b) we use $F = '0'$ as an example to demonstrate the influence of flipped raw bits on their transformed 1-bit F . The two ℓ_1 -Norms are firstly reshaped to a single row as shown above to backtrack to the same formulation strategy in S-Norm. The reshaped ℓ_1 -Norm is partitioned into two segments: A and B, the size of segment A is $n + \theta$, and the size of B is $n - \theta$ under the boundary condition $F = '0'$. Consider three cases: ① any single raw bit flip in B will enhance the reliability of transformed F bit; otherwise, ② any single raw bit flip in segment A will degrade the reliability of F . The case of F will fail/flip ③ if there are θ or more raw bits flipped in segment A.

F.1 Derivation of Performance Metric Models

the probability of y bits in segment B to be flipped is formulated as $\Pr_{|y| \in B}^{\text{flip}} = \text{binopdf}(y, \frac{n}{2} - \theta, \text{BER}_f)$.

Although bit flip could occur in either segment A or B, consequential BER_F of F bits are opposite: flipped bits in segment A reduction in the margin or potentially increases the BER_F (shown as the dashed boundary line in Figure F.1 that moves toward the left). In contrast, flipped bits in segment B increase the margin or potentially decrease BER_F (the boundary moves toward the right). If the boundary crosses the middle point of $\frac{n}{2}$, the $\|\mathbf{f}_i\|_1$ falls below $\frac{n}{2}$, and consequently, the F bit flipped—exhibiting an error.

Starting from the extreme but straightforward condition—there is no bit flip in segment B (i.e., $y = 0$). the maximum number of erroneous bits that can be tolerated is θ as discussed above. This can be expressed as $P_{\|\mathbf{f}_i\|_1}^{\text{fail}} = \Pr(x - y \geq \theta) = \Pr(x \geq \theta \mid y = 0) = \Pr(x \geq \theta) \times \Pr(y = 0)$, where the term $\Pr(x \geq \theta)$ can be expressed as $1 - \Pr(x < \theta) = 1 - \sum_{x=0}^{\theta} \left(\Pr_{|x| \in A}^{\text{flip}} \right) = 1 - \text{binocdf}(\theta, \lceil \frac{n}{2} \rceil + \theta, \text{BER}_f)$. By substituting $\Pr_{|y| \in B}^{\text{flip}} = \text{binopdf}(0, \lfloor \frac{n}{2} \rfloor - \theta, \text{BER}_f)$ into the $P_{\|\mathbf{f}_i\|_1}^{\text{fail}}$ equation, $P_{\|\mathbf{f}_i\|_1}^{\text{fail}}$ is expressed:

$$P_{\|\mathbf{f}_i\|_1}^{\text{fail}} = \left(1 - \text{binocdf}(\theta, \lceil \frac{n}{2} \rceil + \theta, \text{BER}_f) \right) \times \text{binocdf}(0, \lfloor \frac{n}{2} \rfloor - \theta, \text{BER}_f) \quad (\text{F.1})$$

However, there is more than one case that satisfies $x - y \geq \theta$ for $\{(x, y) : |x| \in A, |y| \in B\}$. Since A and B are finite sets, the combination of x and y are numerable. Another property worth mentioning is $|A| > |B|$. Therefore, the total number of combinations is up bounded by $|B| = \lceil \frac{n}{2} \rceil - \theta$ where ‘ $|$ ’ denotes the cardinality or the size of a set. If we enumerate and sum up all possible combinations, we obtain the complete form of equation (6.8):

$$\text{BER}_F = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - \theta} \left(\left(1 - \text{binocdf}(\theta + i, \lceil \frac{n}{2} \rceil + \theta, \text{BER}_f) \right) \times \text{binopdf}(i, \lfloor \frac{n}{2} \rfloor - \theta, \text{BER}_f) \right) \quad (\text{F.2})$$

F.1.3 Unreliability Formalisation of D-Norm Transformation

As discussed in Section 6.2.2. The transformed bit via D-Norm is determined as below:

$$F \leftarrow \mathcal{T}_{\text{DNorm}}(\mathbf{f}^{n \times m}, \theta) = \begin{cases} 1, & h - l \geq \theta \wedge [h] < [l] \\ 0, & h - l \geq \theta \wedge [h] > [l] \\ \perp, & h - l < \theta \end{cases} \quad (\text{F.3})$$

where $[]$ indicates the index.

To apply the same derivation strategy as the S-Norm, as illustrated in Figure F.2 (b), the two $\ell 1$ -Norms are reshaped into a single row, and four partitions are now rearranged as two segments: A and B. The length of segment A is eventually $h + (n - l)$ by considering the fact $h = l + \theta$, whereas we can see that the length of A is $n + \theta$. The largest number of errors/flips within raw bits \mathbf{f} that still can not result in error or flip to the transformed F bit is $(n + \theta) - n = \theta$.

The rest of the steps are identical to those in S-Norm. Using ① as an example, on the one hand, Case ①, if one raw bit in segment B is flipped, it will increase the tolerance of the number of raw bits in segment A which allows being flipped *without influencing* F bit. On the other hand, Case ② flipping one raw bit in segment A will potentially result in an error to the F bit. Further, for Case ③, supposing that segment B's raw bits remain unchanged, if more than θ raw bits flipped in segment A, the \mathbf{F} will exhibit an error—flipping from ‘0’ to ‘1’. To be precise, the transformed \mathbf{F} *will not exhibit* error unless more than $\theta + i$ raw bits in segment A flip.

Now, an extreme condition is considered as a starting point: as shown in the third column in Figure F.2, we have two words, labelled with spatial index the ‘first’ and the ‘second’. We denote the raw bits as $\mathbf{f}_{\text{first}}$ and $\mathbf{f}_{\text{second}}$, respectively.

In the exemplified case, $\mathbf{f}_{\text{first}}$ has the lowest $\ell 1$ -Norm while the $\mathbf{f}_{\text{second}}$ has the highest $\ell 1$ -Norm in the m -word block. (i.e., $\|\mathbf{f}_{\text{first}}\|_1 = l$, $\|\mathbf{f}_{\text{second}}\|_1 = h$). From the diagram, we can write the following equation:

$$\|\mathbf{f}_{\text{second}}\|_1 - \|\mathbf{f}_{\text{first}}\|_1 = \theta \quad (\text{F.4})$$

By substituting $\|\mathbf{f}_{\text{second}}\|_1 = h$ and $\|\mathbf{f}_{\text{first}}\|_1 = l$ into the equation above (the case ① in Figure F.2 (a)), we obtain:

$$h - l = \theta \quad (\text{F.5})$$

Add n (number of bits in one word/group) to both sides of the equation. We obtain:

$$h + (n - l) = n + \theta \quad (\text{F.6})$$

F.1 Derivation of Performance Metric Models

If we reshuffle the four partitions in Figure F.2 (b), the error rate of D-Norm can be formalised in a similar manner as the S-Norm (equation (6.8)). The margin (denoted as a dashed line) reduces and results in an unstable trend if any bit flips in the segment A. Once the margin crosses n (marked as a solid line) from the right to the left, the transformed \mathbf{F} is, therefore, erroneous. In contrast, bits flipped in segment B increase the margin and stabilise the \mathbf{F} .

The probability of x error bits occurring in segment A can be expressed as $\Pr_{|x| \in A}^{\text{flip}} = \text{binopdf}(x, n + \theta, \text{BER}_f)$. Similarly for y bits in segment B to be flipped can be expressed as $\Pr_{|y| \in B}^{\text{flip}} = \text{binopdf}(y, n - \theta, \text{BER}_f)$.

Now consider the special case where there is no bit flip in segment B; then the highest number of bits allowed to be flipped in segment A is simply θ . Otherwise, the \mathbf{F} will exhibit errors. Consequently, the $P_{\text{DNorm}}^{\text{fail}}$ can be expressed as:

$$P_{\text{DNorm}}^{\text{fail}}(y = 0) = \left(1 - \text{binocdf}(\theta - 1, n + \theta, \text{BER}_f) \right) \times \text{binopdf}(0, n - \theta, \text{BER}_f) \quad (\text{F.7})$$

If the number of flipped bits in segment B is non-zero, $\Pr_{|y| \in B}^{\text{flip}} = \text{binopdf}(y, n - \theta, \text{BER}_f)$, where $y \in [0, |B|]$, $|B| = n - \theta$. In other words, flipped bits in segment B allows more tolerance of error bits in segment A, before \mathbf{F} exhibiting error. Therefore, the D-Norm, BER_F , is the summation of $P_{\text{DNorm}}^{\text{fail}}(y)$ for all possible y , finally formulated as in equation:

$$\text{BER}_F = \sum_{y=0}^{n-\theta} \left(\left(1 - \text{binocdf}(y + \theta - 1, n + \theta, \text{BER}_f) \right) \times \text{binopdf}(y, n - \theta, \text{BER}_f) \right)$$

F.1.4 Extraction Efficiency of S-Norm Transformation

For the S-Norm, if one group/word \mathbf{f} is selected, it must satisfy the selection criteria $\|\mathbf{f}\|_1 \in [0, \lfloor \frac{n}{2} \rfloor - \theta] \cup [\lceil \frac{n}{2} \rceil + \theta, n]$. Hence, the probability of a group being selected can be expressed as:

$$\begin{aligned} P_{\text{SNorm}}^{\text{select}} &= \Pr(\|\mathbf{f}\|_1 \leq \lfloor \frac{n}{2} \rfloor - \theta) + \Pr(\|\mathbf{f}\|_1 \geq \lceil \frac{n}{2} \rceil + \theta) \\ &= \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - \theta} \left(\Pr(\|\mathbf{f}\|_1 = i) \right) + \sum_{k=\lceil \frac{n}{2} \rceil + \theta}^n \left(\Pr(\|\mathbf{f}\|_1 = k) \right) \end{aligned}$$

By substituting $\Pr(\|\mathbf{f}\|_1 \leq i) = \text{binocdf}(i, n, 0.5)$ and $\Pr(\|\mathbf{f}\|_1 \geq k) = 1 - \text{binocdf}(k, n, 0.5)$, we get:

$$P_{\text{SNorm}}^{\text{select}} = \text{binocdf}(\lceil \frac{n}{2} \rceil - \theta - 1, n, 0.5) + \left(1 - \text{binocdf}(\lfloor \frac{n}{2} \rfloor + \theta, n, 0.5) \right) \quad (\text{F.8})$$

The $P_{\text{SNorm}}^{\text{select}}$ formulates the probability that one group is selected under S-Norm. The extraction efficiency η_{SNorm} can be directly expressed via $P_{\text{SNorm}}^{\text{select}}$:

$$\eta_{\text{SNorm}} = \frac{1}{n} \times P_{\text{SNorm}}^{\text{select}} \times (1024 \times 8) \quad (\text{F.9})$$

Where $\frac{1}{n}$ means that a transformed F bit is from n raw bits, The last term 1024×8 is the conversion factor between bit and Kilo bytes (bit/KiB). By substituting $P_{\text{SNorm}}^{\text{select}}$ into η_{SNorm} , we can finally obtain equation (6.10).

$$\eta_{\text{SNorm}} = \frac{1}{n} \times \left(\text{binocdf}(\lceil \frac{n}{2} \rceil - \theta - 1, n, 0.5) + \left(1 - \text{binocdf}(\lfloor \frac{n}{2} \rfloor + \theta, n, 0.5) \right) \right) \times (1024 \times 8) \quad (\text{F.10})$$

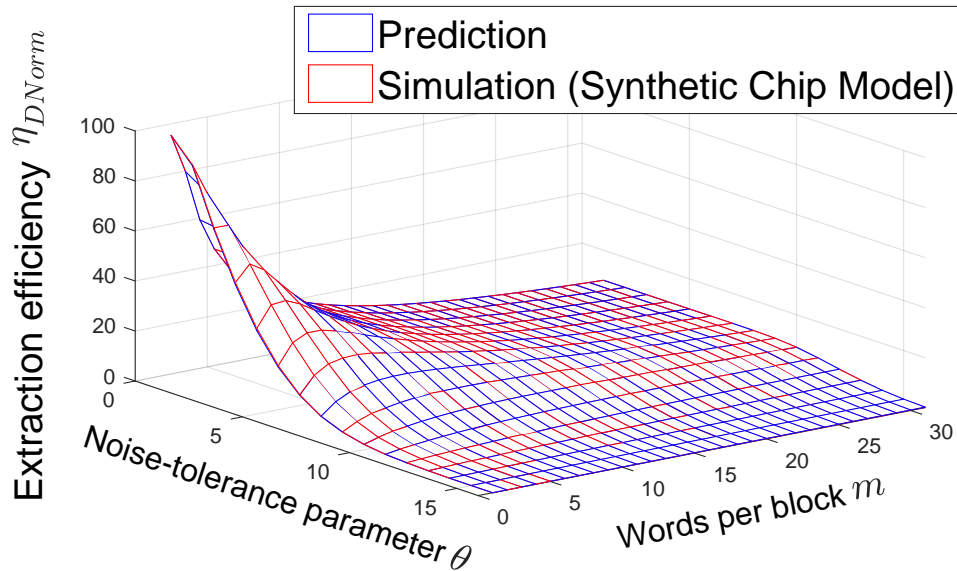


Figure F.3: Validation on equation (6.11) (extraction efficiency of D-Norm) using a simulated chip (the **Simulation** test setting in Section 6.4). Here, $n = 32$, while m and the noise tolerance parameter θ are varied. Overall, the simulation agrees well with the prediction, as two values overlaps.

F.1.5 Extraction Efficiency of D-Norm Transformation

To estimate the extraction efficiency of D-Norm, what we need to do first is estimate the probability that among m groups/words $\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m$, the minimum ℓ_1 -Norm $\|\mathbf{f}_i\|_1$ is any given value a from 0 to n , and the maximum ℓ_1 norm $\|\mathbf{f}_i\|_1$ is another given value z from 0 to n :

$$P(a, z) \triangleq \Pr(l = a \wedge h = z)$$

Recall that:

$$h \triangleq \arg \max_{\mathbf{f}_i | i \in \{1, \dots, m\}} (\|\mathbf{f}_i\|_1)$$

$$l \triangleq \arg \min_{\mathbf{f}_i | i \in \{1, \dots, m\}} (\|\mathbf{f}_i\|_1)$$

Once we comply with the above principle, the $P_{\text{block}}^{\text{select}}$, that one block to be selected for noise-tolerant fingerprint extraction is simply the sum of all $P(a, z)$ over $z - a \geq \theta$.

$$P_{\text{block}}^{\text{select}} = \sum_{a=1}^{n-\theta} \sum_{z=a+\theta}^n P(a, z)$$

$P(a, z)$ is a non-trivial to estimate. Fortunately, we can solve an easier and related problem first:

$$Q(a, z) \triangleq \text{Prob}(l \geq a \wedge h \leq z) = \left(\sum_{i=a}^z \text{binopdf}(i, n, 0.5) \right)^m$$

Another angle to look at $Q(a, z)$ is: What is the probability that among m words in a block with all ℓ_1 -Norm are at least a and at most z ? That question can be answered because it poses an independent question on each word \mathbf{f}_i : is $a \leq \|\mathbf{f}_i\|_1 \leq z$ or not? The answer must be ‘yes’ for all m words, and it is ‘yes’ for a single word with probability $\sum_{i=a}^z \text{binopdf}(i, n, 0.5)$ (the usual formula for the number of $\|\mathbf{f}_i\|_1$ meet θ divided by the number of all m words), and because those events are independent, the probabilities can be consequentially multiplied.

The question becomes: how do we get from $Q(a, z)$ to $P(a, z)$?

Note that:

$$\begin{aligned} & \{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l = a \wedge h = z)\} \\ &= \{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq a \wedge h = z)\} - \{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq a+1 \wedge h = z)\} \end{aligned}$$

because for the l to be equal to a it is equivalent to ask for the l to be at least a but not to be at least $a+1$. In addition, the set we are subtracting is actually a subset of the set we are subtracting

from, so we obtain:

$$\begin{aligned} P(a, z) &= \text{Prob}\{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l = a \wedge h = z)\} \\ &= \text{Pr}\{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq a \wedge h = z)\} - \text{Pr}\{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq a + 1 \wedge h = z)\} \end{aligned}$$

Our two operands are of the same type. We can do the same operation to reduce each probability to something expressible by some $Q(r, s)$:

$$\begin{aligned} P(a, z) &= \{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq r \wedge h = z)\} \\ &= \{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq r \wedge h \leq z)\} - \{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq r \wedge h \leq z - 1)\} \end{aligned}$$

And we obtain:

$$\begin{aligned} &\text{Pr}\{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq r \wedge h = z)\} = \\ &\text{Pr}\{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq r \wedge h \leq z)\} - \text{Pr}\{(\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_m) : (l \geq r \wedge h \leq z - 1)\} \\ &= Q(r, z) - Q(r, z - 1) \end{aligned}$$

And finally, for $P(a, z)$, by substituting this in the above formula:

$$P(a, z) = (Q(a, z) - Q(a, z - 1)) - (Q(a + 1, z) - Q(a + 1, z - 1))$$

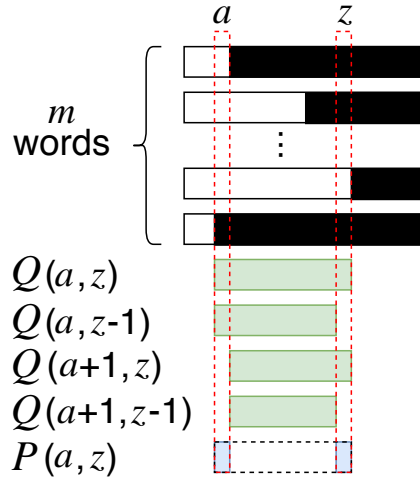


Figure F.4: Showing the relationship between P_{block} and Q terms.

The normalised D-Norm extraction efficiency η_{DNorm} is finally given:

$$\eta_{\text{DNorm}} = \frac{1}{n \times m} \times P_{\text{block}}^{\text{select}} \times (1024 \times 8)$$

To be concise, we keep $P_{\text{block}}^{\text{select}}$, P_{block} and $Q(l, h)$ to be expressed separately. The term of $\frac{1}{n \times m}$ stands for $n \times m$ raw bits producing a 1-bit noise-tolerant bit, per (1024×8) 1 KiB memory. It

F.2 Remote Attestation

tends to be hard to follow when we substitute all terms and write a huge equation. To be concise, we keep expressions $P_{\text{DNorm}}^{\text{select}}$, P_{block} and $Q(l, h)$ separately.

F.2 Remote Attestation

The following description is based on the setting shown in Figure 6.18 where the Prover device implements a secure WORM memory for storing the enrolled **mask**. During the one-time enrolment conducted by the trusted Verifier, we use a cabled JTAG interface and Segger J-link command-line tool to read out the start-up state (fingerprints) of Prover's (Nordic Semiconductor nRF52832) SRAM. Readout raw fingerprints are saved as binary files and then processed (using Matlab) for performing the D-Norm transform and selection (Section 6.2). Such a process produces: i) a database entry containing Prover **id** and selected reliable noise-tolerant F bits; and ii) a C language header file containing the **mask** indicating the memory addresses of raw fingerprint bits to be employed for obtaining F bits to be compiled with the sensor node code. Next, we elaborate on an efficient means for organising the memory addresses defined by the **mask**.

D-Norm Mask. The **mask** first specifies the starting address of the fingerprint zone, which is set to 0x4000, reserving the lower 16 KiB of SRAM space for system run-time operations. To reduce the storage footprint of the **mask**, only the relative offsets between selected memory addresses, rather than the 32-bit absolute addresses are recorded. Once the **mask** is determined, the server computes a MAC **tag** over the **mask** with the derived key **F** for integrity checks.

Implemented System. During the attestation phase, we employ a command-line Verifier tool, ① shown in Figure 6.16 (b), to randomly generate a challenge (nonce). We look up the **DB** according to the Prover's returned **id** and compute the expected response. To visualise the data exchange for *demonstration purposes*, we built our Gateway ② using an Android demo APP based on FastBLE library³⁹ and used the smartphone's built-in Bluetooth-LE interface to communicate with the Prover. In practice, the Gateway could be realised by any base station with a Bluetooth-LE transceiver. The Prover ③ in this case study is a representative low-end sensor node equipped with an ARM-Cortex M4-based nRF52832 Bluetooth-LE SOC. The code to be attested on the Prover is statically allocated with a linker Preprocessor command.⁴⁰ The noise-tolerant fingerprint regeneration function, the **mask**, and the immutable bootloader are placed in WORM memory using an ARM MPU.

³⁹FastBLE is available: <https://github.com/Jasonchenlijian/FastBle>

⁴⁰For example `__attribute__((section('.ARM.__at_0x50000')))` in Keil uVision specifies placing the function at memory starting from address 0x50000.

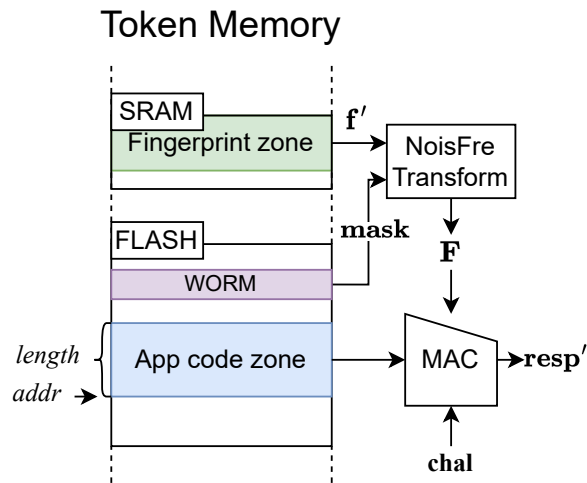


Figure F.5: Memory management and data flow for the remote attestation at the Prover. Notably, the total SRAM memory size is 64 KiB. We only use 48 KiB for the fingerprint zone and reserve 16 KiB for system run-time operations.

Appendix G

Chapter 7 Appendix

G.1 Bit-aliasing Pattern

The bit-aliasing is measured with equation (7.12) to calculate bit frequency for each F bit. We define three regions to describe the bit-aliasing for a given F bit: i) *good* if the bit frequency within 0.5 ± 0.1 ; ii) *intermediate* if the bit frequency fallen outside 0.5 ± 0.1 but within 0.5 ± 0.3 ; and iii) *highly aliased* if bit frequency is beyond 0.5 ± 0.3 . The three regions *good*, *intermediate* and *highly aliased* are colour coded in green, yellow and white, respectively in the plots below.

Our bit-aliasing evaluation is based on the physical chip dataset MSP20: introduced in Section 2.3.4, the bit-aliasing measurement for the Baseline key extraction method recorded mean is 0.50, the standard division is 0.13, and the worst-case is 0.96, as illustrated in Figure G.1. In addition, 78 out of 128 bits fall into the green, *good*, region, 48 bits fall into the yellow, *intermediate*, region, and only 2 bits fall into the white, *highly aliased* region.

For the same MSP20 physical chip dataset in Figure G.2, the Fixed-d method in the bit-aliasing evaluation shows a mean = 0.53, standard division = 0.12 and a worst-case = 1.00 (this implies bit 0 in all 25 tested chips always presented as logic '1'). The detailed statistic indicates 90 good, 35 intermediate and three highly aliased bits.

For the Variable-d method shown in Figure G.3, the mean aliasing value is 0.54, standard division is 0.13 and the worst-case is 0.96. With 85 bits in good, 41 in intermediate, and 2 bits in the highly aliased region.

G.1 Bit-aliasing Pattern

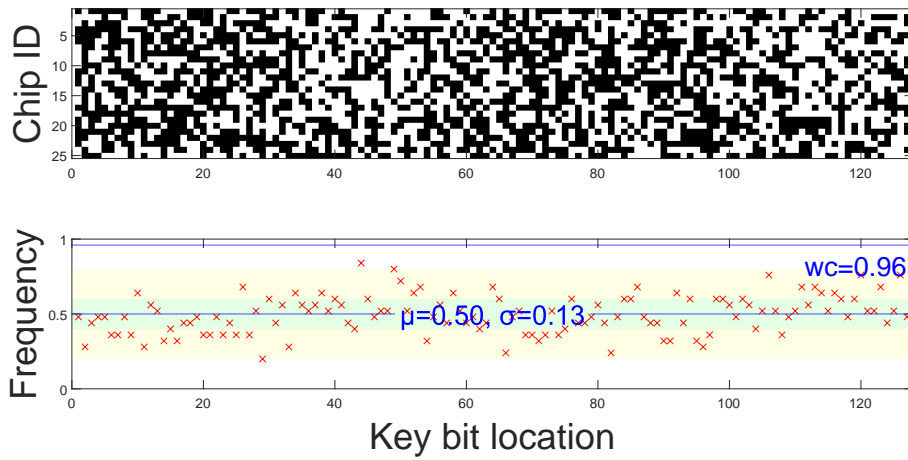


Figure G.1: Evaluating bit-aliasing over 25 MSP430FR5969 chips. The key is extracted with the Baseline method.

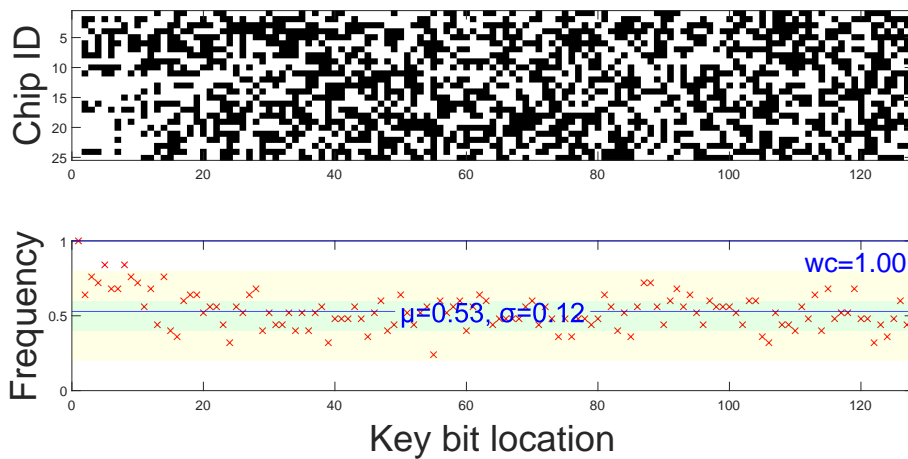


Figure G.2: Evaluating bit-aliasing over 25 MSP430FR5969 chips. The key is extracted with the Fixed-d method.

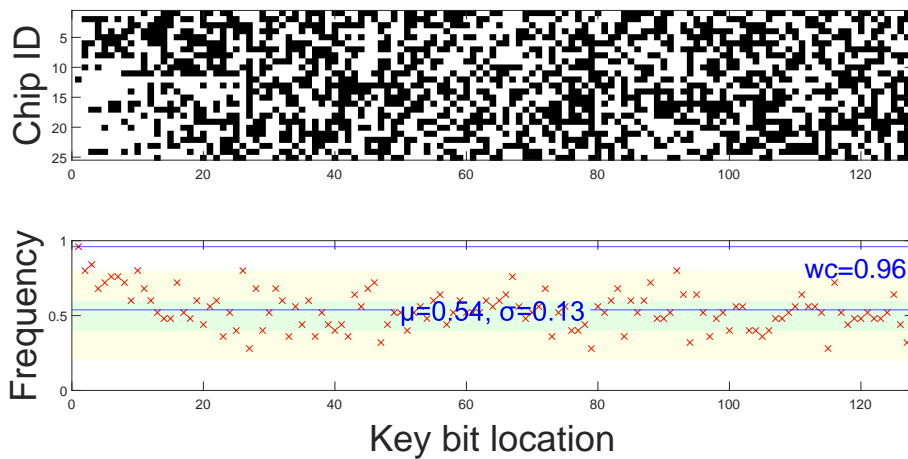


Figure G.3: Evaluating bit-aliasing over 25 MSP430FR5969 chips. The key is extracted with the Variable-d method.

Bibliography

- [1] S. Butler *et al.*, *Macquarie dictionary*. Macquarie Dictionary Publishers, 2017, vol. 5.
- [2] *IEEE reference guide*, IEEE Periodicals, accessed: 2021-02-24, Dec. 2018. [Online]. Available: <https://ieeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf>.
- [3] “IEC 80000-13 international electrotechnical commission quantities and units,” 2008, accessed: 2021-02-24. [Online]. Available: <https://www.iso.org/standard/31898.html>.
- [4] D. Mukhopadhyay, “Pufs as promising tools for security in internet of things,” *IEEE Design & Test*, vol. 33, no. 3, pp. 103–115, 2016.
- [5] H. Jiang, X. Zhou, S. Kulkarni, M. Uranian, R. Seenivasan, and D. A. Hall, “A sub-1 uw multiparameter injectable biomote for continuous alcohol monitoring,” in *2018 IEEE Custom Integrated Circuits Conf. (CICC)*, 2018, pp. 1–4.
- [6] A. Afuang, T. Rago, A. Mukherjee, B. Rojas, and H. Ujhazy, *IoT growth demands rethink of long-term storage strategies, says idc*, Jul. 2020. [Online]. Available: <https://www.idc.com/getdoc.jsp?containerId=prAP46737220>.
- [7] M. Liyanage, A. Braeken, P. Kumar, and M. Ylianttila, *IoT security: Advances in authentication*. John Wiley & Sons, 2020.
- [8] S. Al-Sarawi, M. Anbar, R. Abdullah, and A. B. Al Hawari, “Internet of things market analysis forecasts, 2020–2030,” in *2020 Fourth World Conf. on Smart Trends in Systems, Security and Sustainability (WorldS4)*, IEEE, 2020, pp. 449–453.
- [9] A. Parks, *WISP 5*, GitHub repository, accessed: 2021-02-24, Jul. 2016. [Online]. Available: <https://github.com/wisp/wisp5>.
- [10] Q. H. Dang, S. J. Chen, D. C. Ranasinghe, and C. Fumeaux, “Modular integration of a passive rfid sensor with wearable textile antennas for patient monitoring,” *IEEE Trans. on Comp., Packaging and Manufacturing Tech.*, vol. 10, no. 12, pp. 1979–1988, 2020.
- [11] A. Jayatilaka, Q. H. Dang, S. J. Chen, R. Visvanathan, C. Fumeaux, and D. C. Ranasinghe, “Designing batteryless wearables for hospitalized older people,” in *Proc. the 23rd Intern. Symp. on Wearable Computers*, 2019, pp. 91–95.

BIBLIOGRAPHY

- [12] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, “Intermittent computing: Challenges and opportunities,” in *Summit on Advances in Programming Languages*, 2017.
- [13] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith, “Design of an RFID-based battery-free programmable sensing platform,” *IEEE Trans. Instrum. Meas.*, vol. 57, no. 11, pp. 2608–2615, 2008.
- [14] H. Zhang, J. Gummesson, B. Ransford, and K. Fu, “Moo: A batteryless computational RFID and sensing platform,” *Uni. of Massachusetts Comput. Sci. Technol. Report UM-CS-2011-020*, 2011.
- [15] Farsens, *The spider is an evaluation board for the andy100 uhf rfid ic*. <http://www.farsens.com/en/products/spider-h254/>, accessed: 2021-09-20, 2016.
- [16] A. J. Bandodkar, P. Gutruf, J. Choi, K. Lee, Y. Sekine, J. T. Reeder, W. J. Jeang, A. J. Aranyosi, S. P. Lee, J. B. Model, *et al.*, “Battery-free, skin-interfaced microfluidic/electronic systems for simultaneous electrochemical, colorimetric, and volumetric analysis of sweat,” *Science Advances*, vol. 5, no. 1, eaav3294, 2019.
- [17] R. L. Shinmoto Torres, R. Visvanathan, D. Abbott, K. D. Hill, and D. C. Ranasinghe, “A battery-less and wireless wearable sensor system for identifying bed and chair exits in a pilot trial in hospitalized older people,” *PLOS ONE*, vol. 12, no. 10, pp. 1–25, 2017.
- [18] A. Wickramasinghe, D. C. Ranasinghe, C. Fumeaux, K. D. Hill, and R. Visvanathan, “Sequence learning with passive rfid sensors for real-time bed-egress recognition in older people,” *IEEE journal of biomedical and health informatics*, vol. 21, no. 4, pp. 917–929, 2016.
- [19] J. S. Besnoff, T. Deyle, R. R. Harrison, and M. S. Reynolds, “Battery-free multichannel digital ecg biotelemetry using uhf RFID techniques,” in *In Proc. IEEE Intern. Conf.on RFID*, 2013, pp. 16–22.
- [20] Y. Álvarez López, J. Franssen, G. Álvarez Narciandi, J. Pagnozzi, I. González-Pinto Arrillaga, and F. Las-Heras Andrés, “Rfid technology for management and tracking: E-health applications,” *Sensors*, vol. 18, no. 8, p. 2663, 2018.
- [21] R. Visvanathan, D. C. Ranasinghe, K. Lange, A. Wilson, J. Dollard, E. Boyle, K. Jones, M. Chesser, K. Ingram, S. Hoskins, *et al.*, “Effectiveness of the wearable sensor-based ambient intelligent geriatric management (ambigem) system in preventing falls in older people in hospitals,” *The Journals of Gerontology: Series A*, vol. 77, no. 1, pp. 155–163, 2022.

- [22] H. Aantjes, A. Y. Majid, P. Pawełczak, J. Tan, A. Parks, and J. R. Smith, “Fast Downstream to Many (Computational) RFIDs,” in *Proc. IEEE Intern. Conf. on Comput. Commun.*, 2017.
- [23] D. Marasová, P. Koščák, N. Staricna, S. Mako, and D. Matisková, “Digitization of air transport using smart tires,” in *New Trends in Aviation Development (NTAD)*, 2020, pp. 164–167.
- [24] S. Yang, M. Crisp, R. V. Penty, and I. H. White, “Rfid enabled health monitoring system for aircraft landing gear,” *IEEE J. of Radio Frequency Identification*, vol. 2, no. 3, pp. 159–169, 2018.
- [25] M. D. Santonino III, C. M. Koursaris, and M. J. Williams, “Modernizing the supply chain of airbus by integrating RFID and blockchain processes,” *Intern. J. of Aviation, Aeronautics, and Aerospace*, vol. 5, no. 4, p. 4, 2018.
- [26] M. Buettner, R. Prasad, A. Sample, D. Yeager, B. Greenstein, J. R. Smith, and D. Wetherall, “Rfid sensor networks with the intel wisp,” in *Proc. the 6th ACM Conf. on Embedded network sensor systems*, 2008, pp. 393–394.
- [27] R. L. Shinmoto Torres, R. Visvanathan, D. Abbott, K. D. Hill, and D. C. Ranasinghe, “A battery-less and wireless wearable sensor system for identifying bed and chair exits in a pilot trial in hospitalized older people,” *PloS one*, vol. 12, no. 10, e0185670, 2017.
- [28] G. V. Research. (Jan. 2021). “Internet of things in retail market size report, 2021-2028.” accessed: 2021-02-24, [Online]. Available: www.grandviewresearch.com/industry-analysis/internet-of-things-iot-retail-market.
- [29] J. Tan, P. Pawełczak, A. Parks, and J. R. Smith, “Wisent: Robust downstream communication and storage for computational RFIDs,” in *Proc. Ann. IEEE Intern. Conf. on Comput. Commun.*, 2016, pp. 1–9.
- [30] D. Dinu, A. S. Khrishnan, and P. Schaumont, “SIA: Secure intermittent architecture for off-the-shelf resource-constrained microcontrollers,” in *IEEE Intern. Symp. on Hardware Oriented Secu. and Trust*, 2019, pp. 208–217.
- [31] K. Maeng, A. Colin, and B. Lucia, “Alpaca: Intermittent execution without checkpoints,” *Proc. of the ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 96, 2017.
- [32] D. Wu, L. Lu, M. J. Hussain, S. Li, M. Li, and F. Zhang, “R3: Reliable over-the-air reprogramming on computational RFIDs,” *ACM Trans. on Embedded Comput. Syst.*, vol. 17, no. 1, p. 9, 2018.

BIBLIOGRAPHY

- [33] A. Santhana Krishnan, “Top-down approach to securing intermittent embedded systems,” Ph.D. dissertation, Virginia Tech, 2021.
- [34] A. S. Krishnan and P. Schaumont, “Exploiting security vulnerabilities in intermittent computing,” in *Intern. Conf. on Security, Privacy, and Applied Cryptography Eng.*, 2018, pp. 104–124.
- [35] Y. Su and D. C. Ranasinghe, “Leaving your things unattended is no joke! memory bus snooping and open debug interface exploits,” in *Proc. IEEE Int. Conf. on Pervasive Comput. and Commun. Workshops (PerCom Workshops)*, 2022, pp. 643–648.
- [36] Z. Liu, H. Seo, Z. Hu, X. Hunag, and J. Großschädl, “Efficient implementation of ECDH key exchange for MSP430-based wireless sensor networks,” in *Proc. ACM Symp. on Inf., Comput. and Commun. Security*, 2015, pp. 145–153.
- [37] Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede, “Efficient ring-LWE encryption on 8-bit AVR processors,” in *Intern. Workshop on Crypto. Hardware and Embedded Syst.*, 2015, pp. 663–682.
- [38] Y. Su, Y. Gao, M. Chesser, O. Kavehei, A. Sample, and D. C. Ranasinghe, “Secucode: Intrinsic puf entangled secure wireless code dissemination for computational rfid devices,” *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1699–1717, 2019.
- [39] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, “Sanctuary: Arming trustzone with user-space enclaves,” in *Proc. Network and Distributed Syst. Secur. Symp. (NDSS)*, 2019.
- [40] T. Frassetto, P. Jauernig, C. Liebchen, and A.-R. Sadeghi, “IMIX: In-process memory isolation extension,” in *Proc. 27th USENIX Secur. Symp. (USENIX Security)*, 2018, pp. 83–97.
- [41] D. C. Ranasinghe and P. H. Cole, “Confronting security and privacy threats in modern RFID systems,” in *Proc. IEEE Fortieth Asilomar Conf. on Signals, Systems and Computers*, 2004, pp. 2058–2064.
- [42] P. Tuyls, G.-J. Schrijen, B. Škorić, J. Van Geloven, N. Verhaegh, and R. Wolters, “Read-proof hardware from protective coatings,” in *Intern. Workshop on Crypto. Hardware and Embedded Syst.*, 2006, pp. 369–383.
- [43] S. Larson. (Jan. 2017). “FDA confirms that st. jude’s cardiac devices can be hacked,” [Online]. Available: money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack.

- [44] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, “Smart objects as building blocks for the internet of things,” *IEEE Internet Comput.*, vol. 14, no. 1, pp. 44–51, 2010.
- [45] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, “Internet of things for smart cities,” *IEEE Internet of Things J.*, vol. 1, no. 1, pp. 22–32, 2014.
- [46] T. S. López, D. C. Ranasinghe, B. Patkai, and D. McFarlane, “Taxonomy, technology and applications of smart objects,” *Inf. Syst. Frontiers*, vol. 13, no. 2, pp. 281–300, 2011.
- [47] Z. Ning and F. Zhang, “Understanding the security of arm debugging features,” in *Proc. IEEE Symp. on Secur. and Privacy (S&P)*, 2019, pp. 602–619.
- [48] F. Xu, W. Diao, Z. Li, J. Chen, and K. Zhang, “Badbluetooth: Breaking android security mechanisms via malicious bluetooth peripherals.,” in *Proc. Network and Distributed Syst. Secur. Symp. (NDSS)*, 2019.
- [49] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, “Fpga intrinsic pufs and their use for ip protection,” in *CHES*, vol. 4727, 2007, pp. 63–80.
- [50] D. E. Holcomb, W. P. Burleson, and K. Fu, “Power-up sram state as an identifying fingerprint and source of true random numbers,” *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1198–1210, 2009.
- [51] A. Van Herrewege, “Lightweight puf-based key and random number generation,” 2015.
- [52] A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, B. Škorić, S. Katzenbeisser, and J. Szefer, “Decay-based DRAM PUFs in commodity devices,” *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 3, pp. 462–475, 2018.
- [53] Y. Wang, W.-k. Yu, S. Wu, G. Malysa, G. E. Suh, and E. C. Kan, “Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints,” in *IEEE Symp. on Secur. Privacy (S&P)*, 2012, pp. 33–47.
- [54] Z. Guo, X. Xu, M. M. Tehranipoor, and D. Forte, “FFD: A framework for fake flash detection,” in *Proc. ACM Ann. Design Autom. Conf (DAC)*, 2017, p. 8.
- [55] C.-H. Chang, Y. Zheng, and L. Zhang, “A retrospective and a look forward: Fifteen years of physical unclonable function advancement,” *IEEE Circuits and Systems Magazine*, vol. 17, no. 3, pp. 32–62, 2017.
- [56] C. Herder, M.-D. Yu, F. Koushanfar, and S. Devadas, “Physical unclonable functions and applications: A tutorial,” *Proc. the IEEE*, vol. 102, no. 8, pp. 1126–1141, 8 2014.
- [57] U. Rührmair and M. van Dijk, “Pufs in security protocols: Attack models and security evaluations,” in *Security and Privacy (SP), 2013 IEEE Symp. on*, 2013, pp. 286–300.

BIBLIOGRAPHY

- [58] D. C. Ranasinghe, D. W. Engels, and P. H. Cole, "Security and privacy solutions for low-cost rfid systems," in *Intelligent Sensors, Sensor Networks and Infor. Processing Conf.*, 2004, pp. 337–342.
- [59] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *Proc. Ann. Design Autom. Conf. (DAC)*, 2007, pp. 9–14.
- [60] D. E. Holcomb, W. P. Burleson, K. Fu, *et al.*, "Initial sram state as a fingerprint and source of true random numbers for rfid tags," in *Proc. the Conf. on RFID Security*, vol. 7, 2007, p. 2.
- [61] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems (CHES)*, Springer, 2009, pp. 332–347.
- [62] A. Aysu, E. Gulcan, D. Moriyama, P. Schaumont, and M. Yung, "End-to-end design of a puf-based privacy preserving authentication protocol," in *Intern. Workshop on Crypto. Hardware and Embedded Syst. (HOST)*, 2015, pp. 556–576.
- [63] S. Zeitouni, Y. Oren, C. Wachsmann, P. Koeberl, and A.-R. Sadeghi, "Remanence decay side-channel: The puf case," *IEEE Trans. Information Forensics and Security*, vol. 11, no. 6, pp. 1106–1116, 2015.
- [64] M. Cortez, S. Hamdioui, V. van der Leest, R. Maes, and G.-J. Schrijen, "Adapting voltage ramp-up time for temperature noise reduction on memory-based pufs," in *IEEE Intern. Symp. on Hardware-Oriented Secu. and Trust (HOST)*, 2013, pp. 35–40.
- [65] L. Zhang, Z. H. Kong, and C.-H. Chang, "PCKGen: A phase change memory based cryptographic key generator," in *Proc. of IEEE Intern. Symp. on Circuits and Systems (ISCAS)*, 2013, pp. 1444–1447.
- [66] A. Van Herrewege, S. Katzenbeisser, R. Maes, R. Peeters, A.-R. Sadeghi, I. Verbauwhede, and C. Wachsmann, "Reverse fuzzy extractors: Enabling lightweight mutual authentication for puf-enabled rfids.," in *Financial Crypto.*, vol. 7397, 2012, pp. 374–389.
- [67] G. Selimis, M. Konijnenburg, M. Ashouei, J. Huisken, H. de Groot, V. van der Leest, G.-J. Schrijen, M. van Hulst, and P. Tuyls, "Evaluation of 90nm 6T-SRAM as Physical Unclonable Function for secure key generation in wireless sensor nodes," in *In Proc. IEEE Intern. Symp. on Circuits and Syst.*, 2011, pp. 567–570.
- [68] R. Maes and V. van der Leest, "Countering the effects of silicon aging on SRAM pufs," in *Proc. IEEE Intern. Symp. on Hardware-Oriented Secu. and Trust*, 2014, pp. 148–153.

- [69] D. Ranasinghe, D. Lim, S. Devadas, D. Abbott, and P. Cole, "Random numbers from metastability and thermal noise," *Electronics Letters*, vol. 41, no. 16, pp. 891–893, 2005.
- [70] X. Boyen, "Reusable cryptographic fuzzy extractors," in *Proc. the 11th ACM Conf. on Computer and communications security*, 2004, pp. 82–91.
- [71] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Intern. Conf. on the theory and applications of cryptographic techniques*, 2004, pp. 523–540.
- [72] Y. Gao, Y. Su, L. Xu, and D. C. Ranasinghe, "Lightweight (reverse) fuzzy extractor with multiple reference PUF responses," *IEEE Trans. on Inf. Forensics and Security*, vol. 14, no. 7, pp. 1887–1901, 2018.
- [73] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper data algorithms for PUF-based key generation: Overview and analysis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 889–902, 2015.
- [74] G. T. Becker, "Robust fuzzy extractors and helper data manipulation attacks revisited: Theory vs practice," *IEEE Trans. Dependable and Secure Computing*, vol. 16, no. 5, pp. 783–795, 2017.
- [75] Y. Su, M. Chesser, Y. Gao, A. Sample, and D. Ranasinghe, "Wisecr: Secure simultaneous code dissemination to many batteryless computational rfid devices," *Trans. Dependable Secur. Comput (Early Access)*, pp. 1–18, 2022.
- [76] Y. Gao, Y. Su, L. Xu, and D. C. Ranasinghe, "Lightweight (reverse) fuzzy extractor with multiple reference puf responses," *IEEE Trans. Information Forensics and Security*, vol. 14, no. 7, pp. 1887–1901, 2019. DOI: 10.1109/TIFS.2018.2886624.
- [77] Y. Gao, Y. Su, S. Nepal, and D. C. Ranasinghe, "Noisfre: Noise-tolerant memory fingerprints from commodity devices for security functions," *IEEE Trans. on Dependable and Secure Comput.*, vol. Early Access, 2022.
- [78] D. C. Ranasinghe, M. Sheng, and S. Zeadally, "Unique radio innovation for the 21st century: Building scalable and global rfid networks," 2010.
- [79] Q. F. Hassan, *Internet of things A to Z: technologies and applications*. John Wiley & Sons, 2018.
- [80] D. Ranasinghe and P. Cole, *Networked rfid systems and lightweight cryptography, chapter 8 an evaluation framework*, 2008.

BIBLIOGRAPHY

- [81] T. Yang, J. Xiang, Y. Wang, X. Tan, J. Wang, N. Yan, L. Zheng, and H. Min, “An active tag using carrier recovery circuit for epc gen2 passive uhf rfid systems,” *IEEE Trans. Industrial Electronics*, vol. 65, no. 11, pp. 8925–8935, 2018.
- [82] J. R. Smith, *Wirelessly powered sensor networks and computational RFID*. Springer Science & Business Media, 2013.
- [83] D. Fabbri, E. Berthet-Bondet, D. Masotti, A. Costanzo, D. Dardari, and A. Romani, “Long range battery-less uhf-rfid platform for sensor applications,” in *IEEE Intern. Conf. on RFID Technology and App. (RFID-TA)*, 2019, pp. 80–85.
- [84] A. Wickramasinghe, R. L. Shinmoto Torres, and D. C. Ranasinghe, “Recognition of falls using dense sensing in an ambient assisted living environment,” *Pervasive and Mobile Computing*, vol. 34, pp. 14–24, 2017, Pervasive Computing for Gerontechnology.
- [85] A. Jayatilaka and D. C. Ranasinghe, “Real-time fluid intake gesture recognition based on batteryless uhf rfid technology,” *Pervasive and Mobile Computing*, vol. 34, pp. 146–156, 2017, Pervasive Computing for Gerontechnology.
- [86] ———, “Towards unobtrusive real-time fluid intake monitoring using passive uhf rfid,” in *In Proc. IEEE Intern. Conf. on RFID*, 2016, pp. 1–4.
- [87] D. C. Ranasinghe, R. L. Shinmoto Torres, A. P. Sample, J. R. Smith, K. Hill, and R. Visvanathan, “Towards falls prevention: A wearable wireless and battery-less sensing and automatic identification tag for real time monitoring of human movements,” in *Intern. Conf. of the IEEE Eng. in Medic. and Bio. Society (EMBS)*, 2012, pp. 6402–6405.
- [88] J. Gummeson, J. Mccann, C. (Yang, D. Ranasinghe, S. Hudson, and A. Sample, “Rfid light bulb: Enabling ubiquitous deployment of interactive rfid systems,” *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 2, Jun. 2017.
- [89] C. Jin and M. van Dijk, “Secure and efficient initialization and authentication protocols for shield,” *IEEE Trans. Dependable and Secure Computing*, 2017.
- [90] R. Want, “An introduction to rfid technology,” *IEEE pervasive computing*, vol. 5, no. 1, pp. 25–33, 2006.
- [91] EPCglobal, “Gs1 low level reader protocol (llrp) standard,” vol. 2.0, 2021.
- [92] E. Global, *Epc radio-frequency identity protocols generation-2, uhf rfid standard, specification for rfid air interface protocol for, communications at 860 mhz-960 mhz*, 2018.

-
- [93] M. Philipose, J. Smith, B. Jiang, A. Mamishev, S. Roy, and K. Sundara-Rajan, "Battery-free wireless identification and sensing," *IEEE Pervasive Computing*, vol. 4, no. 1, pp. 37–45, 2005. DOI: 10.1109/MPRV.2005.7.
- [94] M. Chesser, A. Jayatilaka, R. Visvanathan, C. Fumeaux, A. Sample, and D. C. Ranasinghe, "Super low resolution rf powered accelerometers for alerting on hospitalized patient bed exits," in *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2019, pp. 1–10.
- [95] Y. Su, A. Wickramasinghe, and D. C. Ranasinghe, "Investigating sensor data retrieval schemes for multi-sensor passive RFID tags," in *Proc. IEEE Intern. Conf. RFID (IEEE RFID)*, 2015, pp. 158–165.
- [96] M. Sun, S. F. Al-Sarawi, P. Ashenden, M. Cavaiuolo, and D. C. Ranasinghe, "A fully integrable hybrid power management unit for passive UHF RFID," in *IEEE Intern. Conf. on RFID (IEEE RFID)*, 2017, pp. 198–204.
- [97] H. T. Friis, "The free space transmission equation," *Proc. IRE*, vol. 34, p. 254, 1946.
- [98] P. V. Nikitin, R. Martinez, S. Ramamurthy, H. Leland, G. Spiess, and K. Rao, "Phase based spatial identification of UHF RFID tags," in *Intern. Conf. on RFID*, 2010, pp. 102–109.
- [99] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A puf taxonomy," *Applied Physics Reviews*, vol. 6, no. 1, p. 011 303, 2019.
- [100] C. Wachsmann and A.-R. Sadeghi, "Physically unclonable functions (pufs): Applications, models, and future directions," *Synthesis Lectures on Information Security, Privacy, & Trust*, vol. 5, no. 3, pp. 1–91, 2014.
- [101] Y. Su, J. Holleman, and B. Otis, "A 1.6 pj/bit 96% stable chip-id generating circuit using process variations," in *Solid-State Circuits Conf., 2007. ISSCC 2007. Digest of Technical Papers. IEEE Intern.*, 2007, pp. 406–611.
- [102] R. Maes, P. Tuyls, and I. Verbauwhede, "Intrinsic pufs from flip-flops on reconfigurable devices," in *3rd Benelux Workshop on Inf. and System Security (WISSec)*, vol. 17, 2008.
- [103] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The butterfly puf protecting ip on every fpga," in *Hardware-Oriented Secu. and Trust, 2008. HOST 2008. IEEE Intern. Workshop on*, 2008, pp. 67–70.
- [104] P. Simons, E. van der Sluis, and V. van der Leest, "Buskeeper PUFs, a promising alternative to D flip-flop PUFs," in *IEEE Intern. Symp. on Hardware-Oriented Secu. and Trust (HOST)*, IEEE, 2012, pp. 7–12.
-

BIBLIOGRAPHY

- [105] U. Rührmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, and G. Csaba, “Applications of high-capacity crossbar memories in cryptography,” *IEEE Trans. Nanotechnology*, vol. 10, no. 3, pp. 489–498, 2011.
- [106] L. Zhang, X. Fong, C.-H. Chang, Z. H. Kong, and K. Roy, “Highly reliable memory-based physical unclonable function using spin-transfer torque mram,” in *Intern. Symp. on Circuits and Systems (ISCAS)*, IEEE, 2014, pp. 2169–2172.
- [107] O. Kavehei, C. Hosung, D. Ranasinghe, and S. Skafidas, “Mrpuf: A memristive device based physical unclonable function,” *arXiv preprint arXiv:1302.2191*, 2013.
- [108] J. Kim, T. Ahmed, H. Nili, J. Yang, D. S. Jeong, P. Beckett, S. Sriram, D. C. Ranasinghe, and O. Kavehei, “A physical unclonable function with redox-based nanoionic resistive memory,” *IEEE Trans. Inf. Forensics and Security*, vol. 13, no. 2, pp. 437–448, 2017.
- [109] Y. Gao and D. C. Ranasinghe, “R3puf: A highly reliable memristive device based reconfigurable puf,” *arXiv preprint arXiv:1702.07491*, 2017.
- [110] Y. Gao, D. C. Ranasinghe, S. F. Al-Sarawi, O. Kavehei, and D. Abbott, “Emerging physical unclonable functions with nanotechnology,” *IEEE access*, vol. 4, pp. 61–80, 2016.
- [111] A. A. Khan, F. Hameed, and J. Castrillon, “Nvmain extension for multi-level cache systems,” in *Proc. the Rapido’18 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, 2018, pp. 1–6.
- [112] R. Maes, “Physically Unclonable Functions: Constructions, Properties and Applications (Fysisch onkloonbare functies: Constructies, eigenschappen en toepassingen),” 2012.
- [113] U. Rührmair, S. Devadas, and F. Koushanfar, “Security based on physical unclonability and disorder,” in *Introduction to Hardware Secu. and Trust*, Springer, 2012, pp. 65–102.
- [114] R. Maes, P. Tuyls, and I. Verbauwhede, “A soft decision helper data algorithm for SRAM PUFs,” in *IEEE Intern. Symp. on Info. Theory*, 2009, pp. 2101–2105.
- [115] X. Xu, A. Rahmati, D. E. Holcomb, K. Fu, and W. Burlison, “Reliable physical unclonable functions using data retention voltage of SRAM cells,” *IEEE Trans. Comput.-Aided Design of Integr. Circuits and Syst.*, vol. 34, no. 6, pp. 903–914, 2015.
- [116] K.-H. L. Loh, “1.2 fertilizing aiot from roots to leaves,” in *2020 IEEE Intern. Solid-State Circuits Conf. (ISSCC)*, IEEE, 2020, pp. 15–21.

- [117] H. Johari and F. Ayazi, "High-density embedded deep trench capacitors in silicon with enhanced breakdown voltage," *IEEE Trans. Components and Packaging Technologies*, vol. 32, no. 4, pp. 808–815, 2009.
- [118] I. Bhati, M. Chang, Z. Chishti, S. Lu, and B. Jacob, "Dram refresh mechanisms, penalties, and trade-offs," *IEEE Trans. Computers*, vol. 65, no. 01, pp. 108–121, Jan. 2016, ISSN: 1557-9956. DOI: 10.1109/TC.2015.2417540.
- [119] F. Tehranipoor, N. Karimian, W. Yan, and J. A. Chandy, "Dram-based intrinsic physically unclonable functions for system-level security and authentication," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1085–1097, 2016.
- [120] A. Schaller, W. Xiong, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser, and J. Szefer, "Intrinsic rowhammer pufs: Leveraging the rowhammer effect for improved security," in *2017 IEEE Intern. Symp. on Hardware Oriented Secu. and Trust (HOST)*, IEEE, 2017, pp. 1–7.
- [121] S. Jia, L. Xia, Z. Wang, J. Lin, G. Zhang, and Y. Ji, "Extracting robust keys from nand flash physical unclonable functions," in *Intern. Conf. on Information Security*, Springer, 2015, pp. 437–454.
- [122] N. Semiconductor, *Nrf52832 product specification v1. 4*, 2016.
- [123] A. Maiti, V. Gunreddy, and P. Schaumont, "A systematic method to evaluate and compare the performance of physical unclonable functions," in *Embedded systems design with FPGAs*, 2013, pp. 245–267.
- [124] R. Maes, A. Van Herrewege, and I. Verbauwhede, "Pufky: A fully functional puf-based cryptographic key generator," *Cryptographic Hardware and Embedded Systems—CHES 2012*, pp. 302–319, 2012.
- [125] M.-D. Yu, M. Hiller, J. Delvaux, R. Sowell, S. Devadas, and I. Verbauwhede, "A lockdown technique to prevent machine learning on PUFs for lightweight authentication," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 2, no. 3, pp. 146–159, 2016.
- [126] R. Maes, "An accurate probabilistic reliability model for silicon PUFs," in *Cryptographic Hardware and Embedded Systems*, Springer, 2013, pp. 73–89.
- [127] S. Lim, B. Song, and S.-O. Jung, "Highly independent mtj-based puf system using diode-connected transistor and two-step postprocessing for improved response stability," *Trans. on Info. Forensics and Security*, vol. 15, pp. 2798–2807, 2020.

BIBLIOGRAPHY

- [128] Y. Gao, D. C. Ranasinghe, S. F. Al-Sarawi, O. Kavehei, and D. Abbott, “Memristive crypto primitive for building highly secure physical unclonable functions,” *Scientific reports*, vol. 5, 2015.
- [129] G. T. Becker, “On the pitfalls of using arbiter-pufs as building blocks,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Syst.*, vol. 34, no. 8, pp. 1295–1307, 2015.
- [130] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, “Fuzzy extractors: How to generate strong keys from biometrics and other noisy data,” *SIAM J. on computing*, vol. 38, no. 1, pp. 97–139, 2008.
- [131] J. Delvaux and I. Verbauwhede, “Attacking PUF-based pattern matching key generators via helper data manipulation,” in *Topics in Cryptology–CT-RSA*, Springer, 2014, pp. 106–131.
- [132] M. Hofer and C. Boehm, “An alternative to error correction for sram-like pufs,” *Cryptographic Hardware and Embedded Systems, CHES 2010*, pp. 335–350, 2010.
- [133] C. Böhm and M. Hofer, “Physical unclonable functions in theory and practice,” 2013.
- [134] A. CH, *Ch logitech mx master 3 vs. 2s teardown*, Dec. 2019. [Online]. Available: <https://www.cyberhero.tv/blog/2019/12/18/ch-logitech-mx-master-3-vs-2s-teardown/>.
- [135] B. Beeler. (Apr. 2010). “Intel x25-m ssd review (160gb),” [Online]. Available: <https://www.storagereview.com/review/intel-x25-m-ssd-review-160gb>.
- [136] T. Gravekamp. (Dec. 2018), [Online]. Available: <https://www.thomas-gravekamp.nl/2018/12/09/teardown-3-dell-poweredge-2161ds-2/>.
- [137] G. Slot1. (Dec. 2014). “How to tsop xbox easiest method 2020 v1.0 to 1.5,” [Online]. Available: <https://www.youtube.com/watch?v=gIw8k5areQg>.
- [138] Z. Guo, X. Xu, M. T. Rahman, M. M. Tehranipoor, and D. Forte, “SCARe: An SRAM-based countermeasure against IC recycling,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 26, no. 4, pp. 744–755, 2017.
- [139] Y. H. Jung, J. U. Kim, J. S. Lee, J. H. Shin, W. Jung, J. Ok, and T.-i. Kim, “Injectable biomedical devices for sensing and stimulating internal body organs,” *Advanced Materials*, vol. 32, no. 16, p. 1907478, 2020.

- [140] D. Wu, L. Lu, M. J. Hussain, S. Li, M. Li, and F. Zhang, “R3: Reliable Over-the-Air reprogramming on computational RFIDs,” *ACM Trans. Embedded Comput. Syst.*, vol. 17, no. 1, p. 9, 2017.
- [141] R. Brown and K. Pier. (Dec. 2016). “MSP430FRBoot – Main Memory Bootloader and Over-the-Air Updates for MSP430™ FRAM Large Memory Model,” [Online]. Available: <http://www.ti.com/lit/an/slaa721b/slaa721b.pdf>.
- [142] F. Kohnhäuser and S. Katzenbeisser, “Secure Code Updates for Mesh Networked Commodity Low-End Embedded Devices,” in *Proc. Springer Eur. Symp. Res. in Comput. Secur. (ESORICS)*, 2016, pp. 320–338.
- [143] Parks. (Mar. 2016). “Welcome to the WISP 5 wiki!” [Online]. Available: <https://wisp5.wikispaces.com/WISP+Home>.
- [144] C. O. Akash Patel. (Jan. 2017). “Random Number Generation Using MSP430FR59xx and MSP430FR69xx Microcontrollers,” [Online]. Available: <http://www.ti.com/lit/an/slaa725/slaa725.pdf>.
- [145] E. Barker, J. Kelsey, *et al.*, “NIST special publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators,” 2012.
- [146] S. Callegari, R. Rovatti, and G. Setti, “Embeddable ADC-based true random number generator for cryptographic applications exploiting nonlinear signal processing and chaos,” *IEEE Trans. Signal Process.*, vol. 53, no. 2, pp. 793–805, 2005.
- [147] B. Lampert, R. S. Wahby, S. Leonard, and P. Levis, “Robust, low-cost, auditable random number generation for embedded system security,” *IACR Crypto. ePrint Archive*, vol. 2016, p. 884, 2016.
- [148] C. S. Petrie and J. A. Connelly, “A noise-based IC random number generator for applications in cryptography,” *IEEE Trans. Circuits and Syst. I: Fundamental Theory and App.*, vol. 47, no. 5, pp. 615–621, 2000.
- [149] Y. Shen, L. Tian, and H. Zou, “Practical quantum random number generator based on measuring the shot noise of vacuum states,” *Physical Review A*, vol. 81, no. 6, p. 063 814, 2010.
- [150] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo, “A high-speed oscillator-based truly random number source for cryptographic applications on a smart card IC,” *IEEE Trans. Comput.*, vol. 52, no. 4, pp. 403–409, 2003.
- [151] Y. Yan, E. Oswald, and T. Tryfonas, “Cryptographic randomness on a CC2538: A case study,” in *Proc. IEEE Intern. Workshop on Inf. Forensics and Security*, 2016, pp. 1–6.

BIBLIOGRAPHY

- [152] J. Delvaux, D. Gu, I. Verbauwhede, M. Hiller, and M.-D. M. Yu, “Efficient fuzzy extraction of PUF-induced secrets: Theory and applications,” in *Proc. Springer Intern. Conf. on Crypto. Hardware and Embedded Syst.*, 2016, pp. 412–431.
- [153] C. Böhm and M. Hofer, “Puf with preselection,” in *Physical Unclonable Functions in Theory and Practice*, Springer, 2013, pp. 239–248.
- [154] M. Cortez, A. Dargar, S. Hamdioui, and G.-J. Schrijen, “Modeling SRAM start-up behavior for physical unclonable functions,” in *Proc. IEEE Intern. Symp. on Defect and Fault Tolerance in VLSI and Nanotech. Syst.*, 2012, pp. 1–6.
- [155] R. Maes, V. Leest, E. Sluis, and F. Willems, “Secure key generation from biased pufs,” in *Proc. Springer Intern. Workshop on Crypto. Hardware and Embedded Syst.*, 2015, pp. 517–534.
- [156] A. Aysu, Y. Wang, P. Schaumont, and M. Orshansky, “A new maskless debiasing method for lightweight physical unclonable functions,” in *Proc. IEEE Intern. Symp. on Hardware Oriented Secu. and Trust*, 2017, pp. 134–139.
- [157] D. Lemire and O. Kaser, “Faster 64-bit universal hashing using carry-less multiplications,” *J. of Crypto. Eng.*, vol. 6, no. 3, pp. 171–185, 2016.
- [158] K. Maeng, A. Colin, and B. Lucia, “Alpaca: Intermittent execution without checkpoints,” *Proc. ACM Program. Lang.*, vol. 1, 96:1–96:30, 2017, ISSN: 2475-1421.
- [159] B. Ransford, J. Sorber, and K. Fu, “Mementos: System support for long-running computation on rfid-scale devices,” in *ACM SIGARCH Comput. Architecture News*, vol. 39, 2011, pp. 159–170.
- [160] M. Salajegheh, S. Clark, B. Ransford, K. Fu, and A. Juels, “CCCP: Secure remote storage for computational RFIDs,” in *Proc. USENIX Secur.*, 2009.
- [161] M. Hicks, “Clank: Architectural support for intermittent computation,” in *ACM/IEEE Intern. Symp. on Comput. Architecture (ISCA)*, 2017, pp. 228–240.
- [162] M. Buettner, B. Greenstein, and D. Wetherall, “Dewdrop: An energy-aware runtime for computational RFID,” in *Proc. USENIX NSDI*, 2011, pp. 197–210.
- [163] T. Instruments. (Mar. 2017). “MSP430FR59xx Datasheet,” [Online]. Available: <http://www.ti.com/lit/ds/symlink/msp430fr5969.pdf>.
- [164] K. Pier, “Msp code protection features,” *Dostopno na: http://www.ti.com/lit/an/slaa685/slaa685.pdf (2015, pridobljeno 7. 9. 2018)*,

- [165] M. Chesser. (Oct. 2018). “SecuCode GitHub Repository,” [Online]. Available: <https://github.com/AdelaideAuto-IDLab/SecuCode>.
- [166] T. Eisenbarth, S. Heyse, I. von Maurich, T. Poeppelmann, J. Rave, C. Reuber, and A. Wild, “Evaluation of SHA-3 candidates for 8-bit embedded processors,” in *The Second SHA-3 Candidate Conf.*, 2010.
- [167] L. Kusters, T. Ignatenko, F. M. Willems, R. Maes, E. van der Sluis, and G. Selimis, “Security of helper data schemes for SRAM-PUF in multiple enrollment scenarios,” in *Proc. IEEE Intern. Symp. on Inf. Theory*, 2017, pp. 1803–1807.
- [168] R. Maes, *Physically Unclonable Functions*. Springer.
- [169] G. T. Becker, “Robust Fuzzy Extractors and Helper Data Manipulation Attacks Revisited: Theory vs Practice,” 2017.
- [170] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, “A survey on lightweight entity authentication with strong PUFs,” *ACM Comput. Surveys*, vol. 48, no. 2, p. 26, 2015.
- [171] B. Ransford, “A rudimentary bootloader for computational RFIDs,” 2010.
- [172] W. Yang, D. Wu, M. J. Hussain, and L. Lu, “Wireless firmware execution control in computational RFID systems,” in *Proc. IEEE Intern. Conf. on RFID*, 2015, pp. 129–136.
- [173] D. Wu, M. J. Hussain, S. Li, and L. Lu, “R2: Over-the-air reprogramming on computational RFIDs,” in *Proc. IEEE Intern. Conf. on RFID*, 2016, pp. 1–8.
- [174] R. Prasad, M. Buettner, B. Greenstein, and D. Wetherall, “Wisp monitoring and debugging,” in *Wirelessly Powered Sensor Networks and Computational RFID*, 2013, pp. 157–172.
- [175] L. Reynoso. (Jun. 2013). “Mspboot – Main Memory Bootloader for MSP430™ Microcontrollers,” [Online]. Available: <http://www.ti.com/lit/an/slaa600d/slaa600d.pdf>.
- [176] FAA. (Nov. 2018). “Airworthiness approval of installed radio frequency identification (RFID) tags and sensors,” [Online]. Available: www.faa.gov/regulations_policies/advisory_circulars/index.cfm/go/document.information/documentid/1034630.
- [177] W. Feng, Y. Qin, S. Zhao, Z. Liu, X. Chu, and D. Feng, “Secure code updates for smart embedded devices based on PUFs,” in *Intern. Conf. on Crypto. and Net. Security*, 2017, pp. 325–346.

BIBLIOGRAPHY

- [178] F. Piessens and I. Verbauwhede, “Software security: Vulnerabilities and countermeasures for two attacker models,” in *Proc. Conf. on Design, Autom. & Test in Europe*, 2016, pp. 990–999.
- [179] M. Sammler, R. Lepigre, R. Krebbers, K. Memarian, D. Dreyer, and D. Garg, “Refinedc: Automating the foundational verification of c code with refined ownership types,” in *Intern. Conf. Program. Language Design Implementation*, 2021, pp. 158–174.
- [180] A. Levy, B. Campbell, B. Ghena, D. B. Giffin, P. Pannuto, P. Dutta, and P. Levis, “Multiprogramming a 64kb computer safely and efficiently,” in *Symp. Operat. Syst. Principles*, 2017, pp. 234–251.
- [181] G. EPCglobal, “Inc., “EPCTM Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860MHz-960MHz Version 2.0. 1,”” EPCGlobal Inc., Technal. Rep., April 2015. [Online]. Available: <http://www.gs1.org/epcrfid/epc-rfid-uhf-air-interface-protocol/2-0-1>, Tech. Rep., 2015.
- [182] P. Thanigai and W. Goh, “Msp430 fram quality and reliability,” *Texas Instruments, SLAA526A*, 2014.
- [183] K. Fischer, “Advancements in control system data authentication and verification,” in *ASNE Intelligent Ships Symp.*, 2017, pp. 25–25.
- [184] A. Elmangoush, R. Steinke, T. Magedanz, A. A. Corici, A. Bourreau, and A. Al-Hezmi, “Application-derived communication protocol selection in m2m platforms for smart cities,” in *Intern. Conf. Intell. in Next Gen. Netw.*, IEEE, 2015, pp. 76–82.
- [185] B. Buhrow, P. Riemer, M. Shea, B. Gilbert, and E. Daniel, “Block cipher speed and energy efficiency records on the MSP430: System design trade-offs for 16-bit embedded applications,” in *Intern. Conf. on Crypto. and Inf. Security in Latin America*, 2014.
- [186] D. Dinu, Y. L. Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, “Triathlon of lightweight block ciphers for the internet of things,” IACR ePrint archive, Tech. Rep. 3, 2015, pp. 283–302.
- [187] T. Instruments, “MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User’s Guide,” *Texas Instruments*, 2014.
- [188] —, (Jun. 2014). “Msp430 fram technology,” [Online]. Available: <http://www.ti.com/lit/an/slaa628/slaa628.pdf>.
- [189] P.-A. Fouque, J. Jean, and T. Peyrin, “Structural evaluation of aes and chosen-key distinguisher of 9-round aes-128,” in *Springer Ann. Cryptology Conf.*, 2013, pp. 183–203.

- [190] A. Biryukov and D. Khovratovich, “Related-key cryptanalysis of the full aes-192 and aes-256,” in *Springer Intern. conf. on the theory and application of cryptology and info. security*, 2009, pp. 1–18.
- [191] B. Lee and J.-H. Lee, “Blockchain-based secure firmware update for embedded devices in an internet of things environment,” *J. of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017.
- [192] S. Schulz, A. Schaller, F. Kohnhäuser, and S. Katzenbeisser, “Boot attestation: Secure remote reporting with off-the-shelf iot sensors,” in *Eur. Symp. on Res. in Comput. Secur.*, 2017, pp. 437–455.
- [193] E. Dushku, M. M. Rabbani, M. Conti, L. V. Mancini, and S. Ranise, “Sara: Secure asynchronous remote attestation for iot systems,” *IEEE Trans. Information Forensics and Security*, 2020.
- [194] M. Hiller, “Key derivation with physical unclonable functions,” Ph.D. dissertation, Universität München, 2016.
- [195] C.-E. D. Yin and G. Qu, “LISA: Maximizing RO PUF’s secret extraction,” in *IEEE Intern. Symp. on Hardware-Oriented Secu. and Trust*, IEEE, 2010, pp. 100–105.
- [196] M. Hiller, M.-D. Yu, and G. Sigl, “Cherry-picking reliable PUF bits with differential sequence coding,” *IEEE Trans. Information Forensics and Security*, vol. 11, no. 9, pp. 2065–2076, 2016.
- [197] M. Roel, “Physically unclonable functions: Constructions, properties and applications,” Ph.D. dissertation, University of KU Leuven, 2012.
- [198] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor, “Bit selection algorithm suitable for high-volume production of sram-puf,” in *IEEE Intern. Symp. Hardware-Oriented Secur. and Trust (HOST)*, 2014, pp. 101–106.
- [199] A. Aysu, E. Gulcan, D. Moriyama, and P. Schaumont, “Compact and low-power ASIP design for lightweight PUF-based authentication protocols,” *IET Information Security*, vol. 10, no. 5, pp. 232–241, 2016.
- [200] M.-D. Yu and S. Devadas, “Secure and robust error correction for physical unclonable functions,” *IEEE Des. Test. Comput.*, vol. 27, no. 1, pp. 48–65, 2010.
- [201] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, “Secure key generation from biased PUFs: Extended version,” *J. of Cryptographic Eng.*, vol. 6, no. 2, pp. 121–137, 2016.

BIBLIOGRAPHY

- [202] L. Kusters, T. Ignatenko, F. M.J. Willems, R. Maes, E. Van Der Sluis, and G. Selimis, “Security of helper data schemes for SRAM-PUF in multiple enrollment scenarios,” in *IEEE Intern. Symp. Info. Theory (ISIT)*, 2017, pp. 1803–1807.
- [203] A. Schaller, T. Stanko, B. Skoric, and S. Katzenbeisser, “Eliminating leakage in reverse fuzzy extractors,” *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 4, pp. 954–964, 2017.
- [204] J. Delvaux and I. Verbauwhede, “Key-recovery attacks on various ro PUF constructions via helper data manipulation,” in *Proc. the Conf. Design Autom. Test Eur. (DATE)*, 2014, p. 72.
- [205] S. Puchinger, S. Muelich, M. Bossert, M. Hiller, and G. Sigl, “On error correction for physical unclonable functions,” in *Proc. Intern. ITG Conf. on Systems, Communications and Coding*, 2015.
- [206] V. Van der Leest, B. Preneel, and E. Van der Sluis, “Soft decision error correction for compact memory-based PUFs using a single enrollment,” in *Cryptographic Hardware and Embedded Systems*, Springer, 2012, pp. 268–282.
- [207] J. Delvaux, “Security analysis of PUF-based key generation and entity authentication,” Ph.D. dissertation, University of KU Leuven and ShangHai Jiao Tong University, 2017.
- [208] O. Günlü, O. İşcan, and G. Kramer, “Reliable secret key generation from physical unclonable functions under varying environmental conditions,” in *IEEE Intern. Workshop on Information Forensics and Security*, IEEE, 2015, pp. 1–6.
- [209] M. T. Rahman, A. Hosey, Z. Guo, J. Carroll, D. Forte, and M. Tehranipoor, “Systematic correlation and cell neighborhood analysis of SRAM PUF for robust and unique key generation,” *J. of Hardware and Systems Security*, vol. 1, no. 2, pp. 137–155, 2017.
- [210] R. Maes, *Physically unclonable functions: Constructions, properties and applications*. Springer Science & Business Media, 2013.
- [211] C. Herder, L. Ren, M. Van Dijk, M.-D. Yu, and S. Devadas, “Trapdoor computational fuzzy extractors and stateless cryptographically-secure physical unclonable functions,” *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 1, pp. 65–82, 2016.
- [212] Y. Wang and M. Orshansky, “Efficient helper data reduction in sram pufs via lossy compression,” in *2018 Design, Automat. & Test Eur. Conf. & Exhibition (DATE)*, 2018, pp. 1453–1458.
- [213] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, “Physical one-way functions,” *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

- [214] U. Rührmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, and G. Csaba, “Applications of high-capacity crossbar memories in cryptography,” *IEEE Trans. Nanotechnol.*, vol. 10, no. 3, pp. 489–498, 2010.
- [215] L. Kusters and F. M. Willems, “Secret-key capacity regions for multiple enrollments with an sram-puf,” *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 9, pp. 2276–2287, 2019.
- [216] M. A. Qureshi and A. Munir, “PUF-RAKE: A PUF-based robust and lightweight authentication and key establishment protocol,” *IEEE Trans. Dependable Secure Comput.*, 2021.
- [217] Y. Gao, M. van Dijk, L. Xu, S. Nepal, and D. C. Ranasinghe, “TREVERSE: Trial-and-error lightweight secure reverse authentication with simulatable pufs,” *arXiv preprint arXiv:1807.11046*, 2018.
- [218] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, “Mobile device identification via sensor fingerprinting,” *arXiv preprint arXiv:1408.1416*, 2014.
- [219] Y. Son, J. Noh, J. Choi, and Y. Kim, “Gyrosfinger: Fingerprinting drones for location tracking based on the outputs of MEMS gyroscopes,” *ACM Trans. Privacy and Security*, vol. 21, no. 2, p. 10, 2018.
- [220] Y. Kim and Y. Lee, “Cam PUF: Physically unclonable function based on CMOS image sensor fixed pattern noise,” in *Proc. ACM Ann. Design Autom. Conf (DAC)*, 2018, p. 66.
- [221] Z. Ba, S. Piao, X. Fu, D. Koutsonikolas, A. Mohaisen, and K. Ren, “ABC: Enabling smartphone authentication with built-in camera,” in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2018.
- [222] O. Willers, C. Huth, J. Guajardo, and H. Seidel, “MEMS gyroscopes as physical unclonable functions,” in *Proc. ACM Conf. Comput. and Commun. Security (CCS)*, 2016, pp. 591–602.
- [223] J. Zhang, A. R. Beresford, and I. Sheret, “SensorID: Sensor calibration fingerprinting for smartphones,” in *Proc. IEEE Symp. Security and Privacy (S&P)*, May 2019.
- [224] Z. Ba, Z. Qin, X. Fu, and K. Ren, “CIM: Camera in motion for smartphone authentication,” *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 11, pp. 2987–3002, 2019.
- [225] E. Quiring, M. Kirchner, and K. Rieck, “On the security and applicability of fragile camera fingerprints,” in *Proc. Eur. Symp. Res. in Comput. Secur. (ESORICS)*, 2019, pp. 450–470.

BIBLIOGRAPHY

- [226] Y. Cheng, X. Ji, J. Zhang, W. Xu, and Y.-C. Chen, “DeMiCPU: Device fingerprinting with magnetic signals radiated by CPU,” in *Proc. ACM Conf. Comput. and Commun. Security (CCS)*, 2019, pp. 1149–1170.
- [227] Z. Li, A. S. Rathore, C. Song, S. Wei, Y. Wang, and W. Xu, “PrinTracker: Fingerprinting 3D printers using commodity scanners,” in *Proc. ACM Conf. Comput. and Commun. Security (CCS)*, 2018, pp. 1306–1323.
- [228] S. N. Dhanuskodi, X. Li, and D. Holcomb, “COUNTERFOIL: Verifying provenance of integrated circuits using intrinsic package fingerprints and inexpensive cameras,” in *USENIX Security Symp.*, 2020, pp. 1255–1272.
- [229] M. T. Rahman, F. Rahman, D. Forte, and M. Tehranipoor, “An aging-resistant ro-puf for reliable key generation,” *IEEE Trans. Emerging Topics in Computing*, vol. 4, no. 3, pp. 335–348, 2015.
- [230] T. Rahman, D. Forte, J. Fahrny, and M. Tehranipoor, “ARO-PUF: An aging-resistant ring oscillator PUF design,” in *Proc. conf. on Design, Autom. & Test in Eur.*, 2014, p. 69.
- [231] R. Hesselbarth, F. Wilde, C. Gu, and N. Hanley, “Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs,” in *IEEE Int. Symp. Hardware Oriented Secur. and Trust (HOST)*, 2018, pp. 126–133.
- [232] B. Gassend, D. Clarke, M. Van Dijk, and S. Devadas, “Silicon physical random functions,” in *Proc. ACM Conf. Comput. Commu. Secur. (CCS)*, 2002, pp. 148–160.
- [233] Q. Wan and J. Liu, “Smart-home architecture based on bluetooth mesh technology,” in *IOP Conf. Series: Materials Science and Eng.*, IOP Publishing, vol. 322, 2018, p. 072 004.
- [234] Y. Gao, M. Van Dijk, L. Xu, W. Yang, S. Nepal, and D. Ranasinghe, “Treverse: Trial-and-error lightweight secure reverse authentication with simulatable pufs,” *IEEE Trans. Dependable and Secure Computing*, 2020.
- [235] D. Mrityunjaya, N. Kumar, S. Ali, H. Kelagadi, *et al.*, “Smart transportation,” in *Proc. IEEE Intern. Conf. IoT Social, Mobile, Analytics and Cloud (I-SMAC)*, 2017, pp. 1–5.
- [236] D. Nedospasov, J.-P. Seifert, A. Schlösser, and S. Orlic, “Functional integrated circuit analysis,” in *Proc. IEEE Intern. Symp. Hardware-Oriented Secur. Trust (HOST)*, 2012, pp. 102–107.

- [237] J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn, and C. Winnerlein, “Blake2: Simpler, smaller, fast as md5,” in *Intern. Conf. on Applied Cryptography and Network Security*, Springer, 2013, pp. 119–135.
- [238] Y. Su, Y. Gao, O. Kavehei, and D. C. Ranasinghe, “Hash functions and benchmarks for resource constrained passive devices: A preliminary study,” in *IEEE Intern. Conf. on Pervasive Comput. and Commun. Workshops*, 2019, pp. 1020–1025.
- [239] W. M. Bolstad and J. M. Curran, *Introduction to Bayesian statistics*. John Wiley & Sons, 2016.
- [240] Y. Dong, A. Wickramasinghe, H. Xue, S. F. Al-Sarawi, and D. C. Ranasinghe, “A novel hybrid powered RFID sensor tag,” in *IEEE Intern. Conf. on RFID (IEEE RFID)*, 2015, pp. 55–62.
- [241] A. Buffi, A. Michel, P. Nepa, and B. Tellini, “Rssi measurements for RFID tag classification in smart storage systems,” *IEEE Trans. on Instru. and Meas.*, vol. 67, no. 4, pp. 894–904, 2018.
- [242] P. Zhang, J. Gummeson, and D. Ganesan, “Blink: A high throughput link layer for backscatter communication,” in *Intern. Conf. on Mobile Systems, Spp., and Serv.(MobiSys)*, 2012, pp. 99–112.
- [243] A. Bothe and N. Aschenbruck, “Improving UHF RFID read rates by juggling selected bits,” in *Intern. Conf. on RFID (IEEE RFID)*, 2016, pp. 1–8.
- [244] P. Schober, C. Boer, and L. A. Schwarte, “Correlation coefficients: Appropriate use and interpretation,” *Anesthesia & Analgesia*, vol. 126, no. 5, pp. 1763–1768, 2018.
- [245] M. A. Khelif, J. Lorandel, and O. Romain, “Non-invasive I2C hardware trojan attack vector,” in *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI and Nanotec. Syst. (DFT)*, 2021, pp. 1–6.
- [246] H. Badran, “IoT security and consumer trust,” in *Proc. Annu. Int. Conf. on Dig. Gov. Res. (DG.O)*, 2019, pp. 133–140.
- [247] A. Jha and M. Sunil, “Security considerations for internet of things,” *L&T Technology Services*, 2014.
- [248] B. Hammi, S. Zeadally, H. Labiod, R. Khatoun, Y. Begriche, and L. Khoukhi, “A secure multipath reactive protocol for routing in IoT and HANETs,” *Ad Hoc Networks*, vol. 103, p. 102 118, 2020.

BIBLIOGRAPHY

- [249] M. Capellupo, J. Liranzo, M. Z. A. Bhuiyan, T. Hayajneh, and G. Wang, “Security and attack vector analysis of IoT devices,” in *Proc. Springer Int. Conf. on Secu., Privacy and Anonymity in Comp., Comm. Storage (SpaCCS)*, 2017, pp. 593–606.
- [250] S. Skorobogatov, “Physical attacks and tamper resistance,” in *Introduction to Hardware Secu. and Trust*, Springer, 2012, pp. 143–173.
- [251] B. D. Davis, J. C. Mason, and M. Anwar, “Vulnerability studies and security postures of IoT devices: A smart home case study,” *Internet of Things J.*, vol. 7, no. 10, pp. 10 102–10 110, 2020.
- [252] Y. Sasaki and K. Aoki, “Finding preimages in full MD5 faster than exhaustive search,” in *Proc. Springer Annu. Int. Conf. on the Theory and App. of Crypto. Tech. (Eurocrypt)*, 2009, pp. 134–152.
- [253] L. Morel and D. Couroussé, “Idols with feet of clay: On the security of bootloaders and firmware updaters for the IoT,” in *Proc. IEEE Int. New Circuits and Syst. Conf. (NEWCAS)*, 2019, pp. 1–4.
- [254] H. Yuan, J. Zhao, B. Zhao, X. Xie, Z. Li, Q. Zhang, G. Wang, F. Ma, Y. Zhi, and S. Chang, “A fast and simple method for obtaining microcircuit card information,” in *Proc. Adv. Info. Manage., Comm., Elec. and Auto. Control Conf. (IMCEC)*, 2019, pp. 1075–1079.
- [255] A. Ibrahim, A.-R. Sadeghi, and G. Tsudik, “Us-aid: Unattended scalable attestation of IoT devices,” in *Proc. IEEE Symp. on Reliable Distributed Syst. (SRDS)*, 2018, pp. 21–30.
- [256] A. A. Pammu, K.-S. Chong, W.-G. Ho, and B.-H. Gwee, “Interceptive side channel attack on AES-128 wireless communications for iot applications,” in *Proc. IEEE Asia Pacific Conf. on Circuits and Syst. (APCCAS)*, 2016, pp. 650–653.
- [257] S. Vasile, D. Oswald, and T. Chothia, “Breaking all the things—a systematic survey of firmware extraction techniques for IoT devices,” in *Proc. Springer Int. Conf. on Smart Card Res. and Adv. App. (CARDIS)*, 2018, pp. 171–185.
- [258] J. Bartlett, *Electronics for Beginners*. Apress, 2020.
- [259] A. Gupta, *The IoT Hacker’s Handbook: A Practical Guide to Hacking the Internet of Things*. Apress, 2019.
- [260] A. Cui, M. Costello, and S. Stolfo, “When firmware modifications attack: A case study of embedded exploitation,” in *Proc. The Network and Distributed System Security Symp. (NDSS)*, 2013.

- [261] H. Li, M. Bhargav, P. N. Whatmough, and H.-S. P. Wong, “On-chip memory technology design space explorations for mobile deep neural network accelerators,” in *Proc. ACM/IEEE Des. Autom. Conf. (DAC)*, 2019, pp. 1–6.
- [262] T. Instruments, *MSP430G2x33 Datasheet*, Apr. 2016. [Online]. Available: <https://www.ti.com/lit/ds/symlink/msp430g2433.pdf>.
- [263] X. Wuhan. (Nov. 2020). “Xm25qh64c 3v 64m-bit serial flash memory with dual/quad spi & qpi,” [Online]. Available: <https://www.xmcwh.com/uploads/207/XM25QH64C.pdf>.
- [264] Y. Gao, Y. Su, W. Yang, S. Chen, S. Nepal, and D. C. Ranasinghe, “Building secure SRAM PUF key generators on resource constrained devices,” in *2019 IEEE Intern. Conf. on Pervasive Comput. and Commun. Workshops*, 2019, pp. 912–917.
- [265] D. E. Holcomb, W. P. Burleson, and K. Fu, “Power-up SRAM state as an identifying fingerprint and source of true random numbers,” *IEEE Trans. on Comput.*, vol. 58, no. 9, pp. 1198–1210, 2008.
- [266] J. Tan, *Robust downstream communication and storage for computational RFIDs*. Department of Software Technology, Delft University of Technology., 2015.

Index

- Adversary Model, 45
- AES, 59, 102, 226
- AES accelerator, 59
- ASK, 23

- Backscatter, 23
- BCH, 54, 67, 132
- BER, 33, 37, 54, 65, 214
- bias, 34, 54, 56, 65, 142, 170, 191, 216, 228
- Bootloader, 46
- bootloader, 63, 91, 92, 102, 105
- broadcast, 89, 235
- brownout, 51, 60, 93, 242, 248, 283

- case study, 70, 111
- checkpointing, 61
- conditional firmware update, 47, 54, 55
- CRFID, 71, 82, 111, 123, 130, 134, 226, 232, 241, 281
- CVN, 57, 143

- dataset, 16, 38, 64, 143, 163, 201, 214, 234, 305
- denial of service, 86
- DoS, 46, 86

- EPC, 24
- EPC Gen2, 22, 47, 64, 85, 86, 94, 278
- experiments, 38, 64, 109, 130, 164, 213
- exposed memory bus, 263
- extraction efficiency, 66

- firmware, 8, 11, 18, 43, 46, 86, 88, 91, 104, 110, 252, 258, 270
- firmware update, 42, 43, 45, 48, 62, 82, 86, 224, 235, 289

- helper data manipulation, 73, 139, 140, 185, 237
- HWAES-CMAC, 59, 102, 226

- IEM, 60, 69, 243
- implementation overheads, 67, 107, 108

- intermittent execution, 60
- IPC, 60, 93, 246

- JTAG, 63, 74, 97, 252, 261, 265, 302

- key derivation, 15, 47, 53, 62, 65, 108, 127, 128, 139, 170, 182, 184
- key failure rate, 55, 129, 134

- LAN, 23
- LDO, 26
- LLRP, 23, 24, 64, 72, 85, 109
- low power mode, 89, 95, 236
- LPM, 243

- MAC, 45, 58, 63, 69, 88, 102
- majority voting, 37, 55, 132, 136
- Man-in-the-middle attack, 72
- mask, 57, 171, 196, 202, 223, 302
- MCU, 26, 226
- memory arrangement, 62, 92, 106
- min entropy, 56, 73, 142

- NIST random number test, 53, 102

- operating condition, 33, 37, 122, 123
- operational condition, 55

- pilot-observer mode, 84, 98
- power aware execution model, 93, 95
- pre-selection, 55, 67, 136
- PSK, 23

- remote attestation, 90, 104, 108
- reverse fuzzy extractor, 67, 226
- RFE, 67, 226
- RFID, 22
- RNG, 45, 49, 53, 88, 124, 137, 140, 154, 197, 223, 224

- secure storage, 7, 46, 88, 91, 106
- security analysis, 72, 112, 184, 227

sense amplifier, 30

symmetric key primitive, 102

threat model, 45, 85, 139, 184, 227, 259

TRNG, 51

unsafe language, 86

video demo, xxvii, xxviii, 12–16, 39, 43, 63, 84, 109,

112, 153, 163, 178, 180, 258, 260, 266,

269–271, 278, 287, 289, 302

Biography

Yang Su received his B.Eng degree in Electrical and Electronics Engineering with the First Class Honours from The University of Adelaide, Australia in 2015. In his third year in the university, Yang Su joined Auto-ID Lab Adelaide as a Research Assistant. Few month later Yang Su published his first research paper based on his summer research study on sensor data revival from battery-free RF powered sensor devices.



After his Bachelor degree, Yang Su worked at the Auto-ID Lab Adelaide as a Research Associate (RA), specialising in hardware security and applied IoT engineering. During his service as RA, Yang Su discovered a channel-to-channel crosstalk phenomenon between adjacent USB ports, such phenomenon is common on numerous commodity devices and could lead to information leakage. This research was published under the title "USB Snooping Made Easy: Crosstalk Leakage Attacks on USB Hubs" in 2017 at the USENIX Security Symposium.

In 2017, he was awarded Australian Postgraduate Awards (APA) Scholarship to pursue his PhD under A/Prof Damith Chinthana Ranasinghe in the School of Computer Science, The University of Adelaide, and Dr Omid Kavehei, School of Biomedical Engineering, The University of Sydney. In 2018 Yang Su received AMSI travel funding to attend Authentication for the Future Internet of Things. Since 2019, Yang Su has been actively involved in teaching at The University of Adelaide as a tutor, master research project supervisor and, most recently, as lecturer.

Mr. Su has served as a reviewer for a number of recognised journals and conferences including IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE INTERNATIONAL CONFERENCE ON PERVASIVE COMPUTING AND COMMUNICATIONS WORKSHOPS, and WORLD CONFERENCE ON INFORMATION SECURITY APPLICATIONS. He is a graduate student member of the IEEE.

Yang Su
yang.su01@adelaide.edu.au
a1601999@gmail.com