Physical Unclonability Framework for the Internet of Things

Konstantinos Goutsos

A thesis submitted for the degree of
Doctor of Philosophy

Newcastle University
School of Engineering
Faculty of Science, Agriculture and Engineering

October 2020

To my late grandparents, Κώστας and Μαρία,
who I wish could enjoy this moment with me.

To my parents, Μπάμπης and Μαρία,
for without them I would have not started this journey.

And to Johanna,
for without her support and affection
I would not have arrived at the destination.

Acknowledgements

Before all others, I am deeply grateful to my supervisor and mentor Dr. Alex Bystrov who introduced me to PUFs while I was still a Master's student. He has been my best ally over the years, and made sure that everything I needed to support my work was always available. I would also like to thank my co-supervisor Prof. Alex Yakovlev for his input and support throughout this project.

It is hard to overstate my appreciation to the technical and administrative staff of the School. They often went above and beyond their duties to assist me with every issue I brought to their attention. I am also profoundly thankful to the School of Engineering (formerly School of Electrical and Electronic Engineering) for funding my PhD studies.

The journey of a PhD is far from exclusively a research one. Over the years I was delighted to take part in a great variety of activities from seminars and conferences to supporting younger students in lectures and laboratories. All these endeavours allowed, to meet a breadth of wonderful people some of which became my (volunteer) reviewers, colleagues, friends and even housemates. Each and every one of them was a piece of this doctoral puzzle in a way that I will forever remember. I would also especially like to thank my colleagues from the School of Engineering who shared their knowledge and skills with me at every opportunity.

Finally, I would like to extend my deepest gratitude and apologies to my family and friends, all of whom I greatly frustrated with my absence and irritable presence at various points during this work. I am glad that all of you were there, pushing me along and believing in me, even when I did not believe in myself.

Abstract

The rise of the Internet of Things (IoT) creates a tendency to construct unified architectures with a great number of edge nodes and inherent security risks due to centralisation. At the same time, security and privacy defenders advocate for decentralised solutions which divide the control and the responsibility among the entirety of the network nodes. However, spreading secrets among several parties also expands the attack surface.

This conflict is in part due to the difficulty in differentiating between instances of the same hardware, which leads to treating physically distinct devices as identical. Harnessing the *uniqueness* of each connected device and injecting it into security protocols can provide solutions to several common issues of the IoT. Secrets can be generated directly from this uniqueness without the need to manually embed them into devices, reducing both the risk of exposure and the cost of managing great numbers of devices.

Uniqueness can then lead to the primitive of *unclonability*. Unclonability refers to ensuring the difficulty of producing an exact duplicate of an entity via observing and measuring the entity's features and behaviour. Unclonability has been realised on a physical level via the use of Physical Unclonable Functions (PUFs). PUFs are constructions that extract the inherent unclonable features of objects and compound them into a usable form, often that of binary data. PUFs are also exceptionally useful in IoT applications since they are low-cost, easy to integrate into existing designs, and have the potential to replace expensive cryptographic operations. Thus, a great number of solutions have been developed to integrate PUFs in various security scenarios. However, methods to expand unclonability into a complete security framework have not been thoroughly studied.

In this work, the foundations are set for the development of such a framework through the formulation of an *unclonability stack*, in the paradigm of the OSI reference model. The stack comprises layers propagating the primitive from the unclonable PUF ICs, to devices, network links and eventually unclonable systems. Those layers are introduced, and work towards the design of protocols and methods for several of the layers is presented.

A collection of protocols based on one or more unclonable tokens or *authority devices* is proposed, to enable the secure introduction of network nodes into groups or *neighbourhoods*. The role of the authority devices is that of a consolidated, observable root of ownership, whose physical state can be verified. After their introduction, nodes are able to identify and interact with their peers, exchange keys and form relationships, without the need of continued interaction with the authority device.

Building on this introduction scheme, methods for establishing and maintaining *unclonable links* between pairs of nodes are introduced. These pairwise links are essential for

the construction of relationships among multiple network nodes, in a variety of topologies. Those topologies and the resulting relationships are formulated and discussed.

While the framework does not depend on specific PUF hardware, SRAM PUFs are chosen as a case study since they are commonly used and based on components that are already present in the majority of IoT devices. In the context of SRAM PUFs and with a view to the proposed framework, practical issues affecting the adoption of PUFs in security protocols are discussed. Methods of improving the capabilities of SRAM PUFs are also proposed, based on experimental data.

Publications

1. Konstantinos Goutsos, *PUF-Based Authority Device Scheme*, NCL-EEE-MICRO-TR-2019-212, uSystems Research Group, EEE, School of Engineering, Newcastle University, May 2019.

   A preliminary version of Chapter 5 was previously made available as a technical report. The content of Chapter 5 constitutes an updated version of that report, including essential improvements to the proposed protocols and their analysis.

2. K. Goutsos and A. Bystrov, "Lightweight PUF-based Continuous Authentication Protocol," in 2019 International Conference on Computing, Electronics Communications Engineering (iCCECE), Aug. 2019, pp. 229–234. [1]

   The studies underpinning this paper can be found in Chapter 6. In the paper an elementary view of the proposed authentication protocol was given, along with a brief analysis. Chapter 6 provides a more in-depth analysis and motivation for the protocol design, and introduces an additional variant based on Zero Knowledge proofs.

Contents

List of Figures

List of Tables

List of Protocols

List of Algorithms

List of Symbols and Abbreviations

$\mathbf{AC_x}$ Monotonic authentication counter for x.

ACK Acknowledgement.

$\mathbf{AD_x}$ Authority Device x.

ADS Authority Device Scheme.

API Application Programming Interface.

ASIC Application-Specific Integrated Circuit.

BER Bit Error Rate.

COTS Commercial Off-the-Shelf.

CRP Challenge-Response Pair.

$\mathbf{DEC_k(x)}$ Public key decryption of x with public key k.

DRBG Deterministic Random Bit Generator.

ECC Elliptic Curve Cryptography.

$\mathbf{ENC_k(x)}$ Public key encryption of x with secret key k.

$\mathbf{FC_x}$ Monotonic failure counter for x.

FPGA Field-Programmable Gate Array.

GPP General Purpose Processor.

$\mathbf{HASH(x)}$ Cryptographic hash of x.

HMAC Hash-based Message Authentication Code.

$\mathbf{HMG_k(x)}$ Calculation of the HMAC of x with secret key k.

$\mathbf{HMV_k(x, y)}$ Verification of the HMAC y of x with secret key k.

HSM Hardware Security Module.

IoT Internet of Things.

$\mathbf{N_x}$ Node x.

NBTI Negative Bias Temperature Instability.

OSI Open Systems Interconnection.

$\mathbf{P_x}$ Public key of x.

$\mathbf{PKG(x)}$ Derivation of an asymmetric key pair from seed x.

PPT Probabilistic Polynomial Time.

PRF Pseudorandom Function Family.

PUF Physical Unclonable Function.

$\mathbf{PUF_x(y)}$ Evaluation of the PUF of device x with challenge y.

$\mathbf{RNG_x}()$ Evaluation of the random number generator of device x.

RSA Rivest Shamir Adleman.

$\mathbf{S_x}$ Private (secret) key of x.

$\mathbf{SIG_k(x)}$ Signature of x with private key k.

SRAM Static Random Access Memory.

TPM Trusted Platform Module.

TRNG True Random Number Generator.

$\mathbf{VER_k(x, y)}$ Verification of signature y of x with public key k.

$\|$ Concatenation operator.

$\oplus$ Bitwise XOR operator.

ZK Zero Knowledge.

$\mathbf{ZKC(x)}$ Calculation of ZK commitment for value x.

ZKP Zero Knowledge Proof.

$\mathbf{ZKP_k(x, y)}$ Generation of ZK proof with key k, value x and challenge y.

$\mathbf{ZKV_k(x, y, z)}$ Calculation of ZK commitment for value x.

Part I

Background

1. Introduction

1.1 Motivation

1.1.1 Ownership and Trust Relationships for Humans and Machines

Recent advances in electronics and networking have facilitated the creation of ubiquitous connected devices and machines which were previously operating in a stand-alone manner. This has led to the emergence of new system paradigms where thousands or even millions of devices are constantly communicating and cooperating towards various goals, creating the Internet of Things (IoT). The integration of these devices in all aspects of our society is widespread, leading to enormous amount of sensitive data being processed and stored in various forms and on various types of devices. As a result, today more than ever, there is an increased need to uphold the four pillars of information security: *integrity, confidentiality, privacy, and non-repudiation.*

At the same time, security comes down to people, as machines alone do not require security provisions; security weaknesses and threats are created and exploited by humans. Unfortunately, the interface between humans and machines is bound to disadvantage the former, leading to lapses of security. For billions of years, humans have been subconsciously developing intuitive methods of evaluating and establishing the trust relationships that underpin societies. Humans have evolved to be able to unquestionably recognise other members of their species, assuming unique identities and creating relationships based on them.

These primitives do not transfer into the domain of machines. Attempting to establish a verifiable and easy to safeguard alternative, researchers have created methods and protocols based on the principle of a 'key': a piece of secret information which guarantees the security of the system as long it remains secret, even if everything else about the system is known. This so-called 'Kerckhoffs' principle' [2] is the basis of every cryptographic method available today, and greatly simplifies the creation and maintenance of trust relationships, be they human-to-human, human-to-machine or machine-to-machine.

However, as the number of machines grows, their owners are unable to keep up with the security provisions required. Merely embedding the appropriate 'key' in every device demands a significant amount of time and, more importantly, expertise. In many situations, none of those resources is readily available. According to a recent survey of the World Economic Forum, almost 1 in 2 businesses cite lack of knowledge and resources to use IoT solutions at scale as the main hindrance for the adoption of such solutions[3].

On the other hand, according to PwC[4], consumer electronics constitute the largest market sector in IoT with over a quarter of the total revenue. At the same time, these users

have very constrained, if any, technical resources. When security is considered, the complexity of IoT deployment and maintenance increases exponentially, with issues including access control, variable security levels, device clustering, and damage containment.

Thus, despite the continued efforts of security researchers, in practice a vast amount of systems have relatively few security provisions. Even worse, the security provisions that do exist are often undermined by poor practices leading to inadvertent exposure of system secrets. We firmly believe that simple and inexpensive security methods should be the focus of future IoT development. These methods should require minimal configuration regardless of the complexity of the system, greatly lowering the barrier of entry for device manufacturers and end users alike. Some human interaction is unavoidable, and in fact desirable, since human operators are able to examine the system and its environment and make informed decisions. However, there is no reason for these operators to have access to the low-level secrets that are involved in security protocols.

Unique device identities form the foundation of the methods discussed above. In addition, these identities need to be utilised and communicated in a manner that ensures their continued singularity and confidentiality. Thus, the challenge lies in forming appropriate security protocols that exploit identity without exposing it.

1.1.2 Unclonability and Physical Disorder

The notion of unique identity based on innate features and behaviours is encapsulated by the concept of *unclonability*, in human or object contexts. Unclonability is manifested in cascading domains, from an unclonable person (object), to unclonable relationships between persons (objects), or even the unclonable fabric of relationships underpinning modern societies (machine-to-machine networks). The parallel between human societies and machine networks is clear, nevertheless a disparity can be observed between the topology of machine networks and the relationships in the human organisations using them.

This disparity is due to the aforementioned lack of continuity between the human and the digital domain. Thus, technical solutions are required to support unclonability and, through it, ownership. These solutions comprise novel methods and protocols in conjunction with the adaptation of existing ones to provide security provisions that fit the structure of human organisations and relationships. While this approach gives the idea of more human involvement, in reality the goal is to prevent human interaction where it is not fundamentally needed and instead introduce the human factor in different parts of the system.

The first step towards the development of such methods is the creation of sources of unclonability in the digital domain. Just as humans exploit minute, almost imperceptible characteristics and behaviours to identify their peers, a similar approach can be followed for objects. Physical Unclonable Functions (PUFs) are based on such inherent properties of objects, compounding these properties into an encoded form, to be used as an

object identifier. These embedded characteristics are the result of physical phenomena that differ between instances of the same PUF, even if the same manufacturing process is explicitly followed. Driven by this *physical disorder*, PUF outputs are highly unpredictable, exhibiting substantial entropy. This entropy, representing the inability to control PUF behaviour, makes PUFs a valuable building block in security protocols.

Although originally introduced by Pappu et al. [5] as analogue devices, PUFs quickly developed into purely digital components, a fact that allows for their inclusion in new and existing device architectures. Electronic PUFs are constructed from common electronic components such as memory or delay elements. They are able to accept a challenge in the form of a binary vector and return a response in the same form, based on their underlying physical construction. This binary interface is invaluable in the context of existing systems, since PUF outputs can replace traditional cryptographic keys while providing novel properties.

A significant benefit of PUF-based secrets over existing key generation methods is the inability of external parties to fully examine and copy the internal state of the PUF. In other words, the attack surface is greatly reduced on a physical level, requiring a high level of expertise from potential adversaries. However, as soon as the secrets leave the PUF their protection seizes to exist. Thus, measures should be taken to ensure the confidentiality of those secrets.

Additionally, secrets naturally occur in PUFs without explicit external intervention. As a result, provided that the aforementioned confidentiality is established, device users or operators are never in knowledge of the secrets. The elimination of the human element in this context allows for the use of secrets with higher entropy and size, characteristics which are exceptionally hard for humans to deal with.

1.1.3 Hardware Roots of Authority

We envision a solution based on one or more 'authority devices (ADs)' which internally contain the necessary secret information, provided by a PUF. Those secrets are used for the configuration of networked systems while ensuring that they are not shared with any party outside the authority device. Network nodes also incorporate a similar secret generation mechanism, essentially infusing every part of the system with unclonability.

In other words, an authority device is an ownership token which can be temporarily connected to each node, exchanging security and configuration information. This information can be subsequently used by nodes to create relationships and organise into clusters or 'neighbourhoods', providing advanced functionality. The use of authority devices has the following benefits:

- The 'human attack surface' is reduced. Since no one has knowledge of the underlying secrets, there is no chance of extortion, threats, or even physical harm aiming to extract private information.

- Secrets are present only in their respective devices, providing a simple, tamper-

evident source of ownership. Destroying the secret after using it or, in case of compromise, is as simple as destroying the device itself. In many cases of PUFs, the unclonable physical phenomena are quite sensitive to physical stress, making the destruction of PUF secrets particularly simple.

- The physical aspect of the AD allows for the delegation of duties, since they provide verifiability. An AD can be handed over to a third-party which is instructed to perform the system configuration. After the end of the configuration process, the device can be returned to the system owner who can examine its condition and be certain that the underlying secrets have not been cloned.

On a conceptual level, ADs serve as extensions of their holders, even though the latter might not directly perceive the secrets included in the ADs. One example can be seen in relation to the use of biometric features in security protocols. Biometrics can be considered physical unclonable secrets that are directly derived from features of the human body and behaviour. However, due to the inability to control those secrets (and thus revoke them or destroy them as necessary) users might well be hesitant to reveal them to third parties. In our scenario, this unclonable power is transferred to the AD and used in place of the users' biometric data. The biometric features can then serve as a second factor authentication method for the device itself. This 'biometrics by proxy' scenario would enable the same level of security as traditional biometrics whilst ensuring the privacy and the ownership of the identifying information itself, leading to a new paradigm of 'private biometrics'.

Further examples include the military or organisations with a similar structure. Such organisations comprise multiple distinct groups which spread over multiple levels of authority. The compartmentalisation and verifiability provided by the use of authority devices allows for the containment of damage in case of a breach, as well as provide undeniable proof that such a breach did not take place. For instance, a commanding officer would be able to delegate the duty of physically deploying and configuring a network system to his subordinates. Upon completion, the officer can ascertain that the 'master' secrets are returned to him upon return of the corresponding AD.

In summary, the development of an authority device as it was described above, in conjunction with higher level protocols based on its existence: (a) lead to less human involvement meaning simpler, cheaper, and safer security provisions, and (b) provides stronger security guarantees including containment of breaches, stronger secrets, and provable secret destruction.

1.2 Research Problem and Scope

Our work addresses the problem of creating and evaluating a framework of methods and protocols in the IoT context that will enable the inclusion of unclonability in every level of the networking stack. While a great amount of work has been performed on PUFs and their use as security primitives, there exists a need to extend this work to encompass

the entirety of modern networked systems. The goals of this work can be split into two branches: firstly, formulating an extended view of unclonability in novel contexts, and secondly, providing practical methods of realising unclonability in those contexts.

For the most part of this work, we choose to abstract away from details of specific PUF classes (with their associated advantages and drawbacks) and instead aim to utilise them as a building block with a specified behaviour. In addition, we choose to examine and discuss several aspects of the framework in order to demonstrate those aspects in an applied context, instead of aiming at information-theoretical solutions that would have little meaning in such a context. Furthermore, our work targets a particular class of devices and scenarios which can be broadly categorised as IoT or M2M networks. These systems typically comprise a large number of devices with relatively limited resources. The devices are also often vulnerable to physical attacks since they are deployed in remote or unmonitored environments. Especially in the IoT context, networking capabilities are increasingly added to consumer and devices, where the majority of end users lack the expertise to keep them secure.

To that end, we aim to construct a security framework based on unclonability, with the following properties:

- Simplicity: can be configured and maintained with minimal need for user interaction or expertise.
- Minimal disruption: does not significantly alter user experience and established processes, ensuring that users will not eventually disable the security provisions.
- Minimal overhead: designed to exploit the increased security provided by PUFs, despite their relatively small cost of implementation.
- Secure by default: security is ingrained in every level, setting a minimum baseline. No option exists for disabling this basic protection.
- Hardware foundations: the framework draws its security guarantees from the hardware domain through the use of PUFs.
- Multiple security levels: alternatives are offered for costly, advanced security functions, which can be employed depending on the application.
- Decentralised centralisation: the centralised control elements (authority devices) remain in the possession and the responsibility of the system owner. This leads to more control in the hands of the users/operators, no need for trusted third-parties, and ultimately increased privacy, since security information is not shared upstream to third-parties.
- Scalability: the framework is structured around edge nodes which operate independently of central authorities and in cooperation with their neighbours.

The proposed framework can be envisioned as an analogy to the Open Systems Interconnection (OSI) reference model, creating an *unclonability stack* as seen in Fig. 1.1. The stack expands the notion of unclonability from the hardware level (PUFs) to the system level of multiple unclonable neighbourhoods of nodes, via creating pairwise unclonable links and combining those links into higher order relationships.

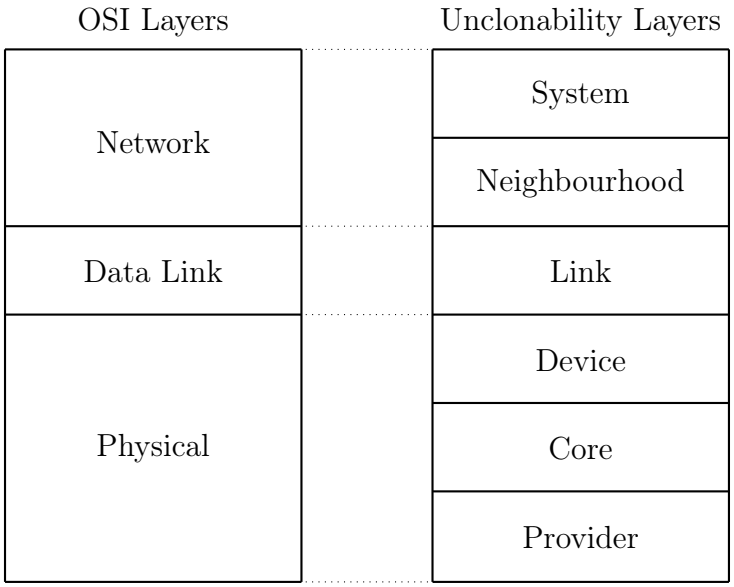


Figure 1.1: OSI layers and Unclonability Stack

1.3 Main Contributions

The main contributions of this thesis are summarised as follows:

- An unclonability stack is formulated and its layers are outlined. This allows us to explore the capabilities and potential of individual layers while retaining a useful isolation from the rest of the stack. To the best of our knowledge, our work is the first to examine unclonability in such a context, aiming at expanding its benefits throughout multiple network layers.

- A collection of protocols supporting the secure creation of network node clusters by providing features such as authentication, key exchange, enrolment, and decommission is presented. This collection, which we name 'Authority Device Scheme' enables the use of the authority tokens discussed in the previous sections and provides the basis for further development of the stack. The majority of the scheme is based on asymmetric cryptography seeded by PUFs, and is thus designed to be used infrequently over the lifetime of a system, to avoid excessive overhead.

- To tackle the issue of authentication in resource constrained nodes, two variations of a mutual authentication protocol are proposed. Both are based on the primitive of secret refreshment or 'ratcheting', and support continuous pairwise authentication

based on refreshed PUF secrets. To the best of our knowledge, our proposal is the first that combines PUFs and so-called cryptographic 'ratchets'[6]. The first variant is designed to have very limited overhead, using only hash functions as a cryptographic primitive. On the other hand, the second variant provides higher security guarantees through the use of zero knowledge proofs, albeit with higher overhead.

- The architecture of a cryptographic core including a PUF is outlined, combining common electronic components and designs into a self-contained block. The core is designed to protect the PUF secrets while exposing the necessary functionality to higher layers of the unclonability stack. Due to the scope of this work, the architecture is presented as a blueprint for future development of similar components, with the main aim of ensuring the least exposure of internal secrets.

- A particular PUF class, SRAM PUFs, is analysed through experimental data with a number of outcomes. Firstly, we we demonstrate the validity of our assumptions and abstractions regarding the behaviour of real world PUFs, over a multiplicity of conditions. Additionally, we gain valuable insight into the intricacies of SRAM cells in the context of their use as PUFs, enabling us to derive metrics and proposed directions for further investigation.

- Finally, a proof-of-concept implementation of the proposed protocols is presented, with an aim towards the verification of the functional integrity of the solutions.

1.4 Structure

This thesis is organised in three parts. In Part I, we describe the foundations of our work. Chapter 2 presents the unclonability approach for enabling and improving the security of networked systems, with a focus on resource constrained devices commonly used in the Internet of Things (IoT). We briefly review the building blocks of security in the digital domain (Chapter 3), and discuss the current state of PUF research in Chapter 4, defining two PUF abstraction models for use in security protocols.

A number of methods supporting our unclonability vision are proposed in Part II. In Chapter 5, we describe a suite of PUF-based asymmetric encryption protocols providing primitives necessary for securely creating groups of network nodes: authentication, key exchange, and group membership management. The interactions for each primitive are specified and their security is discussed.

These primitives are used as the foundation for constructing continuous, mutual authentication protocols, as seen in Chapter 6. After introducing the ideal protocol, two variants of the protocol are presented, with different resource requirements. Both variants make inherent use of PUFs and are designed to periodically refresh their underlying secrets, effectively refreshing the trust between peers involved. Along with a security

discussion of both variants, we outline their comparative advantages and disadvantages.

In Part III, we turn our attention to the practical concerns that arise when using the aforementioned methods in practice. Chapter 7 outlines a blueprint for the construction of a secure cryptographic core with a PUF as its centrepiece. The cryptocore is able to perform cryptographic operations, but more importantly it provides a secure and verifiable interface to the PUF block.

Combining the functionality of all the above in a software model, Chapter 9 discusses the implementation details for the system components mentioned throughout this thesis, providing a proof-of-concept. In this context, we also present a study of a common PUF class, SRAM PUFs, under various environmental conditions, demonstrating its suitability for unclonability applications such as the ones proposed in our work. Finally, we conclude with Chapter 10.

2. The Unclonability Approach

2.1 Introduction

During the first years of Internet development, networking was designed for terminals controlled by an operator. The Internet itself was incrementally built on the idea of connections internal to an organisation, and thus security provisions were added much later, unable to affect the foundations of the system. In addition, over the years, mainly due to commercial reasons, the Internet has become increasingly centralised, used by billions but controlled by comparatively very few parties. Thus, the proliferation of connected objects asks for the development of novel techniques to elevate its security, safeguarding the interests of organisations and individuals, while upholding their rights to ownership, privacy, and safety.

Two major issues were not taken into account in the design of the Internet: human interaction and locality. The number of connected devices is quickly increasing to billions of nodes, in part due to the growth of the 'Internet of Things'. Consequently, the paradigm of 'one operator per machine' is no longer accurate and security methods are required to operate reliably for long periods of time without human intervention.

For the same reason, networked systems can no longer be analysed and modelled as a concise ecosystem. Instead, the locality of these systems needs to be exploited in a manner similar to the concept of neighbourhood in human societies. Building on the existing patterns of human societies can lead to a great improvement in the securing networked systems, both in respect to usability and technical features. Thus, recognising common security methods that have been intuitively used between humans for centuries can enable new perspectives in the information security domain.

In this chapter, we discuss such relationships and patterns, and synthesise a framework of security provisions around them. The foundation of our theory is the concept of *unclonability*, which is a basic primitive of modern societies but has only just recently been made possible in the digital world.

2.2 Relationships

A *relationship* is defined as an association or connection between two or more entities. Depending on the type of relationship, these entities can be humans, animals, or inanimate objects. Some relationships can be wholly passive, especially when participating entities are inanimate, or they can involve a varying degree of activity by the different entities. In most cases, the participation in any kind of relationship attaches new properties to the

entities.

In our work we look at relationships between humans, between machines, and between humans and machines. Perhaps counter-intuitively, individuality is highly important in these relationships. The very nature of such associations, as well as the benefits stemming from them, depend on the ability of the involved parties to establish their identity. For humans this is achieved through the use of unclonable features to recognise, trust and interact, even if this identification is largely subconscious. These features include physical appearance, gait, voice characteristics, behavioural traits, and other details which are generally referred to as biometric features.

A different way of uniquely authenticating a person is through the use of *authority devices* which encapsulate the identity of their owner. Entrusting a device with such capabilities however, creates new technical challenges. Firstly, external parties, whether they are the owner or other entities, should remain oblivious of the internal secrets of the device. This prevents both accidental and malicious exposure of the secrets through human intervention, since it is impossible for the owner to be forced or manipulated into revealing something they do not know. In order for this feature to be achieved, the secret generation mechanism is required to be both automated and reasonably secure against physical attacks.

Secondly, methods of proving the possession of the authority device secrets need to be carefully designed to achieve their purpose without revealing the secrets. Various cryptographic solutions exist for this challenge, including the use of ephemeral information derived from a 'master' secret, 'zero knowledge' proof protocols and others. Most of these solutions are based on identifying information which is statically stored in a form of digital memory and is used over the lifetime of the device. However, manually embedding this information into devices, not only works against the first challenge discussed above, but it also creates additional provisioning costs for device manufacturers. Thus, secret generation methods are required to be scalable whilst protecting the secrecy of their results.

Thirdly, static secret generation and storage can imply the existence of shared secrets among every device instance. Even if the secrets are randomised, they need to be permanently stored in order to be ready for use. As a result, adversaries can duplicate them and create counterfeit authority devices that are indistinguishable from the original. In short, the generated secrets need to be 'naturally occurring' and *unclonable*. This property would ensure that *only one* authority device can exist for a given purpose, with its history and physical location being unquestionably attestable.

While considering these challenges, we should not lose sight of the relationship aspect: potential solutions would operate on multiple levels, starting from a single ownership relationship and extending to multiple groups of entities. Going back to human societies, individuals are organised into all kinds of localities which are important enablers for societal norms that we take for granted. These groups, which we will call *neighbourhoods*

allow, among others, relaying of important information, monitoring of the behaviour of their members, and providing assistance to members in need.

Many of the principles of human relationships have been transferred to the domain of machines, either deliberately or due to the natural tendency of system designers to think in familiar patterns. In the following sections, we discuss some of these principles and how the boundaries between human and machine relationships can be softened.

2.2.1 Ownership

Ownership is ingrained into modern societies to such a degree that we rarely consider how it is affecting every aspect of our lives. However, relationships based on ownership cannot only be formed between humans and their possessions, but also between humans and abstract concepts like information and, as an extension, between machines and information. Abstracting away from the physical domain, ownership can quickly become a complex matter. Besides physically owning a device, its user also owns any data produced and processed by it, along with any resulting actions and consequences.

Data ownership can be defined as the possession but also the responsibility for sets of information[7]. The responsibility for data entails its access, use, and exploitation, the ability to assign such privileges to others, as well as the burden of ensuring that adequate safeguards are in place to manage any risks to the data. There are two sides to this coin; a legitimate user with 'nothing to hide' should nevertheless be able to own their information and any physical objects associated with it. On the other hand, a user that wishes (or needs) to conceal their identity or other aspects of their activity should also be able to disentangle themselves from a device and its digital by-products.

An important issue in data ownership is the establishment of the real owner, due to the nature of digital information. It is difficult to pinpoint and prove the owner of a piece of data as it can be anyone among the creator, the person it possibly refers to or any other middle man, including the parties managing it. This is because ownership was initially defined for physical objects and not information. As such, it cannot be adequately applied to people and organisations that have relationships with data[7]. At the same time, if multiple copies of the same device or data can exist at any given time, the value of ownership is diminished. Therefore, we can discern a gap between the modern digital world and the methods of the past for managing ownership and uniqueness.

We develop technical solutions for the establishment of ownership in the digital domain. Our methods, based on unclonable authority devices, provide a way to unquestionably identify data owners (non-repudiation), while making it possible to transfer or share ownership. Additionally, the aforementioned properties are provided in a manner which is easy to use, while facilitating the containment of damage in case parts of the system are compromised.

2.2.2 Trust and Reputation

Implicit or explicit, *trust* is the cornerstone of relationships and a basic requirement for secure connections. Trust in a party expresses the belief held by other interacting parties that the former will behave as expected. It can stem from the direct observation of an entity's behaviour, or be established through an attestation chain.

In a digital system, trust can be seen on different levels:

- Data are generated and communicated through different entities, possibly belonging to different parties. Thus, data have to be verified for accuracy, origin, and possibly malicious intent.

- Hardware and software components work together to achieve the goals of the system, while also verifying the integrity of their counterpart. However, establishing trust in any of those components implicitly means trusting their developers, manufacturers, and distributors.

- The human element is present on every level of the system: groups and individuals are involved in system specification, design, and manufacturing, as well as ownership and management after deployment. Regardless of the complexity of the process, there are multiple points where an adversary can take advantage of a human operator.

Thus, establishing a trust relationship between two entities has been an important and difficult challenge. Various solutions proposed over time have achieved a partial transfer of societal trust concepts to the digital world, but are often based on assumptions that are difficult to satisfy outside constrained conditions. In the general case, trusting a system means trusting its operations, and trusting the operations means trusting the components that perform them. This concept of building up the trust through the system levels has been described as 'transitive trust'[8].

Transitive trust is an important enabler for security and has been extensively used in various solutions. Commercially available solutions, like Trusted Platform Modules (TPMs), make use of this property to extend the trust boundary from the hardware manufacturer to the software executed on the end system, without requiring the software to be known at manufacturing. A tree of trust relationships is created as a result, with a *root of trust* at its base. While the tree branches are usually created through cryptographic means, the root of trust requires a different relationship, which is often based on the reputation of the hardware manufacturer; the end user accepts that it is more profitable for the manufacturer to uphold their reputation rather than to include malicious logic in their products.

Unfortunately, this assumption is not always true. The increasing value of the personal information exchanged over digital devices provides incentives for manufacturers to intercept device communication and sell the collected data. Using methods with an

unclonable root of trust minimises the trust relationships between the user and external parties. At the same time, it strengthens the user's trust in devices they own, and allows the devices themselves to inherit the trust that their owner enjoys in other relationships.

As mentioned above, in certain cases, trust can only be established through concept of reputation. Similarly to trust, reputation is built on the past behaviour of an entity, either by direct observation or by deduction. Humans make subconscious judgements of other entities based on their reputation and are more likely to trust an entity that they deem reputable.

The concept of reputation in networked systems has been the object of considerable research, with methods of deriving the reputation of nodes involving various primitives like neural networks, probabilistic models, swarm intelligence and others[9]. In summary, for the purposes of our work, the trust and reputation of an entity is influenced by a number of factors that are often weighted differently depending on the application and include:

Behaviour: Behaviour is the main criterion for determining reputation, and is usually quantified as the frequency with which an entity performs unexpected or restricted actions. In addition, irregularities might be signified by the execution of an entity's prescribed duties with an unusual frequency. For instance, a network node that transmits packets more often than expected or seems to be consuming excessive amounts of energy can be deemed suspicious.

Recommendations and accusations: Taking advantage of redundancy in relationships between members of the same group, their reputation can be inferred by collecting positive or negative reviews from their peers. Exploiting redundancy allows for higher quality decisions and improves scalability.

Incentives: The incentives involved in an entity successfully executing its duties, whether monetary or not, tend to affect its reputation, since it has more to lose by breaking trust relationships.

Security features: It is considerably easier to trust a remote entity, when there is knowledge of its security provisions that would prevent impersonation or other malicious actions. For example, an end user is much more likely to trust a manufacturer if the later includes verifiable protections in their production and supply chains.

History: Variations in the above factors or other aspects of its operation over time affect an entity's future reputation. Recent history is often viewed as more important but there can exist certain events that permanently harm the reputation.

Whichever the source, reputation and trust are only valuable when the respective entity can prove its identity as the intended relationship partner. Therefore, proof of unclonability can be an important factor in building and maintaining relationships, be they between people or machines. At the same time however, it is only beneficial to attest

the unclonability of an entity when it already exhibits a certain amount of reputation. This creates an interesting chain of validation that is initiated by the *unclonable root of trust*.

2.3 Unclonability Primitive

2.3.1 Definition

The first step to formulating the notion of unclonability is to clarify the meaning of *clone*. Clones can appear on two levels: mathematical and physical[10]. *Mathematical clones* essentially treat the original object as a black box and try to emulate its behaviour, usually generating the same mapping of inputs to outputs. *Physical clones* on the other hand, are identical copies of the physical structure of the object in detail.

In our work, we consider physical unclonability as the root for unclonable systems, and thus we begin by discussing unclonability in a physical context.Cloning an object can have varying levels of success in practice, and it is difficult to achieve a perfect cloning result. Therefore, the task of cloning and that of clone detection are evaluated by the complexity needed to produce satisfactory results. To make clone detection possible (and thus achieve unclonability) we need to draw on one or more features which are inherent to the object and beyond any level of control that would allow their exact reproduction, given the available technology.

Unclonability refers to the difficulty in controlling all the features of an object in a meaningful way, with the aim of producing a clone that is, or appears to be, an exact copy of the original object. Thus, the following prerequisites are required for an object to be unclonable:

- The presence of *individualising features* which differentiate it from other similar or dissimilar objects[10].
- The ability of an observer to measure those features in a quantifiable manner and utilise the results.
- The persistence of those features over the lifetime of the object or the time of interest.

For formal definitions of unclonability and clones we refer to the work of Maes[10].

Even though unclonability can be considered a property than an object 'has' or 'does not have', harnessing this property is a different matter. Most physical objects can be considered unclonable if one is able to examine them in enough detail but, in order to exploit unclonability in practice, efficient and dependable observation methods are required. Additionally, these methods need to provide significantly different results across the entirety of object instances than can be manufactured. Finally, the process of creating an object instance needs to be such that there is a very low probability of generating two identical objects, either by chance or via malicious interference.

In summary, the individualising features are required to have the following properties, which allow them to be utilised in practice:

Unclonability: Being hard to thoroughly copy or otherwise reproduce.

Modelling resistant: Exhibiting behaviour that is hard to represent with a mathematical model.

Small intra-distance: Generating the same response over multiple observations of the same object, up to a bounded error.

Large inter-distance: Generating very different responses over multiple observations of different objects.

Observability: Providing a practical and efficient way of observation.

Stability: Retaining the same value over time.

According to Maes[10], two variants of unclonability can be defined: *existential unclonability* refers to the difficulty of creating two objects that are a clone of each other, and *selective unclonability* refers the difficulty of creating a clone of a given object. The first variant suggests a stronger adversary (since it implies the second variant) but both have implications in security provisions, albeit on different levels. An adversary who is able to produce two or more identical objects will be required to also deploy one of them as a legitimate object, for example with a supply chain attack. On the other hand, if the adversary is able to selectively replicate a legitimate, trusted object, no further physical effort is needed. Both contexts are considered in our work, where we aim to detect both the replacement of existing network nodes, and the introduction of new nodes, as discussed in Section 2.4.5.

2.3.2 Extending Physical Unclonability

Unclonability is closely related to ownership, since it provides powerful control over objects, often to a micron level. Societies regularly operate on the implicit assumption of ownership and identity, and unclonability is the means to establish these assumptions over the human-machine boundary.

In the words of Ferguson et al., 'the function of cryptographic protocols is to minimise the amount of trust required'[11]. The creation of novel cryptographic protocols based on unclonability will help cultivate security procedures that are suitable for modern societies and the recent networked revolution, essentially transforming the *Internet of Things* into an *Internet of People*. In essence, the inherent unclonability of people can be transferred via the creation of unclonable devices to unclonable networks and, eventually, to unclonable systems.

The value of objects and relationships is often determined by their scarcity which means that cloning an object may directly diminish its value. Scarcity or uniqueness can

be approached with the notion of *atomicity*. Our understanding of atomicity is based on its Greek root 'άτομο' which has a dual meaning. It can either mean 'undivided' (as in 'atomic transactions' of databases) or 'individual'. An atomic object is unique in its context and its identity cannot be duplicated or transferred, further supporting the notion of ownership. In other words, atomicity encompasses the intrinsic unclonability characteristics of an object, together with its context.

The combination of these issues with the modern practice of online interaction, leads to the need to unquestionably prove the identities of humans and machines without allowing their replication. By injecting unclonability in networking protocols, which are already being increasingly used in the place of human interactions, we aim to include machines in the unclonability domain that, until recently, was only encompassing human relationships.

Physical Unclonable Functions (PUFs), which we discuss in Chapter 4, are the most promising provider of unclonability in digital devices. To take full advantage of PUFs and their novel properties, we construct an unclonability framework for the Internet of Things, organised in the form of the unclonability stack outlined in the next section.

Unclonable systems built on PUFs provide several security advantages. Firstly, they make use of secrets that are 'naturally occurring' in hardware without any intervention, with many PUFs using widely available hardware. This can be seen as the first step towards 'secret-free' security[12]. Additionally, unclonable secrets provide novel security guarantees in comparison to static, predefined secrets. These guarantees remove a number of implicit trust relationships throughout the lifetime of devices and establish strong ownership relationships instead. Finally, the increased strength of the secrets allows for reduced resource requirements, which, in conjunction with the lower provisioning costs, lowers the barrier for introducing robust security methods to the IoT domain.

2.4 Unclonability Stack

Exploiting the unclonability primitive, we can construct an *unclonability stack* as a collection of layers built on unclonability to enable novel applications. The stack is summarised in Fig. 2.1 and can be divided into two domains: physical and logical. The layers of the physical domain are implemented and supported in hardware, in order to ensure the security of the unclonability provider. On the other hand, the logical domain is mainly concerned with the higher level interactions of the system and can be implemented in software or hardware.

Every layer provides a set of services to its higher-level counterparts, creating a modular stack where individual layers can be modified or replaced. This modularity greatly reduces the system complexity, since high level protocols can transparently use the unclonability source while staying oblivious of its implementation details. In addition, parts of the stack can be revised based on the requirements of certain applications or future developments in the underlying cryptographic primitives.

Despite resembling the OSI model, the unclonability stack is concerned with the se-

curity interactions between devices and systems,rather than the exchange of application data. Consequently, 'traditional' communication methods are required, and the stack is designed to provide additional security features to existing communication infrastructure.

In the following sections, we outline the purpose and services of each stack layer.

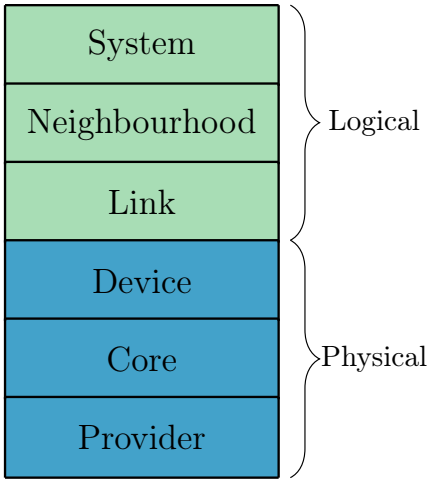


Figure 2.1: Unclonability stack

2.4.1 Provider

The Provider is a low-level hardware construction which generates the physical unclonability, serving as the *unclonable root of trust*. This first layer of abstraction is responsible for the representation of the underlying physical processes in a digital form that can be subsequently used by the Core layer and allows us to reason about unclonability protocols disregarding the implementation details of the hardware.

Every unclonability provider that is currently available can be grouped under the umbrella term 'Physical Unclonable Functions'. Nevertheless, the Provider layer conceals hardware implementation details and exhibits the same I/O behaviour as seen in Section 4.3, regardless of the PUF class used. As a result, existing or future constructions, whether they are PUFs or a different primitive can be used as unclonability providers.

2.4.2 Core

With the Provider as its foundation, the Core supplies the higher layers with cryptographic operations such as key generation, encryption, and signature. Its main aim is to allow access to the Provider through a secure interface with a limited number of strictly defined features. This interface is the only means of communication with the Core, reducing the attack surface against the Provider and ensuring that breaching attempts can be detected. Furthermore, the Core encapsulates all the operations that process private information and is thus the only hardware component that requires physical protection.

Fig. 2.2 shows the architecture of one of the possible embodiments of the Core. The above benefits can only be supported by a hardware implementation, which would also provide additional advantages. Firstly, the acceleration of complex operations is possible,

a feature that is often important in cryptographic scenarios. Secondly, higher quality primitives can be used, for example in the case of the random number generator where hardware TRNGs are preferred over their software counterparts. Finally, physical protection methods can be used to detect and deter tampering attempts, a task that is considerably more difficult in software.

As seen in Fig. 2.2 and discussed in detail in Chapter 7, the Core can be constructed with existing, well-established components which will have a minimal overhead both in terms of development costs and resource consumption.

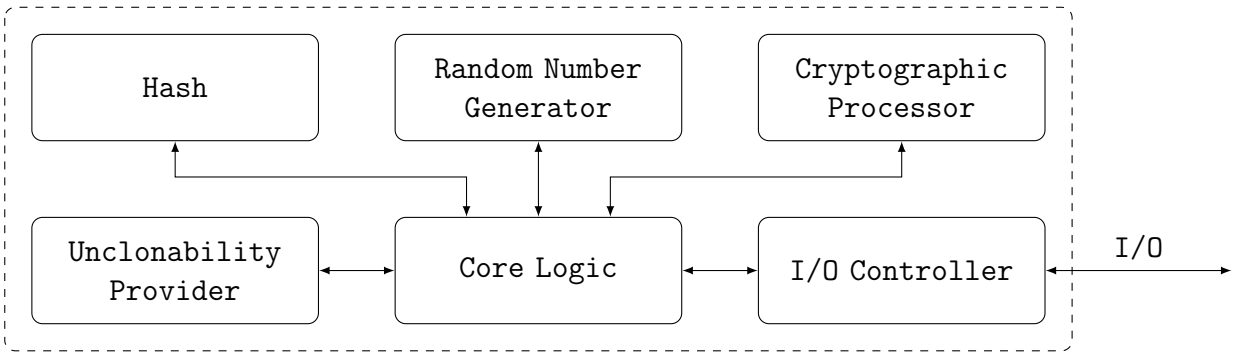


Figure 2.2: Unclonable core reference architecture

2.4.3 Device

Atomicity is exceptionally difficult to achieve in networked systems where devices are assumed to be identical to each other, forming a swarm of perceived clones. While this assumption simplifies the development of large systems, it also hinders the operation of sophisticated security methods. Being unable to differentiate between nodes, security protocols can only distinguish between classes of devices instead of individual devices.

In the past, device identification was based on a secret of any form that had to be generated, safely stored and recalled every time it was used. However, in IoT scenarios, the mathematical guarantees provided via cryptographic means are becoming increasingly irrelevant since attackers have access to the device hardware, allowing them to recover secrets through physical attacks. Furthermore, a lot of applications involve a great number of devices, making the process of generating and storing unique secrets inefficient. An unclonable node is able to provide a secure and high-entropy method of generating a unique secret on the device itself, recreating it every time it is needed, without the need for secure storage.

We define an *unclonable device* as a generic computing device with the components summarised below. A reference architecture with these components is shown in Fig. 2.3.

Unclonability Core: A self-contained block as it was described in Section 2.4.2, providing a *root of unclonability*.

Unclonability framework logic: Implementing the necessary features to provide

higher levels of the stack with access to the unclonable core.

Application logic: Providing capabilities required by the application.

Communication Interfaces: Networking and other I/O interfaces, employed by the unclonability framework and the application logic.

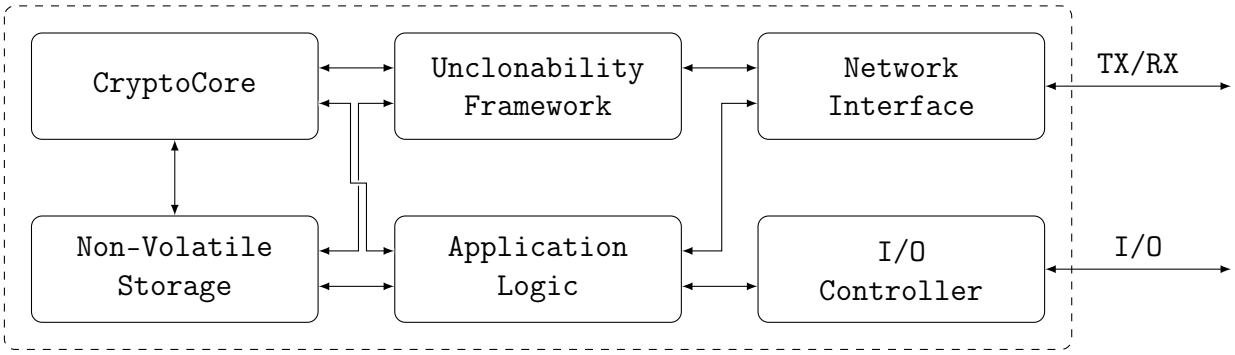


Figure 2.3: Unclonable device reference architecture

Exploiting this architecture, an unclonable device is able to securely generate secrets that are bound to the intrinsic physical properties of its hardware, and carry out the operations needed to support the unclonability goals of higher layers. On a conceptual level, the boundaries between the Core and the Device can be fuzzy but, in essence, the Device usually includes more complex functionality to serve the purpose of the application. This functionality is unrelated to the security aspects of the Device and often expands the attack surface for potential adversaries. As a result, a logical and physical boundary between the Core and Device is highly advisable.

2.4.4 Links

An unclonable device is rarely useful in isolation, but has great value when interacting with other devices. Given two unclonable devices it is possible to generate unclonable links, which are established by a mutual authentication process and maintained by periodically refreshing the authentication state. This procedure injects unclonability in every authentication round and enables the detection of changes in the state of the peers, creating strong trust relationships.

The qualitative difference between conventional network links and unclonable links is that the latter encapsulate the state of both peers as well as their interactions. Secure links could potentially be created through cryptographic means without unclonability, and the security of these links relies on the assumption that access to current and past cryptographic secrets is restricted. However, this is not always a reasonable assumption. The inclusion of unclonability, and specifically the use of PUFs, makes it considerably easier to secure the aforementioned secrets. Additionally, it is possible to split unclonable secrets into smaller parts, that can be used independently. These parts can be utilised

to support periodic authentication protocols without reusing the same secrets, a method that is widely accepted to strengthen the security of cryptographic protocols[13].

As we discuss below, unclonable links are an exceedingly important building block that can be moulded to support several novel features. In Chapter 6, we propose PUF-based methods for instituting such links and their resulting mutual relationships between nodes with limited resources.

2.4.5 Neighbourhood and System

Belonging to a group, be it a human organisation or a network cluster provides a number of benefits to its members. For starters, members are recognised by their peers (often through esoteric dialects or protocols) and enjoy their trust, which in turn allows them access to data and information that is considered private to the group. The collective identity, which is gained upon joining, augments the identities of the members not only inside the group but also in external activities. For example, it can be used to access services from both members and non-members, including task outsourcing, resource sharing, or packet forwarding.

The topmost layers of the unclonability stack concern the organisation of multiple pairs of nodes into groups or *'neighbourhoods'* and the organisation of neighbourhoods into systems. There is no constraint in the maximum number of members in either of these groupings, however, at least three nodes are required to compose a neighbourhood and at least two neighbourhoods are required to compose a system.

Neighbourhoods extend the features of the link layer from pairwise connections to any kind of topology, creating a multiplicity of relationships between nodes, be they immediate or separated by multiple hops. In this way, unauthorised distortions of the network graph can be detected and handled on a collective level, and the topology of the neighbourhood can be exploited to provide additional security guarantees, including relationship redundancy and obfuscation of traffic patterns.

Systems are composed in a similar manner, with a number of neighbourhoods interacting through pairwise protocols. These protocols differ from the intra-neighbourhood protocols in that they are required to protect any transmitted internal neighbourhood information by enclosing it in secure wrappers. For example, we can visualise such an arrangement in a military context where units operate as neighbourhoods, and communication between units requires an additional level of security to avoid compromise during transit. Additionally, it is generally good practice to limit the amount of information shared between units, in order to enable the containment of damage in case of a compromise.

In this thesis, we mainly consider the organisation of neighbourhoods, with system level interactions being part of future work. *Neighbourhood unclonability* can be realised with protocols which take into account both the topology of the system and the relationships between neighbouring nodes. Two challenges exist in forming neighbourhoods: node introduction, and creation of node relationships. The first step is to specify inter-

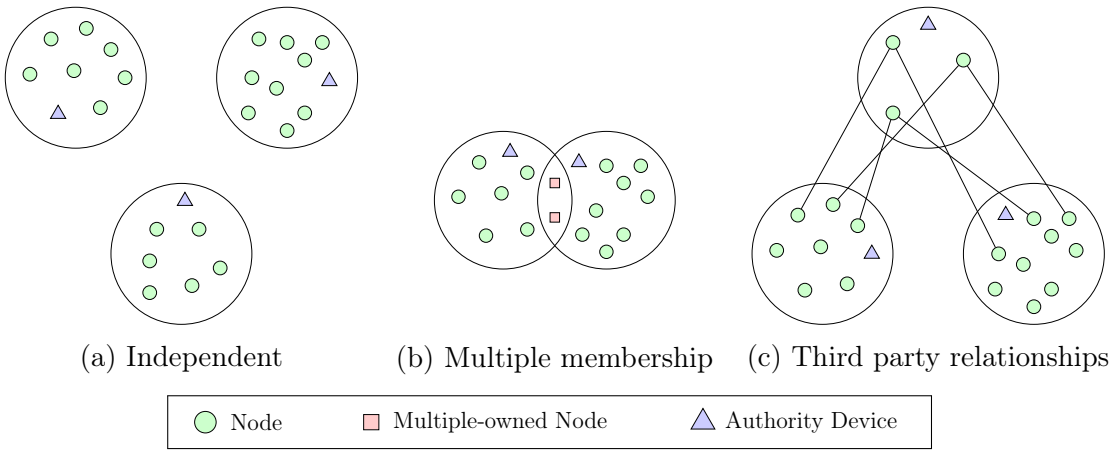


Figure 2.4: Examples of neighbourhood topologies

action protocols for the nodes, with methods akin to the ones discussed in Chapter 5. Each neighbourhood is controlled by a single authority device, and nodes can simultaneously belong to multiple neighbourhoods, if required by the application. Following their introduction, nodes establish unclonable links, and utilise those links to make collective decisions about the state of the neighbourhood.

Since pairwise links are the basic building block for any network topology, neighbourhoods can be created in every network. However, topologies with increased redundancy (for example mesh or bus topologies) are advantageous for providing robust monitoring of neighbourhood behaviour. This is because the sensitivity of monitoring protocols to single node faults decreases with the number of monitoring nodes. Some indicative cases of neighbourhood organisation are illustrated in Fig. 2.4 and discussed in detail in Chapter 5.

Topology Distortion Taxonomy

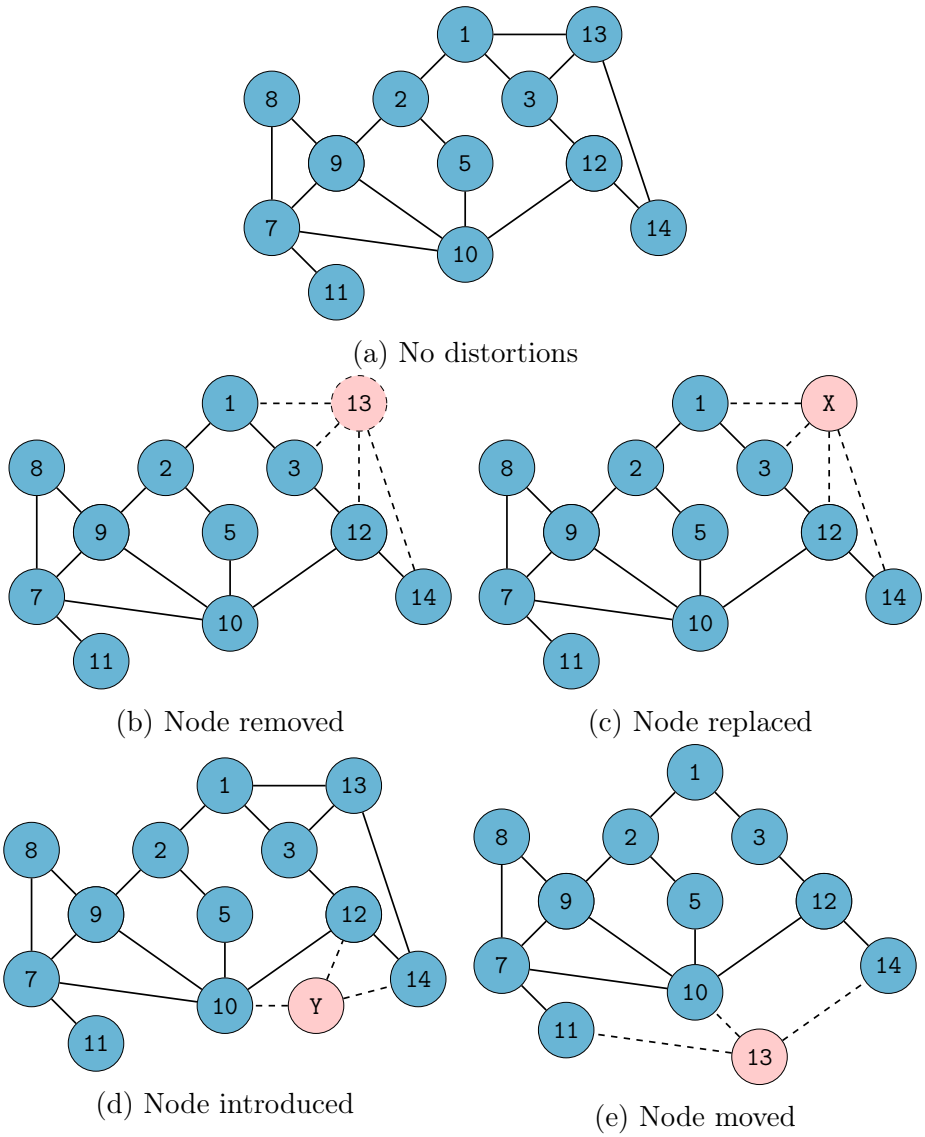
Based on unclonable neighbourhoods we can create a taxonomy of *topology distortions*, with the aim of developing distortion detection methods. Distortions modify the structure of the network and can be the result of an attack by an adversary or simply an unexpected event. With the assumption that a single node is affected, we consider the distortions shown in Fig. 2.5:

Node removed An known node is removed from the network, indicating a potential physical event, such as node destruction or compromise.

Node replaced A known node is replaced by an unknown, possibly malicious or compromised node.

Node introduced A previously unknown node is introduced, without any information about its origin.

Node moved A known node is moved to a different logical position, indicating a physical intervention in the system.



(a) No distortions

(b) Node removed

(c) Node replaced

(d) Node introduced

(e) Node moved

Figure 2.5: Topology distortions

When a distortion occurs, trust relationships with the affected node are considered broken, at least until the source of the distortion is established. As a result, depending on the application, any data that the node is handling (or has handled in the past) is considered malicious or exposed to adversaries. Additionally, future node actions, for example reports about the state of its neighbours are deemed false.

Neighbourhood unclonability protocols are able detect the above distortions and take actions which include alerting a higher level party, destroying secrets, and initiating recovery procedures. Nonetheless, it is important to make the distinction between distortion detection and distortion recovery. For the purposes of security, it is acceptable and even desirable for a system to seize its normal operation after a distortion occurs, requiring an *exceptional 'authority action'* for recovery. This feature ensures that the system will operate in a predictable manner in case of an attack, and is integrated in the authentication protocols discussed in Chapter 6.

2.5 Challenges in Designing Unclonability Protocols

Harnessing the unclonability provided by physical constructions and utilising it in security protocols presents a unique set of challenges. To begin with, the presence of unclonability is not directly measurable and can be found in different forms and measures. While we often consider it in a binary manner of existing or not existing, unclonability can be manifested in certain parts of a system and be absent in others or, more commonly, it can be present 'by association', as discussed in the previous sections. There have been several attempts to formalise unclonability in the context of PUFs [10], [14] but, to the best of our knowledge, no method has been provided that would allow the evaluation and comparison of PUF protocols in an unclonability context. Conventional approaches still apply, however they fail to capture the full potential of the primitive, as their are designed to evaluate 'traditional' cryptographic systems.

Additionally, designing the protocols themselves involves providing answers to two main questions: how can we prove the possession of a secret produced by stochastic physical processes without exposing it, and how can this proof be maintained in time? The first question can be tackled with traditional cryptographic methods including, for example, zero knowledge proofs. However, when we come to the second challenge it becomes evident that it is not sufficient to prove momentary unclonability but instead a temporal 'proof chain' is required. Thus, PUFs (or other unclonability providers) need to be made integral parts of the protocols, continuously injecting fresh unclonability into the system, rather than serving as merely a randomised key generator.

Finally, we tend to consider unclonability as an infinite resource that can be used over and over for as long as it is required. Nevertheless, that is not the case, especially in the context of PUFs. While PUF behaviour remains fairly robust over the lifetime of the hardware, the majority of PUF constructions are not able to provide an infinite amount of outputs. Thus, security protocols need to either have a limited lifetime or account for unclonability depletion and include provisions for its conservation and restoration.

2.6 Unclonability Framework

The creation of an unclonability stack, as it was introduced in Section 2.4, requires the development and orchestration of a variety of methods and protocols. The network nodes need to be rendered unclonable through their unclonability provider, grouped in neighbourhoods, and start interacting in a manner that will allow for detection of topology distortions.

To that extend, we formulate, construct, and evaluate methods on every layer of the unclonability stack, as summarised in the next sections and outlined in Fig. 2.6. These methods include the generation of trust from PUFs, the establishment of trust relationships in pairs and groups of network nodes, and the maintenance of those relationships over the lifetime of the system.

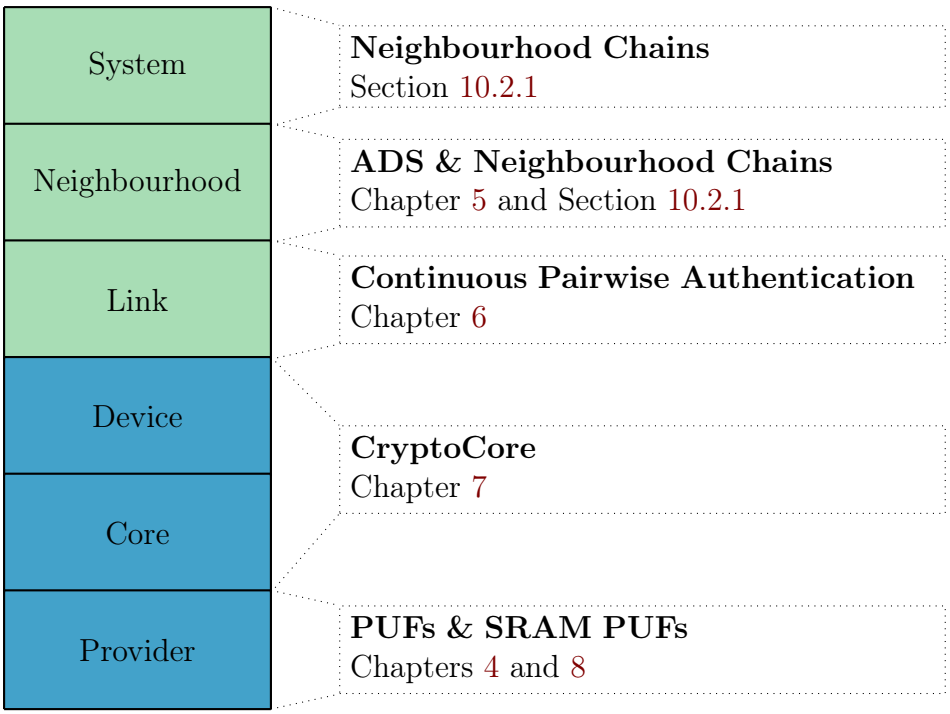


Figure 2.6: Unclonability framework

Neighbourhood Management: Authority Device Scheme

In networking terms, neighbourhoods of nodes are equivalent to clusters. As is the case in a town neighbourhood, neighbours need to be introduced to each other prior to forming trust relationships. Node introduction requires the approval of a higher level entity which is not necessarily part of the neighbourhood. This higher level entity is in turn part of an additional layer of relationships. A few of these layers can exist between the owner of the system and the nodes, for example: system owner → system administrator → authority device(s) → nodes.

Researchers have proposed numerous solutions for introducing devices to each other, including the assumption of a secure environment[15] or the use of user input like PIN codes[16]. We developed a hybrid solution that can be adapted to different applications. Our *Authority Device Scheme*, makes use of authority devices (ADs) held by a trusted party to bootstrap the operation of the system and introduce nodes to each other, enabling them to form clusters. This is achieved by using asymmetric cryptography with key pairs generated on demand by PUFs. After the initialisation phase the ADs can be removed or even destroyed as the nodes operate autonomously and the scheme allows for a 'pre-enrolment' in a secure environment. The pre-enrolment satisfies the security requirements of some applications but can be omitted if not required. Further details of the scheme are discussed in Chapter 5.

Distortion Detection: Continuous Authentication

Going back to our city neighbourhood analogy, people often observe the behaviour of their neighbours and compare it to commonly accepted patterns. In case of a mismatch,

the trust relationships between neighbours are weakened or even broken. In the same way, to maintain the trust relationships between nodes, a continuous 'loop' needs to take place, refreshing the unclonability guarantees. In this loop, nodes periodically exchange authentication information in a continuous pairwise authentication protocol discussed in Chapter 6.

Due to the redundancy that is inherently present in most network topologies, the nodes are able to take advantage of their pairwise relationships to monitor their neighbours and take collective action when unexpected events are detected. Using methods such as the ones outlined in Section 10.2.1, an additional layer of periodic protocols can be constructed, providing unclonability at the neighbourhood level.

3. Security Concepts

Before we delve into the domain of PUFs and their unclonability properties, we review the basic concepts of security, in the context of our work. Many of the methods discussed in this chapter have been the result of a great amount of research and have been included in real world implementations for decades. As a result, each of the following sections can be considered a distinct research area, and thus our aim is to set the foundations for the rest of this thesis, rather than include every possible detail. We endeavour to cite some of the wealth of reference material, to which the reader can refer for further information on the topics covered in this chapter.

3.1 Network Security

The *Internet of Things (IoT)* is a broad term describing a world of networked objects and devices of all kinds. These devices are deeply integrated into everyday life allowing for ubiquitous interaction and communication among humans and machines. Interestingly, humans are no longer in the centre of the architecture, as smarter versions of everyday 'things' are becoming able to self-configure and self-adapt, mainly through collaboration, with little or no human intervention. The underlying structure supporting the IoT can be more aptly described by the term *Machine-to-Machine(M2M) networks* which refers to the communication infrastructure between small, often resource-constrained heterogeneous devices.

M2M platforms define a novel paradigm of user-object interaction which leads to the need of security methods that allow the aforementioned systems to be 'secure by default'[17]. A wealth of security provisions have been developed in the past decades since the inception of the Internet, but unfortunately some of these methods are not directly applicable to the M2M world, since they were designed for different system models. In consequence, many currently available commercial M2M solutions are not secured properly and have been compromised, signifying the great need for focused security efforts[18].

3.1.1 Goals

The wide range of security goals in a variety of networking applications can be reduced to the basic principles of confidentiality, integrity, availability, non-repudiation, and privacy[19]. Not all of these concepts are required in every application, nevertheless they form the basis of modern security provisions. To begin with, *confidentiality* refers to the prevention of unauthorised access to private information, either in transit or at rest. Con-

fidentiality is usually achieved with encryption methods that enforce certain restrictions to the properties of the entities which are allowed access to the information. *Privacy* is also often connected to confidentiality, although the latter does not necessarily mean the former. In order to achieve privacy, especially in a connected world, every aspect of the protected information needs to remain secret from unauthorised parties. A common example of methods which lack privacy are encrypted messaging platforms where the content of the messages is confidential but relevant metadata (e.g. source, destination, encoding etc.) is exposed.

When transmitting information over a network, it is often the case that the communication channel is not fully trusted. As a result, the *integrity* of the information needs to be protected against adversaries who can intercept and potentially modify the transmission. In contrast to confidentiality, adversaries are usually not actively prevented from making such modifications but provisions are put in place to ensure the detection of these events. These provisions are designed to be hardened against malicious actions but have the additional benefit of detecting accidental integrity lapses, which are usually the result of unreliable communication channels.

Safeguarding the *availability* of a system and its associated data is not only related to attacks on the communication medium but also the system itself via physical and logical means. Despite often being overlooked in security protocols, the availability of a system is of paramount importance since no other methods or protocols have any meaning if the system is not in operation when needed. The most common attacks against the availability of a network are the denial-of-service attacks, which aim to deplete system resources and are becoming increasingly relevant in the IoT domain.

Finally, *non-repudiation* is the most abstract of these goals, stemming from a legal context. Essentially, there are applications where it is desirable for entities to perform actions that they are not able to deny performing in the future. This requirement is commonly connected with monetary applications such as banking systems or cryptocurrency exchanges. Non-repudiation is not easy to achieve in practice due to the difficulty of unquestionably relating data to specific devices and individuals. However, unclonability could be a step in the right direction.

3.1.2 Common Tasks

The security goals discussed above are often achieved through cryptographic means, namely encryption and signature methods. These methods provide solutions to several tasks that commonly appear in the context of network security, while also creating some challenges of their own[20].

Key Provisioning

Designing cryptographic systems to rely on secret keys for their security has several advantages, including greatly simplifying the protection of said security. Nevertheless, key generation, distribution, and storage are not simple tasks. The majority of cryptographic

algorithms require keys of high quality which usually translates to long, random data blocks. Additionally, the keys need to be known by at least two parties to achieve the goals discussed above. Consequently, there is a need to generate keys and distribute them to the relevant parties in an efficient manner, while keeping them secure from adversaries. Numerous works have approached the issue of key distribution using a range of primitives. Distribution schemes are categorised in centralised and decentralised variants. Centralised distribution has the advantage of lower complexity and higher scalability, usually relying on multicast transmissions[21]. On the other hand, decentralised schemes avoid single points of failure at the expense of scalability and network overhead, since they typically require more coordination among nodes. Additionally, decentralisation is usually achieved via cryptographic means which have their own complexity overhead[22], [23]. Matters are slightly simpler for asymmetric cryptography since only public information needs to be shared. However, centralised distribution methods, commonly called Public Key Infrastructure (PKI) are still necessary.

In practice, especially in consumer devices, the costs associated with key infrastructure are not justified from the point of view of the manufacturers. Consumers rarely have the skills to recognise the difference and the market is not regulated in that respect. Thus, most products end up with only basic security provisions, including hard coded keys which are either shared among all device instances, or deterministically generated from public information (for example a device's MAC address). Even when random, per-device keys are used, the fact that the manufacturer is involved places unnecessary trust in the latter. One of the advantages of PUF-based key provisioning methods is the ability for manufacturers to embed unpredictable, unique secrets in all of their devices with minimal effort and cost.

Identification and Authentication

The distinction between entity identification and entity authentication is often hard to formulate. This is partially due to several authors assuming these terms are synonyms (e.g. Katz et al. in [24]), and other authors making an implicit distinction (e.g. Maes in [10]). In this work we adopt the definition of authentication by Boyd and Mathuria[20]:

Definition 3.1. *Entity authentication is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired).*

Identification is regarded as a 'weak' authentication, requiring fewer guarantees on behalf of the prover. In the field of PUFs, this often translates to a fuzzy comparison of responses, for example by using a Hamming distance threshold to evaluate matching responses. Authentication on the other hand is based on complete matching of responses which is often associated with a distance metric of zero.

Authentication protocols are not concerned with the security of application data in

any way, and only aim to provide guarantees that only the right entities are allowed to successfully authenticate. Consequently, independent methods are to be employed when the application requirements include data confidentiality and integrity. However, authentication often constitutes the first stepping stone towards achieving the goals discussed in the previous section and is of such importance that it is required in the majority of the modern encryption systems, in the form of authenticated encryption.

3.1.3 Topologies

The structure of M2M networks is generally based on three main components: the devices, the gateway and the cloud. The number of devices is usually much larger than that of the gateways, although in some cases there is no clear distinction between a device and a gateway. In our work we mainly consider topologies of direct communication between devices. For simplicity, in this section we assume a single gateway and a single cloud instance, although the same principles can be expanded to other cases. With this assumption, we briefly review the topologies of M2M networks.

Device-to-Device

In the simplest topology, devices communicate directly with each other ('ad-hoc' communication). The transmission of data is based on mesh multi-hop logic and short-range technologies are used including Bluetooth, WiFi, and ZigBee. Security provisions are mostly based on direct, pairwise relationships between the devices. While convenient and suitable for many applications, this topology lacks scalability. Additionally, due to the resource constraints of the devices, the range of their transceivers is usually limited and thus multi-hop communication is needed to cover larger areas. Unfortunately, multi-hop communication involves a number of issues including higher energy consumption, more complex routing schemes and higher probability of network congestion.

Device-to-Gateway-to-Device

A frequently used solution to some of the limitations of the Device-to-Device topology is the inclusion of a gateway node which acts as an intermediate between devices. Gateways are usually higher-end compared to devices and can thus have an improved transmission range as well as perform additional tasks like complex routing or calculations on the transmitted data. Through the use of a gateway, a star topology is created and gateways can support multiple radio technologies, acting as bridges and, in some cases, routers between heterogeneous devices. From a security perspective, the gateway can either be transparent and merely forward the data to its destination or it can act as an additional party in any secure communication. For example, the latter configuration is needed when the gateway performs any kind of processing on the forwarded data.

Device-to-Gateway-to-Cloud

Gateways can also be connected to a wider network via the Internet or some other WAN connection. This allows the use of advanced functions including aggregation and monitoring services which are hosted on a cloud platform. Cloud platforms are typically deployed on data centres residing in different physical locations than the M2M network, and are often provided by third parties. The advantages of such a topology are the ease of deployment and maintenance, as well as the very high availability of cloud services. However, the cloud providers are able to both control the flow of data and eavesdrop on the content, if appropriate privacy and confidentiality methods are not employed. This high level of control creates a strong dependency on the cloud platform. As a result, cloud providers often attempt to monetise every aspect of their offering by encouraging vendor lock-in practices.

Device-to-Cloud

Directly connecting the devices to the cloud is becoming an increasingly popular choice, appearing in various commercial products. Two topologies can usually be seen in this case: the devices are either equipped with a mobile network module, or a local network is used to provide Internet access. The latter case is very similar to the gateway topology, with the distinction that devices do not make use of this gateway to directly communicate with each other. While the cost of including a long-range communication module in embedded devices is constantly decreasing, the devices are responsible for securing their data in transit. Additionally, many of the devices can be mobile or installed in public places, making it possible for an adversary to gain physical access to them. On the other hand, gateways are easier to physically isolate and protect as, in most cases, they can be physically separated from the devices they serve.

3.1.4 Attacks

Having established the common topologies and goals of M2M systems, we review prevalent attacks against popular security provisions. On a high level, attacks can be categorised as *physical attacks* where adversaries have access to the system hardware, and *protocol attacks* where adversary access is limited to the communication medium. The same classification is also referred to as *passive* or *active* attacks, as seen from the point of view of an adversary. Passive adversaries aim to circumvent the security provisions of the system by observing its operation and intercepting transmissions, without interfering with any part of the system. Such methods usually target the employed cryptographic primitives, gathering information and ciphertexts with the hope of deriving some of the secrets. On the contrary, *active* adversaries aim to modify the operation of the system in order to force it into a vulnerable state. These adversaries can use *invasive* methods, targeting system hardware, often with irreversible results, or *non-invasive* methods, focusing on the manipulation of system data and behaviour.

Protocol attacks intend to analyse, disrupt, and manipulate data flow to the benefit

of the adversary. Due to the diversity of network topologies and applications, a large number of similar attacks have been devised. We summarise the main categories below, and refer the reader to Boyd et al.[20] for a detailed review:

- *Eavesdropping* attacks involve the passive collection and analysis of transmitted data and traffic patterns.

- The messages captured by eavesdropping can be used in *replay* at a later time, often with aim of forcing a predictable sequence of events.

- When integrity protections are not in place, messages can also be modified by adversaries before they reach their destination. This attack is often reoffered to as *'spoofing'*.

- Similarly, in *man-in-the-middle* attacks the adversary acts as a transparent proxy between the communicating parties. Since the intended parties are oblivious to the attack, they proceed to establish a 'secure' connection with the attacker effectively negating any security provisions.

- *Denial-of-service* attacks, as mentioned earlier, aim to overload the system with a large number of requests, leading to depletion of its resources and eventually impairing its availability.

- Finally, an attacker controlling a device or a gateway can perform *black hole* attacks where all incoming packets are silently dropped. A more sophisticated variant is the *greyhound* attack where packets are dropped selectively, based on some condition, in order to avoid detection.

3.2 Hardware-backed Security

For the reasons discussed above, security methods assisted by hardware are increasingly popular. These methods utilise hardware primitives to enhance their performance or provide additional features. We refer to such solutions as 'hardware-backed security', as opposed to simply 'hardware security', which in our view includes methods for securing the hardware itself.

The two main reasons for employing specialised security hardware are often overlapping. As a general rule, hardware implementations of any algorithm are orders of magnitude faster than their software counterparts. In addition, the use of a distinct hardware module allows for the isolation of security-sensitive components from both the software and other hardware, making them easier to protect. In the following sections we review common examples of these methods, which often coexist in the same component.

3.2.1 Attacks and Countermeasures

Current state-of-the-art cryptographic primitives have been the object of decades of research and evaluation in practice. Due to this maturity, cryptanalytic attacks are rare and adversaries rely on implementation shortcomings to perform so-called 'side channel' attacks. These methods are based on the observation of the system behaviour over time in an invasive or non-invasive manner. The majority of hardware side channel attacks employ a probing infrastructure and thus are often noticeable. Additionally, a fairly large observation sample is required in order to perform a successful attack, meaning that such methods are usually unsuitable for closely monitored systems.

There is a wide variety of metrics that can give away partial information about system secrets, including the timing between operations, power consumption, radiation and acoustic emissions. Power consumption is one of the most frequently used metrics, in an attack called *Differential Power Analysis (DPA)*. In DPA, the adversary measures the power consumption of the cryptographic implementation for extended periods of time and uses this information to derive parts of, or even the whole encryption key[25], [26]. Different parts of the cryptographic implementation IC are activated depending one the values of the key bits and thus the power consumption can be correlated with the key.

Another common side channel attack is based on the time needed to complete an operation (i.e. encryption)[26], [27]. Certain implementations behave differently depending on the key value, for example skipping some calculations altogether if a specific key bit has a particular value. Therefore, if the adversary has a knowledge of the employed cryptographic algorithm, it is possible to use the timing information to deduce parts of the secret key. Timing attacks are generally less profitable than DPA, due to the fact that they rarely produce the full key. They do however assist in reducing the search space for brute force attacks.

Fault attacks are a different category of physical attacks. They involve fault injection techniques which aim to manipulate the control flow of the system and force it into an unsafe or erroneous state. For example, a clock glitch can make the CPU omit the execution of a branch instruction, leading to the execution of invalid code[28]. Fault attacks are becoming increasingly important as the push for performance requires the fabrication of ICs with increasing density, rendering them more vulnerable to faults. One infamous case which took advantage of this issue was the 'Rowhammer' attack on DRAM ICs[29].

Countermeasures

Two classes of countermeasures can be used against side channel attacks. Firstly, the implementation can be adjusted in order to avoid common attack vectors, such as power analysis and timing[27]. In most cases, this method of defence involves additional development time but does not usually require hardware overhead. Nevertheless, it often has an impact on performance since it disallows the use of certain optimisations. It is also

not always possible to obfuscate every side channel, especially because new ones can be discovered after fabrication.

A more comprehensive, albeit costly, coverage is provided by the second class of defences which relies on physical protection[30]. These include the use of specialised hardware encapsulating the actual logic circuits without considerable modifications to it. The purpose of this hardware is to provide either tamper-evidence or detection of intrusion attempts upon which internal secrets are erased. Depending on their intended cost and application, physical protection methods can range from simple epoxy blobs (designed to destroy the IC when tampering occurs) to sophisticated meshes of sensors detecting all kinds of changes to temperature, light or electromagnetic fields. The drawbacks of these methods are their relatively high cost and their heavy reliance on the limited capabilities of the adversary. Some of the added cost can be avoided by physically partitioning the IC, creating smaller areas which process sensitive data and which are easier to protect. Nonetheless, physical protections are highly successful in providing tamper-evidence and deterring the majority of 'casual' attackers.

No universal countermeasures exist that would cover all side channel attacks. Even if they did, the cost of implementation would be prohibitive for their inclusion in commercial products. Thus, every defence strategy needs to consider the capabilities of the adversary and the actual impact of every individual attack vector. PUFs are not capable of preventing every attack of this type, without additional provisions, but they are a relatively cheap and convenient way to increase the difficulty of invasive attacks, especially in the context of secure key storage.

Transistor or circuit level modifications are needed to defend against fault attacks. While there is no direct way to prevent the attacks themselves, their effect can be lessened by designing fault-resilient circuits. These circuits employ redundancy and error correction to reduce the influence of faults to their output. Unfortunately, additional logic comes with associated costs, both in silicon area and increased exposure to side-channel attacks[28].

3.2.2 Cryptographic Processors

In order to keep up with the demand for faster and more reliable cryptography, security engineers have to maximise the performance of both algorithms and hardware. To that end, a number of hardware solutions have been proposed, generally referred to as cryptographic processors or 'cryptoprocessors'[31]. The most important advantages and disadvantages of each method are discussed below and summarised in Table 3.1.

The different architectures of these solutions can be categorised in two groups: crypto-processors and crypto-coprocessors. The difference between the two is often minimal and thus the distinction is frequently disregarded. Cryptoprocessors are complete -albeit specific purpose- processors, with their own registers and instruction set, while crypto-coprocessors are merely a collection of ICs that perform cryptographic tasks and share the system buses and memory. From a security viewpoint, cryptoprocessors are usually

preffered since they enable the containment of all private material in a single IC, and only protected (i.e. encrypted) data is communicated to the rest of the system. A generic cryptoprocessor architecture is presented in Fig. 3.1.

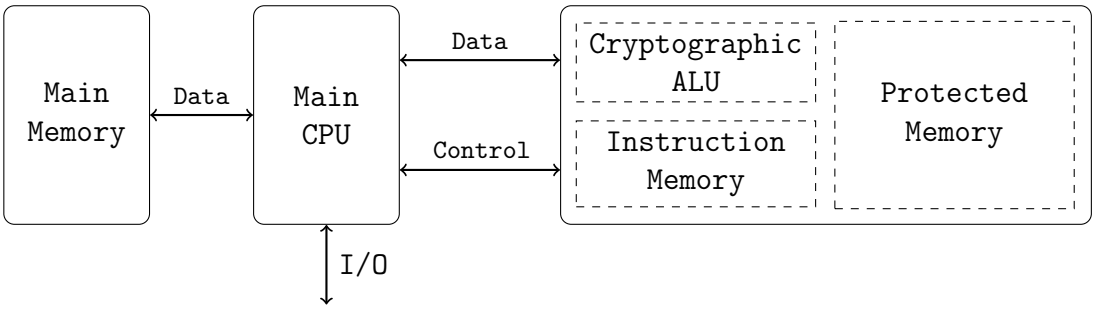


Figure 3.1: Generic cryptographic coprocessor architecture

Being the oldest and most widely available solution, the implementation of cryptography functions on *General Purpose Processors (GPPs)* has been the primary choice for many applications. However, despite their low cost and flexible nature, GPPs are not built with cryptography in mind and most cryptographic operations have to be performed in software. This leads to two major disadvantages: low throughput and reduced security. Not only do the ALUs of these processors need more time to execute encryption and decryption tasks but they also rely on unprotected data buses and memory. This means that adversaries could easily either intercept data and keys in transit or simply read out sensitive information from memory. Recent GPP models provide better support for cryptography operations but the added design costs are often not justified in the case of embedded systems.

A different approach is provided with *Application-Specific Integrated Circuits (ASICs)* which are designed with the specific purpose of implementing one or more cryptographic primitives in hardware. ASICs provide a great improvement in throughput, but also present the drawback of inflexible operation. This inflexibility leads to increased research and development costs. Thus, ASICs are only suitable when large numbers of devices are produced. In addition, future modifications to the implemented primitives have a high cost, making it unlikely for manufacturers to be willing to keep their products up to date.

Finally, thanks to their flexibility, *Field-Programmable Gate Arrays (FPGAs)* are becoming increasingly popular in a wide array of applications, including cryptographic processors. They combine the best features of both the previous categories, implementing logic in hardware while being reconfigurable. Unfortunately, FPGAs were meant to be used mainly as prototyping tools and as such, they are not optimised for use in commercial environments. However, recent developments in the field have greatly improved the situation and FPGAs are being included in more and more commercial products.

3.2.3 Security Modules

Several architectures have been proposed, geared towards protecting against the attacks discussed above, mainly through the implementation of a cryptographic processor. The

Metric	GPP	ASIC	FPGA
Throughput	Low	High	Low
Power Consumption	Depends on software	Low	Very High
Design Effort	Medium	High	Medium
Time to Market	Low	High	Low
Size	Small - Medium	Large	Small
Flexibility	Very High	Very Low	High
Density	High	Very Low	High
Testing Complexity	Low	High	Low

Table 3.1: Cryptoprocessor architecture comparison [32]

basic structure of all of the following methods is similar to the one shown in Fig. 3.1, with differences stemming from the targeted application domain. In the following, we only consider solutions that are, fully or in part, based on hardware. Alternative methods provide isolation on the software architecture level, with the most prominent ones including *Trusted Execution Environment*[33] and *ARM Trustzone*[34].

Hardware Security Modules

The most advanced solution are arguably *Hardware Security Modules (HSMs)*. These modules are typically used in server systems and are thus mostly free of size and cost restrictions. They are essentially complete systems, including a microprocessor, one or more cryptographic ICs, and secure storage. The complete architecture is enclosed in a casing which is hardened against invasive attacks, and an I/O interface is used to communicate with the main CPU. HSMs are mainly used in security or safety critical applications like banking or automotive systems where compliance reasons motivate manufacturers to bear their relatively high cost[35]–[37].

Secure Elements

Due to their form factor,*secure elements* are often included in embedded systems and smart-cards, used for identification purposes in a range of applications including banking, ticketing, access control, and identity documents[38], [39]. Due to their size, secure elements are implemented in a single IC which is tamper resistant and is often produced in large quantities, accentuating the importance of associated cost constraints. As a result, the functionality of the elements is highly restricted and, in some cases, the IC relies on external power from a different device.

External power is common practice in smart cards. In these cases, the secure element is only active when the card is connected to a reader. However, active protection methods are not available without power and thus such cards often rely on passive physical protection. This is an additional area where PUF circuits can replace secure secret storage, since the

PUF secrets are only available while the device is powered up.

In the realm of devices with increased resources, for example in smartphones and other 'smart' devices, similar components are utilised under the term *secure enclave*[40]. These enclaves serve the same purpose as the secure elements, but are normally part of a larger IC, and are capable of more complex operations since they can often make use of host device's features. Manufacturing cost is still a concern but much less so, due to the increased value of the host devices.

Trusted Platform Modules

Trusted Platform Modules (TPMs) are designed to be included in consumer systems and thus their cost and capabilities lie in the middle ground between secure elements and HSMs. Initially, TPMs were part of higher end, business oriented models but in recent years they made their way into the majority of computers. This has lead to wide support of TPMs from operating systems, making them readily available to applications with minimal effort compared to the solutions discussed above[41], [42].

While TPMs support a variety of cryptographic operations, their main focus is on generating a root of trust for the system. Thus, they are mainly concerned with the attestation of the system state and software, usually during boot time, and the secure storage of small amounts of sensitive data[8].

The TPM standard developed by Trusted Computing Group[43] allows for different types of TPMs, some of which are implemented in software and thus provide no physical tamper resistance[8]. In any case, much of the functionality of TPMs is carried out in software and thus their security is subject to common software shortcomings.

3.2.4 Disorder-based Security

The architectures discussed above are assisted by well-established advantages of hardware, but they do not directly draw their security from physical primitives. Fairly recently, a new paradigm of security, which can be summarised under the term *'disorder-based security'*, has been the focus of many research efforts[44].

As suggested by their name, disorder-based security methods involve extracting secret identifying information from physical structures and utilising this information in security protocols. Due to the physical aspect of the secrets, they are considered unique for a specified instance of the hardware, and are often hard to duplicate, providing a level of unclonability.

The first examples of methods employing physical disorder were based on analogue primitives such as optical reflection patterns[5] and paper microfeatures[45]. Digital primitives which would eventually become known as Physical Unclonable Functions (PUFs) were developed shortly after, and gained much greater popularity than their analogue counterparts due to their ease of integration in existing digital systems.

Regardless of its source, physical disorder is not a direct replacement for the solutions discussed in previous sections but rather complements their functionality. Besides the un-

clonability advantages that they provide, disorder-based hardware components are often easily affected by variations of their environment, for example changes in ambient temperature or power supply characteristics. This feature can be used for attack detection via detecting unexpected environmental conditions. Additionally, these components are inherently sensitive to invasive probing, which has a high probability of disrupting the disorder source. Thus, the barrier for physical attacks is raised, with some attacks being likely to destroy their target secrets before managing to extract them.

3.3 Cryptography

Cryptographic algorithms and other cryptographic primitives are essential building blocks which provide the formal guarantees underpinning the security of the protocols. Most proposals, including our work, operate on the assumption that these basic blocks are robust against cryptanalysis and other attacks. In this section, we discuss some basic notions of cryptography focusing on the aspects that are valuable in the context of our work. For further details on the mathematical foundations of these notions we refer to the wealth of available cryptography texts[11], [24], [46], [47].

3.3.1 Asymmetric Cryptography

The complexities of key managements in symmetric cryptography as they were discussed above sparked the creation of an alternative paradigm by Helllman and Diffie[48], where each party holds a pair of keys. Thus, each party has a *private* and a *public* key, and is able to safely share the latter without revealing anything about the former. Public keys can then be used to encrypt data which can only be decrypted with the associated private key, effectively allowing anyone to securely communicate with a recipient of their choice, without prior agreement, as long as they possess the relevant public key. The reverse process is used for *digital signatures*: a message can be signed with a private key and the signature can be verified with the corresponding public key. The nature of asymmetric cryptography ensures that only the holder of the private key can generate valid signatures for any given message.

Unfortunately, asymmetric cryptography algorithms have a higher overhead compared to their symmetric counterparts both in terms of computation and in terms of key length. The main alternatives for practical implementation of the above primitives are the *Rivest Shamir Adleman (RSA)* cryptosystem and methods based on *Elliptic Curve Cryptography (ECC)*. RSA was the industry standard for decades, with its main advantage being the support of encryption and signature with the same building blocks. ECC-based algorithms are capable of the same operations but require additional complexity, for example in the case of encryption. However, ECC alleviates some of the added cost of asymmetric cryptography due to its use of considerably shorter keys and signatures[47].

In our work, we only consider ECC-based methods for the implementation of the proposed protocols. This is because of the scalability benefits of shorter data blocks,

especially in IoT scenarios with a large number of nodes. In addition, ECC can be used in a variety of ways, including encryption, signature, and zero knowledge proofs. All of these primitives can be built on the same algebraic foundations of curve points and operations between them. Therefore, a relatively simple ECC arithmetic block can be created where the necessary operations are performed securely and efficiently, as discussed in Chapter 9.

Zero Knowledge Proofs

Zero Knowledge Proof (ZKP) protocols are an especially useful application of asymmetric cryptography. Also, called 'zero knowledge proof of knowledge', these methods allow for an entity to unquestionably prove its possession of a specific piece of information without in fact revealing the information itself. In the context of digital data, ZKPs make use of public keys: the prover initially publishes her public key. Then, the prover generates a *commitment* value which is derived from the combination of her private key and the secret value, the possession of which she needs to prove. This commitment can be safely shared with the verifier since it does not reveal anything about the secret value. When verification is required, the verifier sends a *random challenge* to the prover, who replies with a *proof* based on the secret value and the received challenge. The verifier is then able to establish the validity of the proof using the previously shared commitment, the received proof, and the public key of the prover.

Most practical ZKP applications use secret values of considerable length and thus there is a very low probability that a malicious prover can successfully guess the secret value (since no indication of the secret value is is available). However, this probability is lowered even further by repeating the proof process. Due to the randomised challenge, even a small number of repetitions can significantly lower the guessing probability.

3.3.2 Cryptographic Hash Functions

Hash functions are functions which accept inputs of arbitrary size and produce a fixed-sized output. In a general computing context the term 'hash function' might refer to similar constructions that are used in the creation of hash tables. However, in the context of our work we use the term 'hash functions' to refer to *cryptographic* hash functions. These functions have the following properties[24]:

Preimage resistance For a given output h, describes the difficulty of reversing the hash function to find an input m such that $h = hash(m)$. This is commonly referred to as the one-way property.

Second preimage resistance For a given input m_1, describes the difficulty of finding a second input $m_2 \neq m_1$ such that $hash(m_2) = hash(m_1)$.

Collision resistance Without any input constraints, describes the difficulty of finding two inputs m_1 and m_2 with $m_1 \neq m_2$ such that $hash(m_2) = hash(m_1)$.

In an applied context, cryptographic hash functions have additional useful properties,

some of which are based on their mathematical foundations:

- Given the output of a hash function, it is impossible to derive the input in polynomial time. Essentially, adversaries are limited to the brute force method of trying all possible input combinations (with certain significant improvements in some cases). Evidently, the size of the input also greatly affects this search space.
- A hash function output should not provide any information about the corresponding input. This property is achieved by uniformly mapping the inputs to the outputs.
- Due to their uniform mapping, hash functions present the so-called 'avalanche effect' where small changes in the input result in a significantly different output.
- The length of the output is constant and does not depend on the length of the input. For this reason, hash functions are often used as one-way compression functions. This compression property allows hash functions to be used as entropy accumulators, condensing their inputs without affecting their entropy.
- Modern hash functions are designed to be efficient to compute and implement.

Definition 3.2. *A hash function with arbitrary input length and output length m is defined as a deterministic, one-way mapping:*

$$HASH : \{0,1\}^* \rightarrow \{0,1\}^m$$

3.3.3 Random Number Generation

The ability to generate high quality random numbers is of paramount importance for the majority of cryptographic algorithms and security protocols. Random numbers are used as the basis for key generation, and as 'nonces' which provide freshness in protocols, preventing a number of attacks. Generating unpredictable random numbers is a difficult task, especially in a distributed setting like IoT: the resulting numbers need to not only be unique for a specific device but also across all similar devices.

Randomness is often measured with the notion of *entropy*[24]. Entropy expresses the uncertainty regarding the outcome of a random variable, with entropy content said to be 'higher' when each outcome is equally likely (uniform distribution of outcomes). Unfortunately, due this stochastic nature, deriving exact entropy results is not possible and randomness evaluation methods are mainly confined to statistical analysis. Such analysis is usually aimed at proving the 'non-randomness' of a specified stream of data[49].

As we will see in Part III, the chaotic behaviour of PUF responses makes PUFs a convenient building block for random number generators with high entropy.

4. Physical Unclonable Functions

4.1 Introduction

The most prominent practical embodiment of unclonability are *Physical Unclonable Functions (PUFs)*. Initially proposed in an analogue form by Pappu[5], PUFs quickly became a prominent fully digital building block for security protocols. They make use of random variations in the fabrication process of hardware components, to generate unique secrets based on a challenge and the physical behaviour of the underlying hardware. Thus, PUFs are an efficient way to harness the disorder present in many common digital elements and represent it in a binary form that can be used in conventional and novel security methods. It is considered impossible to predictably control this disorder and therefore, given a PUF, it is not possible to produce a second one which is physically identical.

While many types of PUFs exist, electronic PUFs have been the focus of most research efforts, due to their efficiency and low cost. Thus in our work, we only consider PUFs which have digital inputs and outputs making it possible to integrate them in algorithms and methods designed for binary operands. At the same time, this PUF category can be constructed out of components that are often encountered in digital systems, as discussed in Section 4.2.

4.1.1 Definition

A formal definition of the behaviour and operation of PUFs is not an easy task due to the variety of constructions as well as the physical aspect. Since we are mainly interested in the properties of PUFs, we use the succinct definitions introduced below, where the influence of physical variations is implied. For extended formalised definitions we refer the reader to [10].

Definition 4.1. *PUFs can be described as a mapping from the set of challenges C to the set of responses R:*

$$PUF : C \rightarrow R : PUF(c) = r, c \in C, r \in R$$

The resulting pairs are referred to as Challenge-Response Pairs (CRPs) *and can be expressed as:*

$$(c_i, r_i) = (c_i, PUF(c_i))$$

Unless specified otherwise, for the rest of this thesis we use the following simplified definition, representing the PUF as a mathematical function with a single variable.

Definition 4.2. *For a challenge $c = \{0,1\}^n$ and a response $r = \{0,1\}^m$, a PUF instan-*

tiation is defined as:

$$r \leftarrow PUF(c)$$

The lengths n, m of the challenge and the response respectively, are constant for the specified instantiation.

4.1.2 Properties

In order for PUFs to be utilised in practical applications while retaining their valuable behaviour, the following properties are required[10], [50].

- *Evaluable*: Given a PUF instantiation and a challenge c, there exists an efficient method to generate the response $r = PUF(c)$
- *Unclonable*: Given a PUF instantiation, it is hard to construct a procedure $\gamma : \forall c \in C, \gamma(c) \approx PUF(c)$ up to a small error.
- *Unpredictable*: Given a set Q of CRPs: $Q = \{c_i, r_i\}$ of a specified PUF instantiation, it is hard to compute $r_u = t(c_u), c_u \notin Q$, without evaluating the PUF.
- *Unique*: $\forall c \in C, PUF(c)$ is uniquely mapped to the physical entity in a way that no other entity, however similar, will present the same CRP even up to a small error.
- *One way*: Given a PUF instantiation and a response r, it is hard to find c *suchthat* $r = PUF(c)$
- *Tamper evident*: Invasive intrusion attempts to the physical construction will transform PUF to PUF' *suchthat* $\forall c \in C : PUF(c) \neq PUF'(c)$, with a high probability.
- *Reproducible*: Given a PUF instantiation and a challenge c, the response $r = PUF(c)$ is reliably the same across any number of PUF evaluations, up to a small, bounded error.

Most prominent PUF constructions are able to provide these properties, albeit to varying degrees. Stemming from these properties, PUFs can provide a number of novel features, when used as building blocks in security protocols:

- PUFs can serve as an *unclonable root of trust* the properties of which can then be propagated throughout the system. As a result, the random manufacturing variations are transformed into digital secrets and eventually into protocol-level properties.
- Due to their unpredictability, PUFs can supply the necessary entropy for generating high quality random numbers or cryptographic keys.
- The instability of responses is limited to a correctable degree and, for the majority of PUF classes, the regeneration process is very efficient. Thus, there is no need for secure storage of secrets since they can be regenerated when needed.

- Through the use of appropriate methods, it is possible for protocol secrets to be generated automatically without any intervention from users or administrators. Therefore, the secrets are not exposed to human operators and trust relationships can be minimised, leading to a reduced attack surface.

4.1.3 Quality Metrics

The variety of PUF constructions also creates a need for metrics of their quality that quantify their unique features while being independent of implementation details. The main focus of PUF evaluation methods is on two distance metrics:

- The *intra-distance* which is the distance between two responses of the same PUF instance to the same challenge, representing the stability of the PUF. In some cases, intra-distance is normalised over the response length, giving the Bit Error Rate (BER).

- The *inter-distance* which is the distance between two responses of different PUF instances to the same challenge, representing the uniqueness of the PUF.

Another interesting measure of PUF quality is the entropy content of the responses. As discussed in Chapter 9, the measurement or even the approximation of entropy is not trivial. The *min-entropy*[51] and the *bit bias*[10] are often derived from experimental observations. Researchers have also proposed the derivation of maximum entropy bounds based on theoretical reasoning about adversary capabilities[10].

The above metrics are discussed in detail in Section 8.3 where additional metrics are also defined in the context of SRAM PUFs. Table 4.1 and Section 8.5 provide an overview of the metrics in practice.

Applications

Due to their qualities, PUFs can be used as a building block in a variety of security scenarios and protocols. A number of PUF-based variants of cryptographic protocols have been proposed, including key exchange[5], bit commitment[52], and oblivious transfer[5], [14], [53].

The majority of existing work focuses on identification or authentication protocols[5], [54], [55]. In such applications, one or more CRPs are produced and handed over to the verifier. The verifier can subsequently identify or authenticate the prover through querying for PUF responses and comparing to pre-shared ones. A number of issues make this simplified protocol impractical, including the requirement of large numbers of CRPs, and thus numerous proposals have been developed to alleviate some of the issues[16], [56].

Another major PUF application area is that of entropy generation. PUFs have been employed in generating high quality random numbers which can be used as freshness nonces in security protocols, or as seeds for the generation of further randomness or cryptographic keys[57], [58].

The unpredictability property of PUFs is also valuable in securing local storage, whether it is volatile or permanent. Conventional symmetric encryption with PUF-derived keys is used to encrypt data on-the-fly before it is stored, providing at-rest protection[58]. Protection methods for runtime data such as cache contents have also been proposed[59]. In some cases, PUFs can replace the traditional encryption algorithms, leading to a reduced computation overhead.

In the commercial world, PUFs are still having limited adoption, arguably due to the difficulty of clarifying their advantages for a larger audience. To the best of our knowledge, IntrinsicID is the only organisation where the main aim is to utilise PUFs. IntrinsicID provides several variants of PUF IP, based on SRAM PUFs and used for key generation and management[60]–[62]. Similar applications are also targeted by eMemory[63] and Microsemi[64], utilising PUFs as key generators for conventional symmetric and asymmetric cryptographic schemes. Maxim Integrated[65], [66] focuses instead on the production of PUF-enabled microcontrollers providing authentication of a variety of hardware ranging from batteries and cables, to medical devices.

4.2 Classification

In the recent years a number of efficient and highly secure PUF constructions have been proposed with the most notable being SRAMs[67], Arbiters[68], and Ring Oscillators[69]. Most electronic PUFs can be adapted to provide the I/O behaviour required by the application, mainly through concatenation of multiple CRPs.

The methods proposed in this work, do not depend on a specific PUF construction but rather on high level properties as they are discussed in Section 4.1.2. Thus, we do not describe the implementation specifics of every physical construction. However, a quick comparison of popular constructions can be made based on Table 4.1.

PUF Construction	**Intra-distance**	**Inter-distance**	**Maximum Entropy**
SRAM[70]	7.78%	48.72%	94.09%
Latch[71]	26.33%	30.77%	71.92%
Arbiter (basic)[68]	7.75%	46.43%	>30.00%
Arbiter (2-XOR)[72]	14.25%	49.71%	>78.00%
Ring Oscillator[54]	3.88%	49.54%	94.69%

Table 4.1: Quality metrics for major PUF constructions[10]

4.2.1 Intrinsic and Non-Intrinsic

One classification axis is with regards to the evaluation mechanism, categorising constructions between *intrinsic* and *non-intrinsic*. We adopt the intrinsic PUF definition by Maes[10]: an intrinsic PUF is a PUF where (a) the evaluations are performed internally

by embedded measurement equipment, and (b) the underlying random physical processes are implicitly introduced during manufacturing.

The majority of electronic PUF constructions are considered intrinsic under the above definition, which gives them additional advantages in the context of security protocols. Firstly, the implicitly introduced physical processes incur no additional production costs and leave less margin for interference from external actors. Additionally, due to their internal evaluation mechanism, intrinsic PUFs can be integrated into existing cryptographic schemes with minimal modifications to the scheme logic. Due to these advantages, very few non-intrinsic PUFs were substantially introduced, mainly while the PUF concept was still in its infancy.

4.2.2 Strong and Weak

On the protocol level, the most important categorisation of PUFs is between *strong* and *weak* constructions. Initially, strong PUFs were defined as (a) having a very large challenge space, making it impossible to create a full list of CRPs even with prolonged access, and (b) generating uncorrelated responses, making it impossible to predict future responses based on previous ones[70]. Evidently, both properties are exceedingly important in security applications. The first property enables the prevention of replay attacks in authentication protocols via the use of fresh CRPs in every interaction. The second property implies unconditional modelling resistance, which is effectively the foundation of unclonability. Constructions that are not considered strong PUFs are instead categorised as 'weak', although the terminology might be considered misleading, in regards to the actual unclonability properties.

Various protocols were developed on the basis of strong PUFs[73] and the primitive was also formally evaluated by Rührmair et al. in [74]. However, intrinsic PUFs initially considered *strong*, were eventually proven to be vulnerable to modelling attacks[75]–[77]. Thus, the unpredictability property was largely abandoned and additional methods of providing modelling resistance are actively being researched[15], [78], [79]. The development of truly *strong* PUFs remains an open problem.

4.2.3 Disorder Source

A further categorisation of the available PUF constructions can be made based on the source of their physical disorder. No clear universal advantages exist for any of the classes discussed below. For example, some of the constructions exhibit a higher entropy content, and others are significantly more robust against environmental influences. The choice of a specific class is based on the application requirements with consideration including the required security guarantees, the required number of CRPs, the operation conditions of the system and potential existing circuitry.

The number of alternative PUF constructions has grown exponentially in the recent years, often targeted at specific applications or hardware. Therefore, this section does not provide an exhaustive list of constructions but rather an overview of the most commonly

used solutions across multiple domains. Detailed discussions of the following and other PUF constructions can be found in [10] and [80].

Bistable Memory Elements

PUFs based on memory elements exploit process variations which lead to size mismatches in the underlying transistors and thus stochastically asymmetric behaviour.

The most thoroughly researched construction of this category, and arguably one of the most commercially successful ones is the *SRAM PUF*[70]. Its operation is based on the size mismatch of the two inverters which comprise the 6T SRAM cell (shown in Fig. 8.1a). Due to this mismatch, SRAM cells are likely to have a preferred power-up state which is defined by the process variations. The challenge is applied as one or more memory addresses and the response comprises the concatenated power-up values of the specified cells. A small number of cells have no preferred state, making them highly sensitive to noise and leading to a degree of instability in the PUF responses. No modifications are required for an SRAM IC to be used as a PUF, therefore SRAM PUFs have a low implementation cost. Additionally, there is no strong correlation between responses, hindering modelling attempts. Unfortunately, SRAM PUFs are capable of producing a limited number of CRPs and are thus not suitable for conventional authentication protocols. We extensively discuss the operation and properties of SRAM PUFs in Chapter 8.

A number of constructions making use of the same primitive were also proposed, mainly aiming to rectify some of the disadvantages of SRAM PUFs. For example, *Latch PUFs*[71] allow repeated access to their responses without the need for power-up cycling and *Butterfly PUFs*[81] are designed specifically for FPGAs where traditional SRAM architectures are unable to operate due to automated reset.

Delay Paths

The second major category of disorder is based on the random mismatch between digital delay paths that are designed to be identical. PUFs of this category were among the first electronic PUFs to be proposed and include constructions based on arbiters, ring-oscillators, or self-times rings.

Arbiter PUFs[68] take advantage of race conditions in pairs of identically designed paths of inverters. The PUF challenge is applied to the inverters, configuring the delay paths. Due to manufacturing variations, the two paths exhibit a random difference in their propagation time, and an arbiter is used to resolve the race condition, generating 0 or 1 depending on the fastest path. There is a significant probability that both paths will have nearly identical delay, leading the arbiter to temporary metastability. This metastable state is resolved almost immediately, with the outcome being heavily influenced by external conditions like ambient temperature. Thus, the metastability of the arbiter is the root cause for the instability of Arbiter PUF responses. Arbiter PUFs can be easily modelled and thus considerable literature has been focused on both hardening the original construction[72] and evaluating the limits of such methods[75].

The second delay-based PUF comprises a similar construction, with an additional negative feedback path,creating a *Ring Oscillator PUF*. The challenge is applied in the same manner and a counter is used to derive the frequency of the oscillator. This frequency depends on the process variations and thus constitutes the PUF response. The first Ring Oscillator PUF was proposed by Gassend et al. [54].

Mixed Signal

The third class of PUF constructions includes architectures where the disorder is provided by analogue phenomena. As a result, besides the PUF block, measurement circuitry is required for the PUF responses to be represented in digital form. For this reason, mixed signal PUFs have not seen the same adoption rate in industry as other PUF classes.

Current flow is the most common quantity involved in mixed signal PUFs. In a proposal by Lofstrom et al.[82] addressable arrays of MOSFETs are used. When a particular transistor is addressed, a comparator is used to convert its voltage to a digital response. A similar method was used in [83] and improved in [84]. In these works, the output voltage of two MOSFET arrays is compared and the response is determined by the comparison outcome. In the above solutions, process variations cause a mismatch in the threshold voltage of the transistors, providing a PUF behaviour.

DRAM ICs have increasingly been the focus of recent PUF research, due to their ubiquity and low cost. In contrast with SRAMs, in DRAMs the leakage current of the transistors causes a decay of the stored bits. The decay rate is affected by process variations and thus Keller et al. proposed the generation of PUF responses based on measurements of this rate[85]. Various improvements of the original method were later proposed, removing the need for modifications to the DRAM logic[86], [87].

4.2.4 Extended Functionality

Certain disadvantages of the major PUF constructions were apparent soon after these solutions were proposed. Several extensions followed, aiming to alleviate specific issues. Eventually, some of the extended functionality was integrated into most PUF implementations and is now considered part of standard features.

Controlled PUFs

Controlled PUFs[88] combine the PUF block with additional logic that is considered practically inseparable. The added functionality includes error correction and hardening against modelling attacks by applying cryptographic hash functions to the inputs and outputs of the controlled PUF. Additionally, as seen in Fig. 4.1 identification information is included in the output, to provide identity obfuscation in privacy-focused applications. Similar features are included in the cryptocore of Chapter 7.

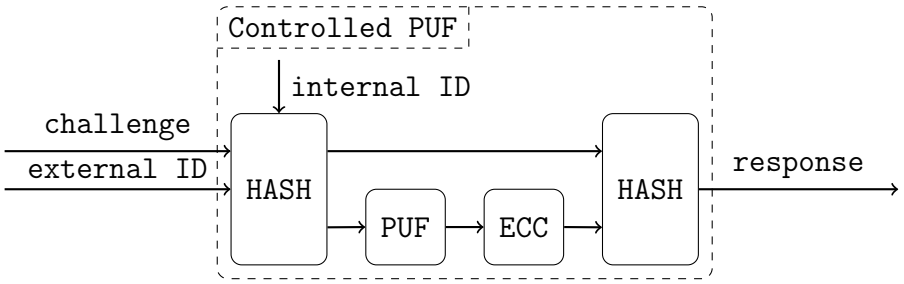


Figure 4.1: Controlled PUF[88]

Reconfigurable PUFs

Reconfigurable PUFs remove the need for an extensive response space by providing mechanisms to update the internal state of the PUF. This transformation is one-way and does not depend on previous state, effectively refreshing the CRPs. Thus, the most important properties of strong PUFs can be achieved, with the additional benefit of allowing the revocation of secrets. The reconfiguration can be either physical, permanently changing the hardware variations[89] or logical, applied to the input and outputs before they are processed by the PUF[90]. A generic block diagram of the latter is shown in Fig. 4.2.

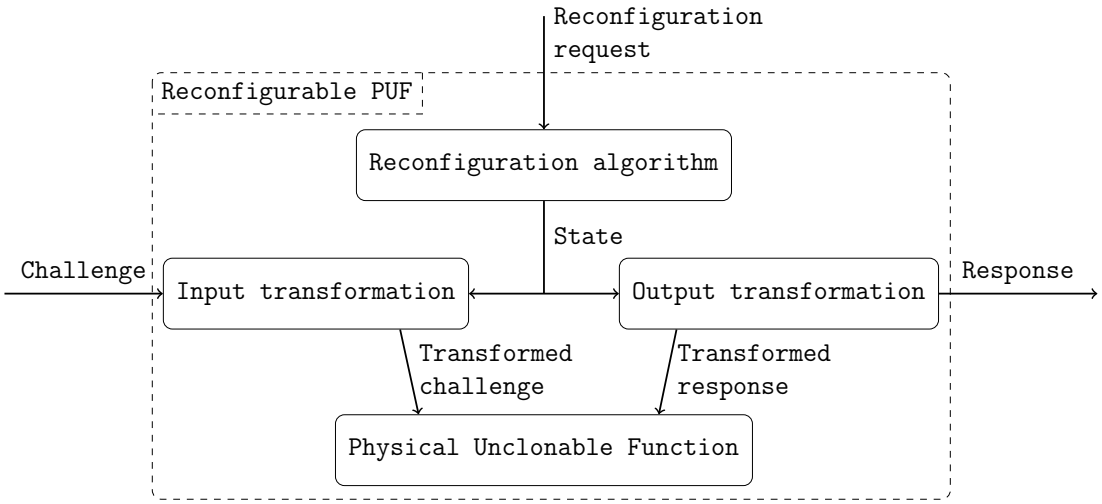


Figure 4.2: Generic block diagram of a logically reconfigurable PUF[90]

Rührmair et al.[91] introduced *Erasable PUFs*. Besides having a very large response space, these constructions allow the erasure of single CRPs. This is a valuable property in protocols where secrets are derived directly from the PUF responses, as it defends against attackers with continued access to exchanged CRPs[55]. Unfortunately, the physical complexity of PUFs that provides their unpredictability is also a hurdle when it comes to realising erasable CRPs. Due to the absence of granular control over the physical processes, logical erasure is achieved via reconfiguration[92].

Public PUFs

Public PUFs are an interesting class of PUFs that aims to provide completely new paradigms in security protocols. Initially described as 'SIMulation possible but Laborious' (SIMPL)[93] systems, they evolved into various implementations that have the common

goal of eliminating the need for secrecy of the PUF responses and internal state[94]. The method of realising this goal might seem counter-intuitive in security scenarios: the responses are no longer kept secret and can be legitimately produced by more than one parties. This is mainly achieved in two ways. Firstly, SIMPL systems introduce a timeout notion in the security protocols by using electronic constructions that can generate a response to any given challenge in bounded time. At the same time, mathematical models of these constructions are made public but their evaluation is much slower than that of the original PUF. As a result, the verifier is able to differentiate between the IC and the model merely by measuring the time between submitting a challenge and receiving a response. The second method, called *matched PUFs*, is based on creating two distinct ICs that will have the same behaviour up to an acceptable error[95]. The creation of a third matching IC is considered statistically impossible. Matched PUFs have been realised in practice using fabrication and ageing methods. On the contrary, the requirement for guaranteed time bounds in SIMPL PUFs is hindering practical implementations of the primitive.

Commutative and Invertible PUFs

A PUF operation that is both commutative and invertible would allow for the complete replacement of symmetric key encryption by PUFs. Proposed in [96], the so-called *Barrel Shifter* PUFs can be applied in arbitrary order and also provide the inverse operation, transforming responses into challenges. Thus, these PUFs can be employed in sigma protocols, binding the cryptographic processes to specific hardware instances.

4.3 Models

To facilitate the discussion of PUF properties and applications, we define two abstraction models, outlining the various assumptions about the behaviour of the PUFs. The entirety of these assumptions can be achieved in practice, albeit with varying levels of efficiency and cost. Both models include the majority of electronic PUFs, and are aimed at the functional aspects of the PUFs.

4.3.1 Block Level

The simplest abstraction level models the PUF as a self-contained block, providing the properties of Section 4.1.2. In summary:

- When provided with a challenge, the PUF block returns a response, which is a function of both the challenge and the physical characteristics of the PUF. (Unique and Evaluable in Section 4.1.2.)

- The lengths of both challenges and responses can be freely chosen by the application, as long as they remain the same for a particular instantiation. In practice, such a feature cannot be taken for granted but can be provided via simple methods such as the one proposed in Chapter 9.

- When provided with the same challenge, the PUF block returns the same response, with an error probability ε. We call this probability Bit Error Rate (BER). (Reproducible in Section 4.1.2.)

- Any correlation between individual PUF responses, or other attributes of the underlying PUF construction, are propagated to the output unchanged.

- Adversaries do not have access to the structure of the block and thus cannot directly access its internal state. Invasive attempts have a high probability of destroying the PUF secrets.

The block level model provides a universal I/O interface for any electronic PUF, regardless of the peculiarities stemming from the specific hardware construction.

4.3.2 Component Level

In the second model, many of the intricacies of the PUF block are disguised by additional logic. Thus, the PUF is seen as a component, based on the block of Section 4.3.1 and with additional properties that resemble those of ideal PUFs:

- Bit errors are corrected with an appropriate method, leading to fully stable responses[97], [98].

- For PUFs with correlated responses, countermeasures against modelling are also included. A simple method is the use of a cryptographic hash function to obscure the mapping of challenges to responses, by increasing the uniformity of the generated responses[10], [97].

- Adversaries do not have access to raw PUF responses or other internal state of the model. Error correction helper data is considered public.

Gassend et al. described a similar model in [99], including however a general purpose processing element which allowed the execution of arbitrary code. In our case, only the logic that is absolutely necessary (for error correction and entropy enhancement) is included within the model.

A note on error correction: Due to its complexity and overhead, the process of reducing or removing bit errors in PUF outputs is a major issue, especially for practical implementations. Throughout Part II, we regard the error correction process as an implicit step involved in the generation of CRP responses. In reality, two phases are involved in the creation of a stable PUF response: *enrolment* and *reproduction*. The enrolment phase occurs once for every distinct challenge applied to the PUF. An error correction code is used on the 'raw' PUF output, producing a block of helper data which is stored locally along with the supplied challenge. In the reproduction phase, occurring on every subsequent application of the same challenge, the helper data is employed by the error

correction algorithm to transform the new raw PUF output into a consistent PUF response. In many applications, the enrolment phase takes place offline and often for the entirety of the PUF CRPs. On the contrary, in our work, the enrolment executed on-demand every time a new, unseen challenge is applied to the PUF. As a result, we avoid the characterisation of the whole PUF, reducing both the exposure of its internal secrets and the storage costs involved.

4.4 Adoption Challenges

In theory, PUFs and their associated protocols appear to be the holy grail of information security. Despite their shortcomings, PUFs are an exceedingly promising primitive and the majority of the problems are likely to be mitigated in future research efforts. However, care should be taken when adding constructions external to the PUF, since some of them negate PUF benefits and especially their low manufacturing cost. In this section, we highlight a number of issues that hinder the adoption of PUFs in every application domain.

Research Focus and Assumptions

It is often assumed that PUFs constitute a replacement for expensive cryptographic operations, inherently improving the security of any system. However, that is rarely the case and additional mechanisms are required to integrate PUFs in any security application. In most cases, the advantages are qualitative and cannot be directly measured with conventional methods. Thus, it is difficult for researchers to evaluate PUF-enabled solutions and demonstrate their practical value.

The focus on advantages that are easy to measure is also evident in the body of literature regarding PUF protocols. In the majority of publications, the highly complex behaviour of PUFs is reduced to a set of binary bit vectors which are subsequently used in the place of traditional cryptographic keys. While the value of such proposals is unquestionable, it does not capture the full potential of PUFs. For example, instead of simply masking the instability of responses, it would be possible to harness it as a means of detecting modelling attempts or other abnormal PUF behaviour.

Threats

The Achilles' heel of PUFs is their vulnerability to a variety of mathematical modelling attacks, mainly driven by machine learning algorithms. Thus, there exists a gap between physical and mathematical unclonability, as they were discussed in Chapter 2. A number of researchers have proposed methods to emulate the behaviour of certain classes of PUFs, creating mathematical clones[75], [76]. For many of these methods simply augmenting the complexity of the PUF with additional logic is enough to thwart such attacks but, for others, higher level solutions are needed.

In addition, despite the common assumption of tamper resistance, certain classes of

PUFs are vulnerable to various side channel attacks which circumvent the interface to the PUF block and directly access the internal state[100]. Conventional physical protection methods are required to thwart such attacks.

Finally, the implicit trust in the random manufacturing process assumes the correct operation of any given PUF block. However, the unique characteristics of PUFs are exceptionally hard to validate. Hardware that appears to be a PUF can be replaced with static or more sophisticated logic that will undermine its perceived unclonability benefits[101]. Even when hardware authenticity is attested, the same is not necessarily true for the behaviour of the PUF.

Error Correction

All PUF constructions exhibit a considerable error rate stemming from their inherent instability. Therefore, in most cases, error correction is required, involving a substantial overhead both in terms of complexity and helper data storage. In addition, part of the generated entropy is lost through exposure of the helper data, requiring even more sophisticated error correcting methods. Numerous error correction solutions have been proposed for PUFs, with some of them aiming to reduce the entropy losses[10], [55], [70], [102]. Unfortunately, none of the existing methods provides any major reductions to the aforementioned overhead.

A different class of error correction methods involves the characterisation of individual PUF bits and the selection of the most stable ones for response generation[103], [104]. While these methods greatly reduce the error rate, they are only applicable to certain PUF constructions and are not scalable, since they have to be performed on every PUF instance separately. Additionally, the whole PUF secret has to be revealed during characterisation, effectively placing trust into the entity performing the process.

4.5 Conclusion

It is evident that Physical Unclonable Functions and their underlying theory is an exceptionally rich field which promises to revolutionise security and cryptographic primitives.

In this chapter, we summarised the state of knowledge regarding PUFs. We discussed well-established as well as emerging hardware constructions, and introduced generalised models to encompass those constructions. The models will allow us to reason about PUFs in the context of security protocols, without the constraints of implementation details.

Part II

Methods and Protocols

5. Authority Device Scheme

5.1 Introduction

This chapter details Authority Device Scheme (ADS), a collection of cryptographic protocols based on Physical Unclonable Functions and one or more *authority devices (ADs)*. The scheme includes protocols for the introduction, mutual authentication and clustering for network nodes along with advanced features by combining the novel properties of Physical Unclonable Functions with existing key management methods and public key cryptography.

The ADS enables grouping nodes into clusters or *neighbourhoods*, making them aware of their neighbours and using this awareness as a security enabler. This is achieved through the distribution of identifiers among the nodes, through the authority devices. These devices act as a proxy both among pairs of nodes and between the system and its owner.

Researchers have proposed numerous solutions for securely introducing devices to each other and forming clusters, including the assumption of a secure environment[15] or the use of user input like PIN codes[16]. However, an entity which owns a system should be able to ascertain this *authority*, since it is only via this ownership that the system exists in its current form. In other words, the owning entity (an individual or an organisation) should have complete authority over the hardware but also the information that powers the system.

Nevertheless, it is neither secure, nor convenient for human operators to have access to device secrets such as cryptographic keys, identifiers etc. In our scheme, by representing the authority of the system owner with *authority devices*, the owner retains her authority over the system without being exposed to implementation secrets and specifics. As a result, the user is relieved of the burden of key provisioning and management, reducing the attack surface, and advanced features such as delegation of authority and behaviour attestation are enabled, while higher security is obtained via exploiting the *unclonability* property of PUFs. Additionally, strong guarantees can be achieved regarding the decommissioning of devices, since the secrets are never exposed to third parties, even if they are 'trusted'. As such, a decommission process can simply consist of blacklisting and physically destroying the target device.

The proposed scheme has a dual purpose: to act as an enabler for introducing the unclonability primitive in higher level protocols but also further research around the primitive. We strongly believe that there is high value in new methods of managing the security of network systems via exploiting secret information that inherently occurs in

electronic devices, without manual generation or exposure to the device's environment.

Due to the nature of the scheme, there is no need for a permanent managing authority that would create a single point of failure. In fact, if the application scenario requires it, authority devices can be destroyed after the initial system setup. System operation would then continue normally, albeit without the ability to make topology or identity modifications.

In order to focus on a high-level view of the proposed methods, this chapter does not include a discussion of specific cryptographic algorithms. In the proof-of-concept implementation of Chapter 9, a number of design decisions were made to match common practice at the time of writing. As a result, Chapter 9 can also be seen as a set of guidelines that would be considered secure in the context of our work. However, the ADS is fully flexible as it merely relies on the generic features of unclonability and asymmetric cryptography and can be adapted to the application and/or future cryptography needs.

A preliminary version of this chapter was previously made available as a technical report[105].

5.1.1 Contributions

In summary, the Authority Device Scheme delivers the following:

Improved security: All key pairs are dynamically and automatically generated based on PUFs challenged with randomised inputs. Therefore, no parties outside the cryptographic core have access to any private key material. This leads to the minimisation of trust relationships and thus, increased security since no party can be coerced into revealing device secrets. In addition, even if an adversary obtains an authority device, the information stored on it will only allow her to perform high level operations on the system without compromising the communication of the nodes.

Reduced complexity and improved scalability: Due to the dynamic nature of the key material, the need for user interaction is reduced to simply connecting an authority device to the nodes. Additionally, on-demand key generation removes the need for secure non-volatile storage as the keys are only present while the system is powered on.

Decentralised operation: After their initial introduction, nodes operate without the need for a central authority. Thus, neither the security nor the robustness of the system depend on a single device.

Novel neighbourhood features: The ADS encompasses the first four layers of the *unclonability stack* described in Section 2.4 (provider, core, device, and protocols) and, when combined with the strong security properties of unclonability, enables a range of new scenarios and features, as seen in Chapter 6.

The main contribution of this chapter is the introduction of the *Authority Devices (ADs)*. Despite similar primitives being commercially available in the form of 'physical

keys'[106]–[108], the existing solutions mainly act as storage for traditional cryptographic secrets that are provided by the user. In contrast, Authority Devices have the following benefits:

- The cryptographic secrets are occur 'naturally', without any interaction. Thus, any human operators are not aware of secret values and, as a result, they cannot be coerced or tricked into revealing them.

- Since the secrets are protected against copying by virtue of the unclonable core, atomicity and non-repudiation are ensured. The holder of the device can be certain that no other device exists that is indistinguishable from the original AD. This property also implies that actions involving the AD can be unquestionably attributed to the specific device.

- Combining the above properties, decommissioning an AD and its associated secrets is simply a matter of physically destroying the device. The secrets are only present in a single hardware instance and, by virtue of the underlying PUFs, physical actions on said hardware irrevocably destroy the secrets.

5.2 Preliminaries

5.2.1 Application Scenario

The system comprises a number of networked nodes and at least one authority device which is mobile. The architecture of all the devices, nodes and ADs, is a variant of the reference architecture described in Section 2.4.3 but, to match common Internet-of-Things scenarios, the nodes are assumed to be more resource constrained than the ADs.

In the context of this chapter, we make use of the PUF model of Section 4.3.2 and thus treat the PUF as a fully stable block, similar to a cryptographic hash function with unclonable state.

As per the adversary model discussed below, any data that is stored in non-volatile memory during the operation of the scheme is considered accessible to potential adversaries and should thus be appropriately protected. When, in the remainder of this chapter, we refer to storing and retrieving data, we implicitly assume that the integrity of these data is verified. This assumption is based on existing solutions in literature, for example by Hoffman et al.[59].

In the basic application scenario for the scheme, a party which we call *the owner*, purchases a number of networked devices (*nodes*). The owner needs to deploy the nodes in the field, creating a network of devices. As in many practical applications, we assume that the nodes are manufactured by an honest entity and then come under the control of the owner who has full authority on them and performs the initial setup. However, for practical or security reasons, the owner might want to delegate the duty of enrolment (performed in the field) to an external party that is again partially trusted. This third

party is tasked with enrolling the nodes and given an authority device to make this possible.

The authority device is designed to be connected to each of the nodes. The connection can be wired or wireless but a physical proximity between nodes and ADs is required while they are taking part in the proposed protocols, to ensure the physical validation of the nodes. Upon connection, the AD performs the necessary operations to enable the nodes to act as a group and effectively join the same *ownership domain* or *neighbourhood*. It should be highlighted that none of the configuration requires special expertise from any of the human operators.

Multiple ownership is achieved with distinct authority devices. Using a large number of ADs does not affect the scalability of the scheme as those devices only take part in operations that are performed offline and not during the normal operation of the system. Also, as evidenced in the proof-of-concept implementation (discussed in Chapter 9), the size of the information that devices have to retain throughout the protocol is relatively small in comparison to modern device capabilities. In any case, most practical scenarios would require the use of a limited number of authority devices. In the following sections we will refer to a maximum of two authority devices for clarity, although all protocols are designed to support any non-zero number of ADs.

The scheme employs the properties of Physical Unclonable Functions to: (a) securely generate cryptographic keys, without the need of keeping them in storage, and (b) enable the system entities to undeniably prove and verify the identity of their interacting parties.

5.2.2 Notation

To simplify the protocol descriptions, we make use of the standard notation used throughout this thesis, summarised in Table 5.1 for convenience. Operations that are considered well-known or have been described in a different section, are omitted from the descriptions for clarity, unless they are crucial for presentation of the protocol. It should also be noted, that all the operations are executed in the cryptocore and only their results (and any external inputs) cross the boundaries of the core.

5.2.3 Adversary Model

The various protocols of the Authority Device Scheme can be divided into two broad phases: initialisation, and normal operation. It is assumed that the initialisation phase is performed in a secure environment or using secure channels established via other methods. This assumption means that there are no adversaries involved in the protocols of this phase.

On the other hand, the system spends the majority of its lifetime in normal operation where adversaries can be present. Our adversary model is based on the Dolev-Yao model[109] which we expanded to include the physical properties provided by unclonability and PUFs. In summary, the following assumptions are made:

Symbol	Definition
AD_x	Authority Device x
ACK	Acknowledgement
N_x	Node x
P_x	Public key of x
S_x	Private (secret) key of x
$SIG_k(x)$	Signature of x with private key k
$VER_k(x, y)$	Verification of signature y of x with public key k
$PKG(x)$	Derivation of an asymmetric key pair from seed x
$PUF_x(y)$	Evaluation of the PUF of device x with challenge y
$RNG_x()$	Evaluation of the random number generator of device x
$\|$	Concatenation operator

Table 5.1: Summary of symbols

Channel: There is no limitation to the physical medium of communication i.e. wired or wireless. The adversary can eavesdrop on any communication without being detected.

Adversary Capabilities: The adversary is able to observe, intercept, modify, delay, replay and synthesise messages. She is able to guess keys and execute any cryptographic algorithm involved in the protocols. She is however unable to break the cryptographic algorithms used.

It is assumed to be computationally impractical, over the lifetime of the system, for the adversary to exhaustively search the cryptographic key space or the CRP space of any PUF ICs or otherwise accurately generate new CRPs without access to the ICs themselves.

On a physical level, the adversary has the ability to observe the operation of the system and its individual parts either during normal operation or by invasive hardware attacks. In other words, data in device memory and data buses are considered to be available to the adversary unless they are contained within the cryptocore boundary. However, due to the properties of PUFs, we assume that the adversary is unable to probe PUF ICs and extract information (that is not made available through the defined interfaces), without destroying the internal PUF secrets.

Protocol: The adversary acts as a legitimate node and is capable of initiating and taking part in protocols with any of the parties involved in the scheme.

Finally, for the remainder of this chapter, when we refer to 'storing' or 'recalling' data, we assume that the integrity of the data is implicitly ensured. This can be achieved with

PUF-based integrity methods that have been proposed in literature[59].

5.2.4 Use Cases

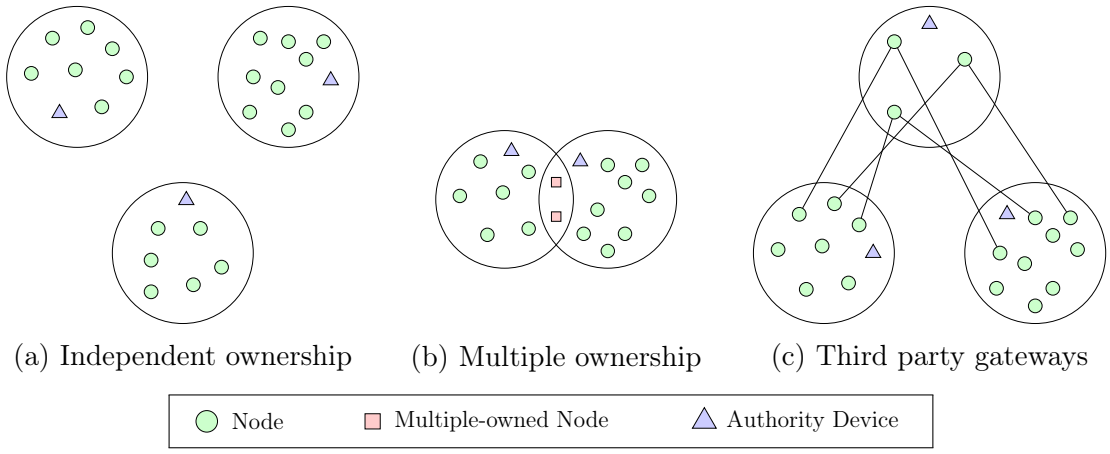
Before proceeding to specifics, it is useful to include a few use cases for the ADS, in order to gain insight into the design decisions presented in the remainder of the chapter. The ADS can be used in a number of topologies such as the ones pictured in Fig. 5.1. In the figure we can discern three main configurations, representing the majority for real world scenarios, with devices under the same authority grouped in circles.

The first case (Fig. 5.1a) is one where nodes under different ownership are divided into separate neighbourhoods with distinct authority devices. For example, this would correspond to a number of discrete company departments.

In the second case of Fig. 5.1b on the other hand, a subset of the nodes belongs to two different groups, and is under the control of two different authority devices, as is the case for the same company employees who belong in multiple teams.

Fig. 5.1c illustrates a hierarchy of nodes where members of one of the groups are tasked with interacting and relaying information between otherwise separate authority domains. The higher level domains interacts directly with just a small number of nodes in the lower level groups, essentially creating two layers of relays. Evidently, this case corresponds to a company with a management hierarchy and individual team leaders.

In this section we present a number of typical use cases for the proposed scheme, in order to clarify the subsequent description of our design choices.



(a) Independent ownership (b) Multiple ownership (c) Third party gateways

Figure 5.1: Example topologies

Military Sensors

Military units often operate in a tree structure where the soldiers (tree leaves) report to higher rank officers (intermediate tree nodes) who recursively report to the next rank, continuing upwards until the commanding officer (root of the tree). Also, soldiers need to be able to perform simple tasks as instructed with the least training possible.

We can envision a situation where a number of sensors need to be deployed over a battlefield to gather and relay sensitive information. The commanding officer, having the

authority over the sensors, prefers to avoid exposure and, as a result, needs to delegate the task of configuring the system to soldiers who are not experts in networking or security. At the same time, it is necessary to ensure that in case a soldier or a sensor is captured, the damage will be contained to the smallest possible organisational unit.

Using an authority device, the commanding officer is able to perform the setup process at the military base, and pass the AD on to his soldiers who will install and configure the sensors in the field. Since the history of the AD is observable, the officer can verify the correctness of the system and be certain that any unexpected behaviour will be detected.

Extending the above to multiple units with separate commanding officers, similarly to Fig. 5.1a, if a soldier bearing the AD for the group is captured, then the officer will be easily able to detect the threat and isolate it, being certain that no sensor secrets were stored on the AD and no other units are affected. Additionally, if a number of sensors are compromised, the remaining ones can be readily reconfigured with a new AD device, invalidating any action or communication performed by the compromised nodes.

Smart Cities

In smart city applications, hundreds of smart nodes are deployed over urban areas and perform operations based on coordination. These devices need to be installed in a physically secure manner but this is not always possible, due to the inability to supervise the devices and the complexity of the task for city staff. As a result, the devices are often not secure, bearing the same secrets as their peers, and waiting to be compromised.

With the ADS, we envisage applications where the supervisor of a smart city managing team will be able to give her employees an authority device and ask them to install and configure the nodes. Upon completing the installation, the city employees will simply connect the AD to each of the nodes and the configuration would be performed automatically, without exposing any secrets to any persons involved. Furthermore, the AD can keep a list of all the nodes it came in contact with, making it possible for the supervisor to verify that all nodes were set up correctly.

A similar topology involving multiple projects with different city officials running them, can be seen in Fig. 5.1a.

Corporate Computers

It is a common occurrence in corporate environments for employees to use 'off the shelf' computers that are reused when they are no longer needed. Also, certain employees might work on projects that involve multiple departments. As such, it would be beneficial for the IT department to be able to easily configure employee machines in a secure way.

In this scenario, resembling Fig. 5.1c, the same employee holds multiple authority devices, corresponding to different teams. Using the ADS, a large number of company workstations can be efficiently configured to provide granular access and communication between different departments. Due to the minimal user interaction required, this configuration can be performed by team leaders and department heads, removing the need for

time-consuming requests to the IT department.

Thus, it is clear that the security and usability of the company's security policy is greatly improved. Without authority devices, the IT team would have to manually install secrets to each machine and consequently be in knowledge of the secrets. Furthermore, in larger companies, managing the complexity of machines belonging to different departments is often problematic, resulting in insecure solutions of granting access to more entities than it is required.

Smart metering

Smart meters have been recently introduced in a many households around the world. They are devices which are relatively cheap and easy to install, and enable the collection of a wealth of energy consumption data. Besides the obvious advantage of automatically submitting billing information, smart meters allow providers to make better use of their resources, in an increasingly competitive field. At the same time, consumers are able to use the same data to make informed decisions about their energy use, reducing their energy bills while improving their quality of life.

However, due to their unattended nature, the metering devices are vulnerable to a range of attacks that can be alleviated with the use of an authority device. Firstly, the issue of secure delegation of duties is tackled by pre-configuring the meters in a secure environment. Subsequently, ADs can be passed to technicians who will perform the installation and enrolment of the meters. The identifying secrets of both the AD and the meters are never exposed to any human operators, thus the provider can be certain that, once the ADs are returned, no more meters can be enrolled and unapproved ADs have not been created. With the additional ability of ADs to keep track of all their transactions, it is possible to verify that the correct meters have been configured and trace back configuration issues to the technician responsible.

It is also easy to imagine the monetary incentives for creating counterfeit meters and ADs. The presence of the unclonable core in every device ensures that such activities bear a high cost, removing a major part of the incentive. At the same time, the methods discussed in this thesis allow existing meters to verify their peers, and detect and report unexpected group members.

We use smart metering applications as a recurring example to highlight the value of the methods proposed in the remainder of this thesis.

5.3 Protocols

The proposed scheme acts as an enabler for the unclonability stack of Section 2.4 through providing the following features:

- Key material is initially generated when the device is powered on using the inherent, unclonable randomness of the PUFs. The key is regenerated when needed without being stored in non-volatile memory. (Algorithm 5.1: Key Generation)

- The ADs and the nodes are introduced prior to deployment, as an extra layer of security. This allows for a decoupling of the owner and the actual holder of the authority device. (Protocols 5.1 and 5.2: Setup, Verification)
- After their initial introduction to the AD, nodes can be verified at any point, using their internal PUFs. (Protocol 5.2: Verification)
- Nodes can be added to one or more ownership domains (neighbourhoods) by interacting with one or more authority devices. (Protocols 5.3 and 5.4: Enrolment)
- After their addition to a neighbourhood, nodes are able to join additional neighbourhoods with additional authority devices but only after the approval of the initial authority device. (Protocol 5.4: Enrolment)
- After their addition to a neighbourhood, nodes are aware of their membership, can exchange public keys with their neighbours, and authenticate them. (Protocols 5.6 and 5.7: Key Exchange, Mutual Authentication)
- After their enrolment, nodes are able to verify requests of their owning authority devices, using digital signatures.
- Authority devices are able to decommission nodes of their authority domain, effectively removing them from the neighbourhood. (Protocol 5.5: Decommission)

The required set of protocols can be divided in four conceptual domains: key generation, member preparation, membership management, and member interaction. These domains and their relationships are visualised in Fig. 5.2.

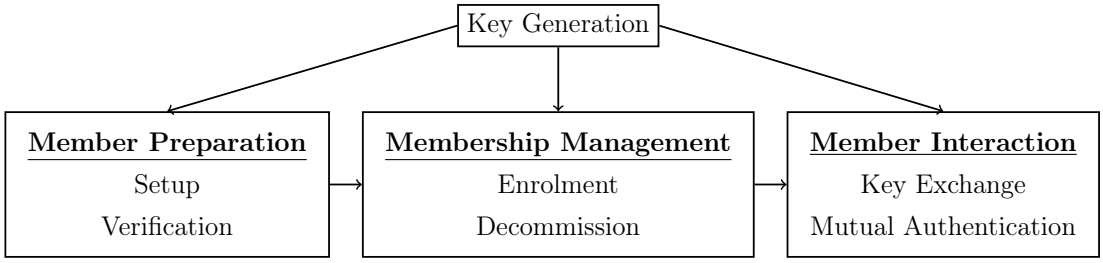
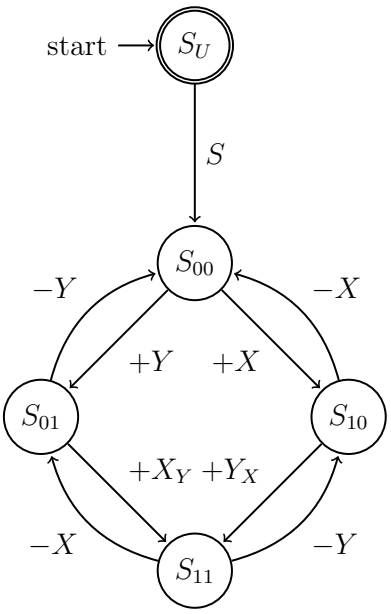


Figure 5.2: ADS protocol domains

To formulate the following protocols we use a multiple ownership scenario, including two authority devices and a number of nodes. Every node starts with no configuration in state S_U. After performing the Setup process (reaching state S_{00}) and being deployed in the field, the node is enrolled by one of the authority devices X or Y and transitions to states S_{10} or S_{01} respectively. Subsequently, the node can start interacting with other enrolled nodes, enrolled with the second AD, or decommissioned. The different states of a node in the two-AD scenario are shown in Fig. 5.3. As can be seen from the descriptions below, the assumption of only two ADs is merely contributing to the clarity of our descriptions and all methods and protocols can be extended to an arbitrary number of authority devices.



Symbol	Action
S	Setup
$+X$	Enrolment with X
$+Y$	Enrolment with Y
$+X_Y$	Enrolment with X, approved by Y
$+Y_X$	Enrolment with Y, approved by X
$-X$	Decommission with X
$-Y$	Decommission with Y

Figure 5.3: Node states in two-AD scenario with authority devices X and Y

5.3.1 Key Generation

An important feature of the ADS is the elimination of private data from both non-volatile memory and human interaction. The unclonable key generation process connects the two lower layers of the unclonability stack, the *unclonable provider* and the *unclonable core*.

The generation takes place on every device after it is powered on, using the entropy provided by the PUF block when it is queried with a random input. The output of the PUF is then used as a seed for the key generation of the chosen public key algorithm. Common public algorithms, including RSA and ECDSA, use a PRNG for their key generation, which can be seeded with the PUF response. Methods for generating a high-entropy seed from PUFs have been extensively covered in literature[10], [69], [110], [111], and discussed in Chapter 7.

The random challenge is stored in non-volatile memory to allow the reproduction of the key pair after a device power cycle. To avoid unnecessary calculations, public keys can also be stored in non-volatile memory and recalled, but their private counterpart is never stored or shared outside the boundary of the cryptocore.

Algorithm 5.1 (Key Generation)**.**
Device D is equipped with a cryptocore. At the end of the procedure, D possesses a public key P_D and a secret key S_D.

1. D generates random challenge $C = RNG_D()$.

2. D evaluates the PUF with the challenge, generating a seed $R = PUF_D(C)$ and derives the key pair $(P_D, S_D) = PKG(R)$.

3. D stores the challenge C to allow key regeneration.

5.3.2 Setup

Prior to the deployment of the nodes, a preparation step is performed between the AD and the nodes, in an environment controlled by the system owner (considered secure). This process, which we call 'Setup', serves as an introduction for nodes and ADs, configuring them to recognise each other. This is achieved by combining the secrets of the node and the AD, to protect the CRP of the node in case of AD compromise.

In addition, the Setup enables the important feature of *authority delegation*, allowing for the AD to be passed to an honest-but-curious third party. Regardless of the holder of the AD, after the successful completion of the Setup phase the nodes are prepared to recognise legitimate ADs and accept further commands from them.

Protocol 5.1 (Setup)**.** *Node N and authority device X. At the end of the protocol, N and X have been introduced and are able to verify each other. See Fig. 5.4.*

1. *X verifies that N has not been decommissioned and aborts on failure.*
2. *X generates a random nonce T_N and a random PUF challenge C_N and sends them to N along with its public key.*
3. *N uses C_N as a challenge to its PUF to generate a response $K_N = HASH(PUF_N(C_N))$.*
4. *N sends (K_N, P_N) to X.*
5. *X generates the response $R_N = HASH(PUF_X(K_N))$ and stores the tuple (P_N, T_N, C_N, R_N).*
6. *X replies with an acknowledgement.*
7. *N stores P_X in its list of potential ADs.*

5.3.3 Verification

The Verification protocol is based on the CRPs previously stored on the AD during the Setup protocol described above, and essentially prevents the AD from enrolling any nodes other than the ones pre-approved during Setup. Additionally, nodes that take part in a Verification exchange are made aware of the public keys of their potential ADs, improving the security of subsequent exchanges.

This protocol is designed to run infrequently, when a node is first enrolled or when changes are made to the configuration of the system. Nevertheless, it is described separately from the enrolment process, as it can be also be used independently. To prevent replay attacks, the CRP exchanged during this protocol is discarded and a new Setup protocol execution is required before further verification executions.

Protocol 5.2 (Verification)**.** *Node N and authority device X. At the end of the protocol, X has verified that it was introduced to N during the Setup protocol. See Fig. 5.5.*

1. *X retrieves (P_N, T_N, C_N, R_N) from its database.*

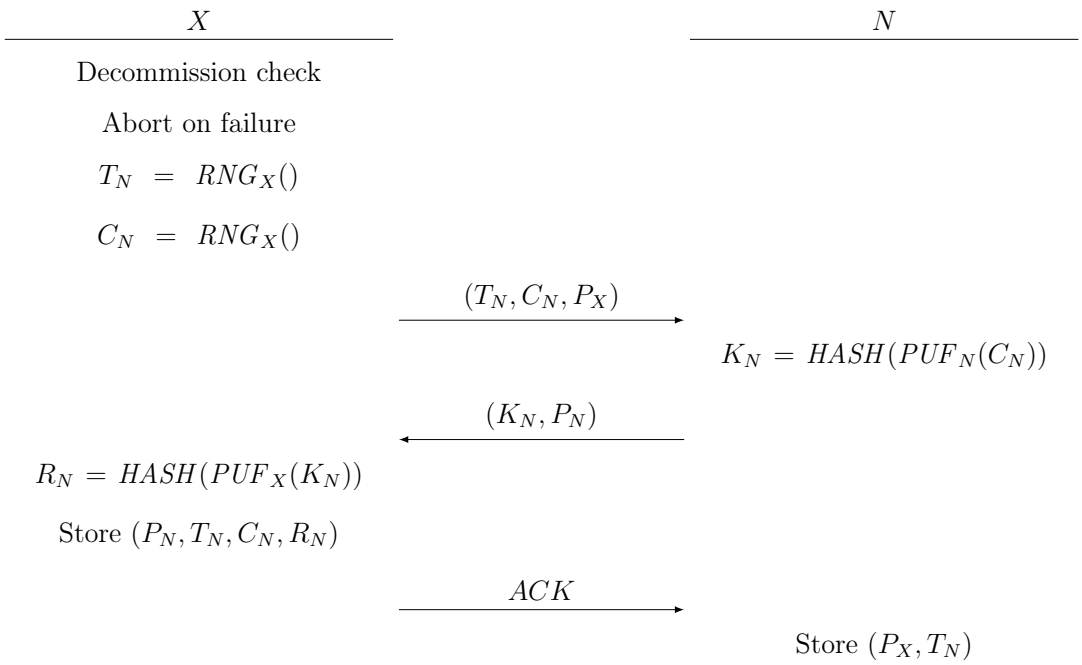


Figure 5.4: Setup

2. *X checks if N has been decommissioned and aborts if true.*

3. *X signs the challenge C_N concatenated with the none T_N and sends the result to N.*

4. *N uses the public key P_X stored during Setup to verify the request, and aborts on failure.*

5. *N verifies the signed challenge and aborts on failure. This prevents unauthorised parties from querying the PUF of N in an attempt to model it.*

6. *N uses C_N to generate the PUF response $K'_N = HASH(PUF_N(C_N))$ and sends K'_N to X.*

7. *X generates the PUF response $R'_N = HASH(PUF_X(K'_N))$, verifies that $R'_N = R_N$ and aborts on failure.*

8. *X replies with an acknowledgement and discards (C_N, R_N).*

5.3.4 Enrolment

Node enrolment is equivalent to an AD (and its holder) claiming ownership of a node. An enrolled node is regarded as a member of the neighbourhood controlled by the corresponding authority device, and possesses the necessary information to prove its membership and communicate with other members. In practice, the AD adds a node to a neighbourhood by signing its public key, thus certifying the validity of the public key.

There are two variants of this protocol to satisfy the scheme goals: (a) for nodes that have not been enrolled before, shown in Protocol 5.3, and (b) for nodes that are already

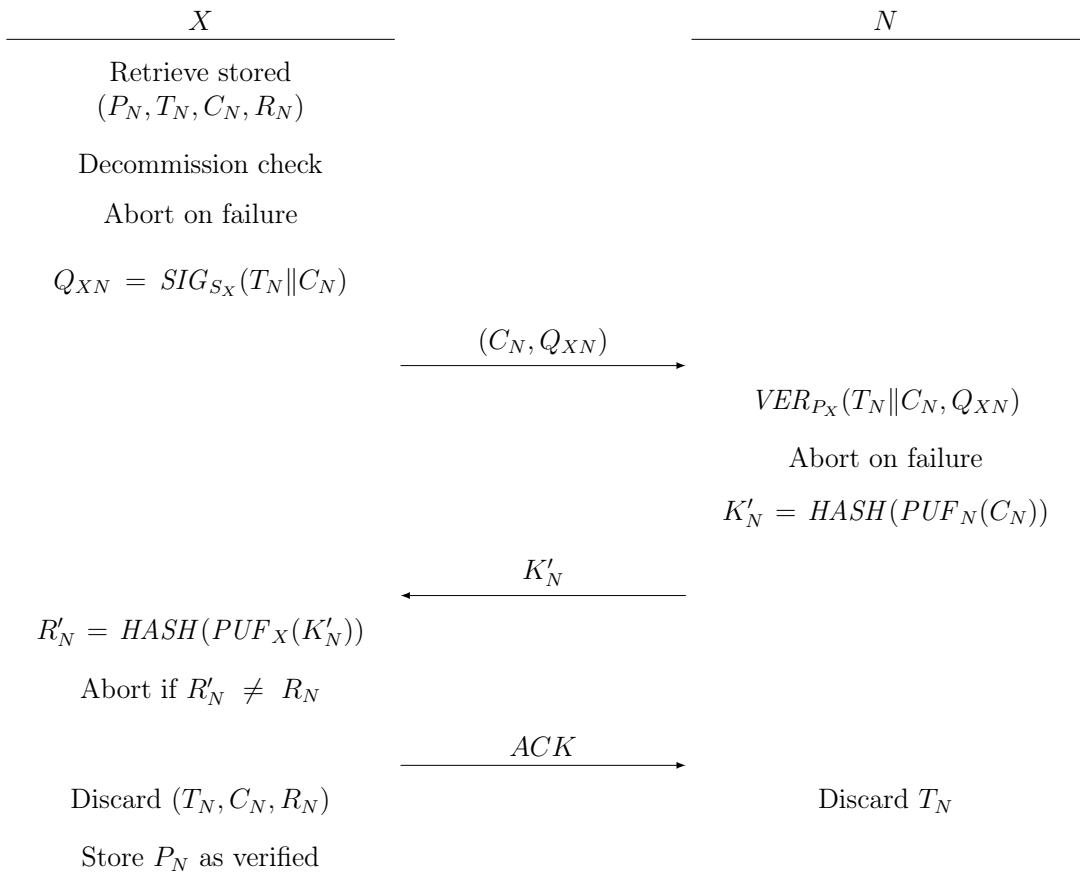


Figure 5.5: Verification

enrolled, shown in Protocol 5.4. In the latter case, at least one previous AD is required to participate in an approval process for the new AD. The task of verifying the legitimacy of the new AD is the responsibility of the holder of the previous AD and is thus not included in the protocol. Enrolling with a new AD however, does not remove any previous owners but creates *joint ownership relationships*.

Protocol 5.3 (Single Enrolment)**.** *Authority device X and Node N, previously set up with X but not enrolled. At the end of the protocol, N is enrolled with X. See Fig. 5.6.*

1. *X executes the Verification protocol (Protocol 5.2) with N and aborts on failure.*
2. *X initiates the enrolment by sending the signed public key of N:* $Q_{XN} = SIG_{S_X}(P_N)$.
3. *N uses the public key* P_X *from its list of approved ADs to verify the request. This list is populated by either the Setup protocol (Protocol 5.1) or the first part of the multiple ownership enrolment process (Protocol 5.4). A aborts on failure of the signature verification.*
4. *N verifies that the received signature matches the received public key and its own public key and aborts on failure.*
5. *N stores* (P_X, Q_{XN}) *and replies with an acknowledgement.*

6. *X stores N in its list of enrolled nodes.*

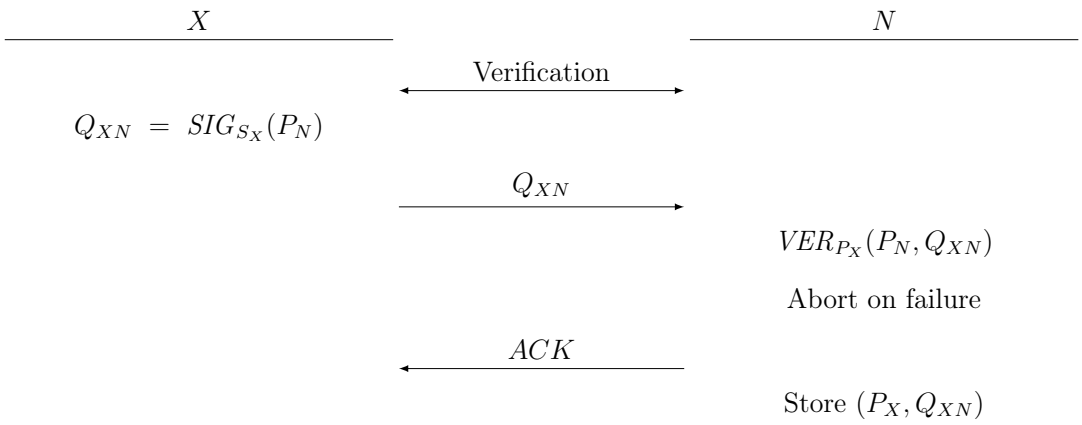


Figure 5.6: Enrolment (single ownership)

In the multiple ownership case, the Verification protocol is omitted, since new ADs, unknown at the time of Setup, can be added over the lifetime of the system.

Replay attacks by recording the (P_Y, Q_{XY}) and replaying it for a different, unauthorised AD are implicitly prevented: N initially accepts the new AD (since the signature verification is successful) but in the subsequent enrolment protocol, the malicious AD will have to sign its request with the private key of Y, which never leaves the latter's cryptocore.

Protocol 5.4 (Enrolment (multiple ownership))**.** *Authority devices X and Y, and node N, previously enrolled with X. At the end of the protocol, N is enrolled with Y and stays enrolled with X. See Fig. 5.7.*

1. *Y sends P_N to X, along with its public key.*
2. *X verifies that N is enrolled and aborts on failure.*
3. *X signs $(P_Y \| P_N)$ and sends it to Y as Q_{XY}.*
4. *Y verifies the signed response and replies with an acknowledgement.*
5. *Y initiates the enrolment by sending its public key and Q_{XY} to N.*
6. *N verifies Q_{XY} with P_X, stored during its enrolment and aborts on failure.*
7. *The remainder of the process is analogous to Protocol 5.3.*

5.3.5 Decommission

Decommissioning a node refers to removing it from one of its neighbourhoods. Since every neighbourhood is controlled by an AD, this AD is responsible for instructing the node to delete any signed keys. As mentioned, the nodes are assumed to be visually verifiable

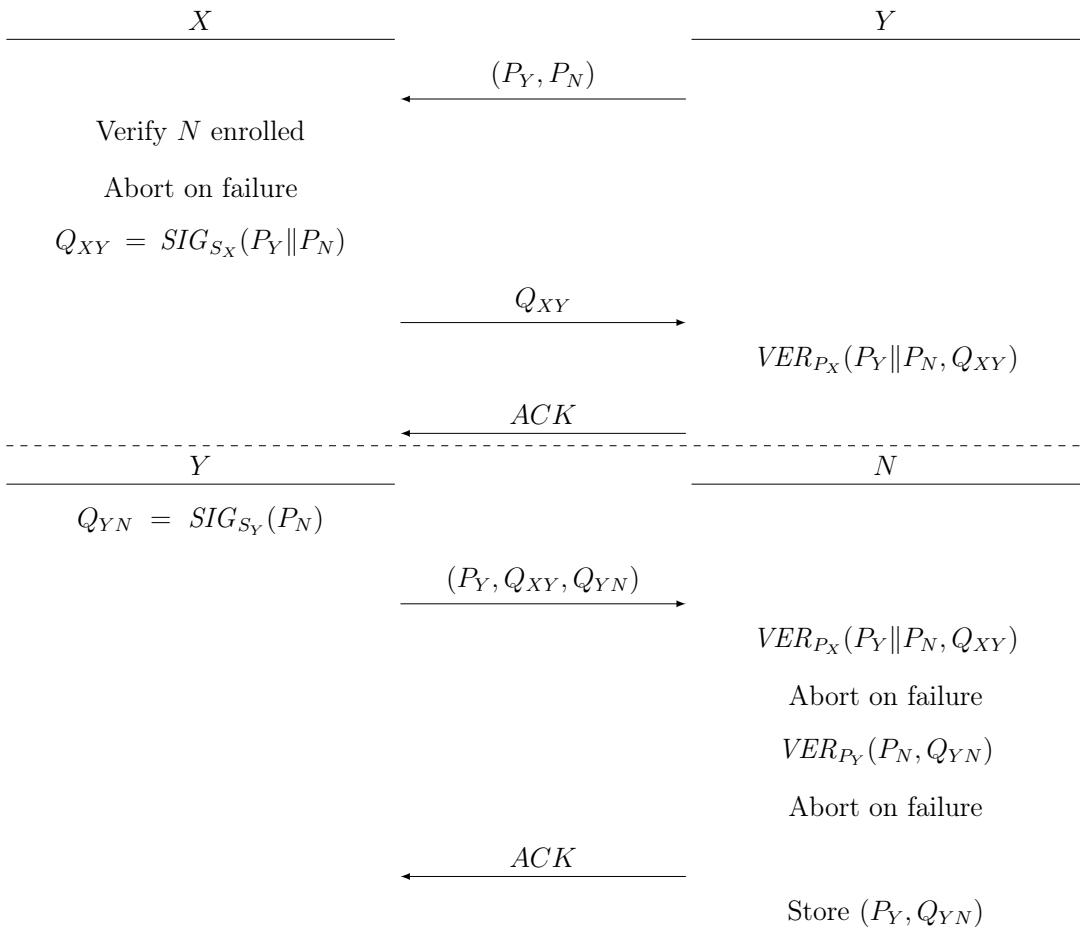


Figure 5.7: Enrolment (multiple ownership)

during their interactions with an authority device. Additionally, the decommission protocol ensures that the node is still in possession of its private key and, in extension, of its PUF. Based on these guarantees, the actual deletion of the signed keys is left to the node, and no further key revocation takes place.

Additionally, ADs keep a 'black list' with the nodes they decommission, in order to take appropriate action if they encounter them again. In case a node is believed to be compromised, a new enrolment round can take place with a fresh key pair for the authority device. At the end of this round, all previous signatures of the AD would be rendered invalid and the nodes would instantly seize to accept them.

Nodes are not required to keep track of their decommissioned peers, since the Key Exchange and Mutual Authentication protocols described below involve the signed public key of every node, as it was stored during the Enrolment protocol. This signed key is removed from storage during decommission.

Protocol 5.5 (Decommission)**.** *Authority device X and Node N, previously enrolled with X. At the end of the protocol, N no longer belongs to the neighbourhood controlled by X, has erased the relevant identifiers, and has been added to a decommission 'black list' kept by X. See Fig. 5.8.*

1. *X initiates the protocol by sending its public to N, as an identifier.*

2. *N verifies that it has been enrolled with X and aborts on failure.*

3. *N generates a random nonce T and sends it X.*

4. *X replies with the signed nonce.*

5. *N verifies the signature and aborts on failure.*

6. *N replies with an acknowledgement and discards* (P_X, Q_{XN}) *from its storage.*

7. *X removes N from its list of enrolled nodes and adds it to a black list.*

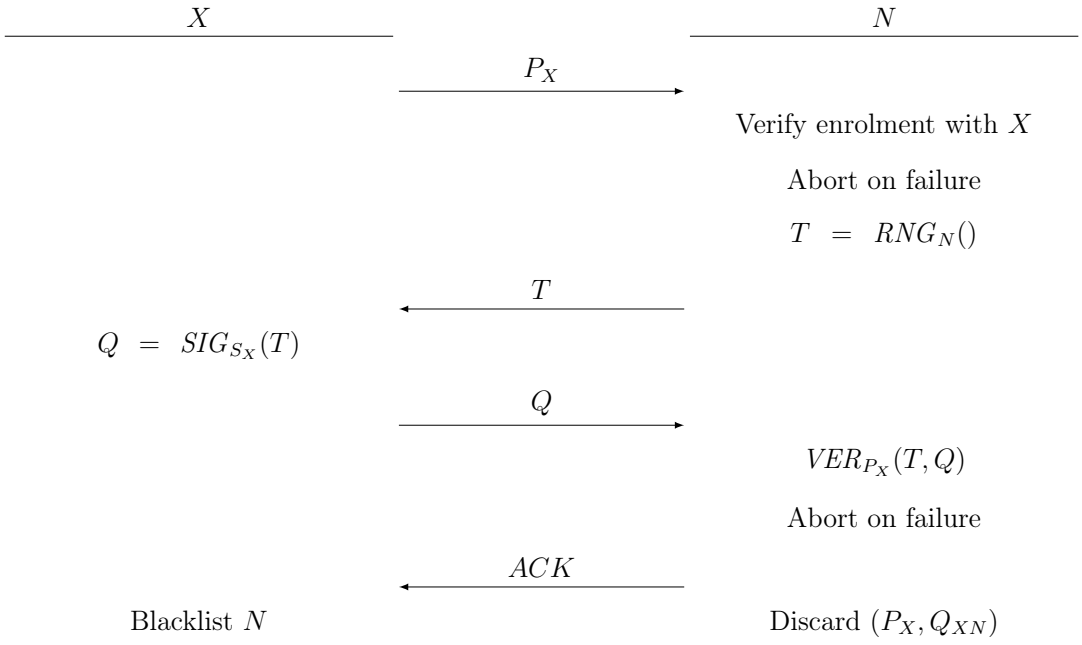


Figure 5.8: Decommission

5.3.6 Key Exchange

This protocol is used to exchange keys between nodes, resembling public key certificate methods. Since each node's public key is signed by the AD during the enrolment phase, all nodes can verify each other's key validity using the public key of the AD. This key, also stored on the nodes while they were enrolled, acts as an *authority anchor*. By verifying the signature of each other's public key, the nodes can identify their neighbours and form neighbourhood relationships that will enable further, higher-level protocols. Additionally, the exchanged keys can be used for the establishment of secure communication channels between the nodes, to provide any required application-level services.

Protocol 5.6 (Key Exchange)**.** *Nodes A and B belonging to the same neighbourhood, both enrolled with AD X. At the end of the protocol, both nodes possess the public keys of each other. See Fig. 5.9.*

1. *A initiates the protocol by sending* (P_X, P_A, Q_{XA}) *to B.*

2. *B verifies that it has been enrolled with X and aborts on failure.*

3. *B verifies the received key signature Q_{XA} and signature and aborts on failure.*

4. *B replies with (P_B, Q_{XB}) as stored in the enrolment phase.*

5. *A verifies the received key signature and aborts on failure.*

6. *A replies with an acknowledgement.*

7. *A stores (P_B, Q_{XB}) and B stores (P_A, Q_{XA}).*

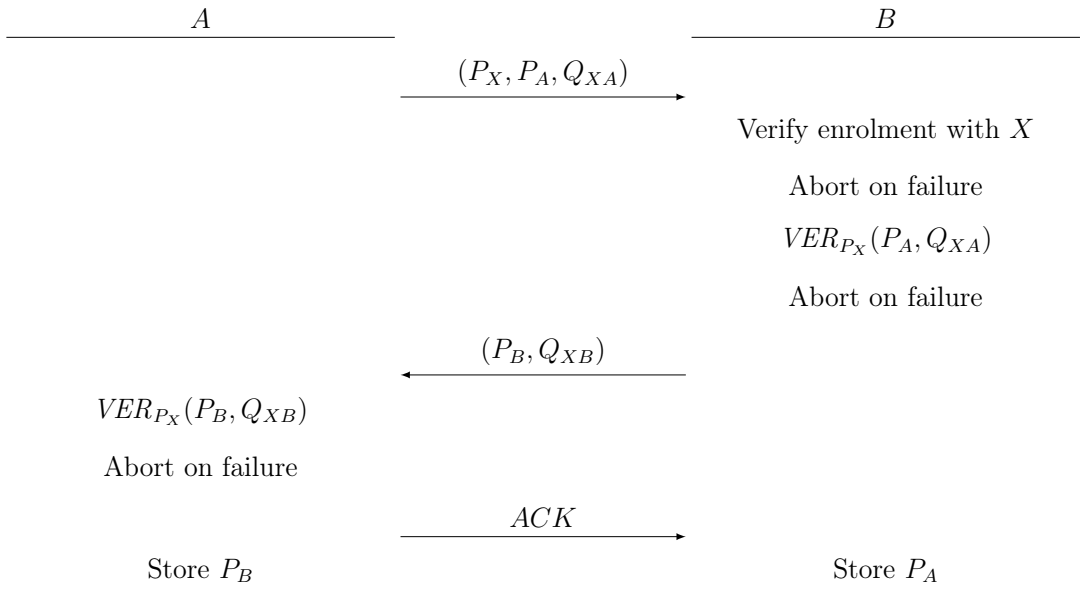


Figure 5.9: Key Exchange

5.3.7 Mutual Authentication

Based on the exchanged public keys, nodes can mutually attest the authenticity of their remote partners by taking turns in signing and verifying a random nonce with their respective key pairs. As with all asymmetric cryptography operations, signing and verifying signatures is relatively expensive and thus node authentication is designed to be part of relatively infrequent protocols. Both nodes include the signatures of their public keys, received during the Enrolment protocol, to ensure that the authentication will fail for decommissioned nodes.

Protocol 5.7 (Mutual Authentication)**.** *Nodes A and B belonging to the same neighbourhood, both enrolled with AD X, and have previously exchanged public keys. At the end of the protocol, the nodes have authenticated each other and verified each other's group membership. See Fig. 5.10.*

1. *A retrieves (P_B, Q_{XB}) which was stored during the Key Exchange protocol.*

2. *A generates a random nonce T_A and signs it, including its public key signature, as it was received from the AD.*

3. *A initiates the authentication by sending the public key of X to B, followed by the signed nonce.*

4. *B verifies that P_A is in its list of peers enrolled by P_X and aborts on failure.*

5. *B retrieves (P_A, Q_{XA}) which was stored during the Key Exchange protocol.*

6. *B verifies the received signature and aborts on failure.*

7. *B generates its own random nonce $T_B = RNG_B()$ and signs it in a similar manner, including the received random token.*

8. *B sends the nonce and the signature to A.*

9. *A verifies the received signature and aborts on failure.*

10. *A replies with an acknowledgement.*

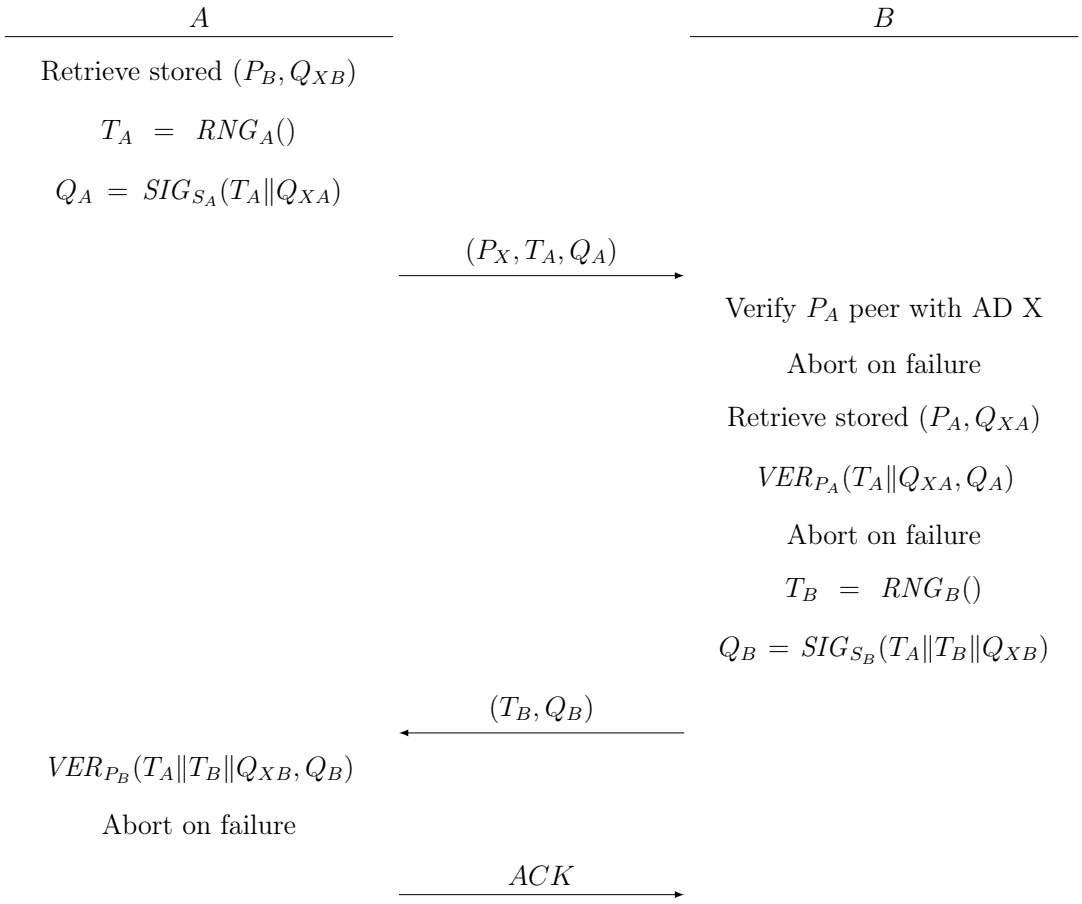


Figure 5.10: Mutual Authentication

5.4 Security Analysis

In this section we provide a security analysis of the Authority Device Scheme in the form of a number of lemmas and theorems. In the following, we are taking into account a single Probabilistic Polynomial Time (PPT) adversary, referred to as $\mathcal{A}$ and bound by the adversary model of Section 5.2.3.

The main operational goal of the scheme is to enable the establishment of neighbourhoods, controlled by their respective authority device. This goal is to be achieved while ensuring the authentication of all system entities with secrets stemming from their PUFs. Thus, our analysis focuses on the (in)ability of $\mathcal{A}$ to masquerade as a legitimate entity (be it an AD, an enrolled node, or a new node) and perform actions that are allowed only to legitimate entities.

We construct our analysis based on a taxonomy of common protocol attacks informed by the work of Boyd and Mathuria[20], and Carlsen[112]. Both works summarise the same classes of attacks albeit using somewhat different terminology, and we extend their taxonomy to include physical security aspect of out work.

Our analysis shows that $\mathcal{A}$ will have to resort to guessing the secret information if she is to compromise any part of the protocol. Thus, sufficiently long secrets are adequate to prevent successful attacks under the PPT adversarial assumption. The implementation choices shown in Chapter 9 provide an overview of sufficient secret lengths based on current best practices, which are expected to remain secure for at least the next few decades.

Lemma 5.1 (Cryptographic Primitive Security (Cryptanalysis))**.** *The security of the cryptographic primitives employed in the ADS is guaranteed against $\mathcal{A}$, with high probability.*

Proof. As per the adversary model of Section 5.2.3, $\mathcal{A}$ is unable to break or otherwise circumvent the security of the underlying public key cryptographic primitives. Additionally, due to the relaxed requirements for specific cryptographic algorithms, the ADS can be easily adapted to use algorithms proposed in the future (e.g quantum secure) as long as they are secure against existential and selective forgery. Thus, we conclude that $\mathcal{A}$ has only negligible advantage over random guessing of the private keys involved. □

Lemma 5.2 (Cryptocore Security)**.** *In the event of a node compromise, $\mathcal{A}$ is not able to gain access to or tamper with the internal secrets of the node.*

Proof. Based on the features of the cryptocore, attempts at invasive probing or tampering with the hardware will result in the internal secrets being either destroyed or significantly transformed. In addition, all protocol operations (and especially the ones accessing secret data) are executed inside the cryptocore boundaries and their results are made available through the cryptocore's interfaces. Thus, a physical attack would be reduced to compromising the cryptocore itself. □

Lemma 5.3 (Private Key Confidentiality)**.** *$\mathcal{A}$ is not able to access the private key of any entity.*

Proof. As seen in Algorithm 5.1, the private key for every entity is dynamically generated by its PUF and is never stored in non-volatile memory. Due to the unpredictability property of the PUF, it is infeasible to derive the private key directly from the key challenge, without access to the generating PUF IC. In addition, the private key is only used inside the cryptocore which by Lemma 5.2 is physically secure. □

Lemma 5.4 (PUF Modelling)**.** *$\mathcal{A}$ given unrestricted access to an entity for a limited period of time, is not able to copy the entity's PUF.*

Proof. According to the properties of PUFs, as they were introduced in Chapter 4, the following are true:

- The knowledge of a limited number of CRPs does not allow $\mathcal{A}$ to predict additional CRPs.
- The CRP space is sufficiently large and given her PPT capabilities, $\mathcal{A}$ is unable to exhaustively query the PUF in a bounded time period.
- $\mathcal{A}$ is unable to control the PUF behaviour with the aim of producing specific responses.

Additionally, via normal protocol interaction in the context of the ADS, the raw PUF responses never leave the boundaries of the cryptocore, before being transformed by a cryptographic hash function. As per their properties seen in Chapter 3, these functions have uniformly distributed outputs and are one-way transformations. Thus, acquiring a number of responses through ADS interactions does not provide $\mathcal{A}$ with any advantage in comparison to guessing attacks. Finally, $\mathcal{A}$ is unable to manipulate the protocols involved in ADS in order to gain access to the internal secrets of any PUF, as shown in Lemma 5.5 and Lemma 5.8. □

Lemma 5.5 (PUF CRP Confidentiality)**.** *Upon inspection of the storage of a compromised authority device, $\mathcal{A}$ is not able to recover the CRPs corresponding to any node, except with negligible probability.*

Proof. The Setup protocol shown in Protocol 5.1, makes use of both the PUF of the AD and the PUF of the node. This method essentially combines two internal secrets from the node and the AD into a unique output. Additionally, due to the unpredictability property of PUFs and as discussed in Lemma 5.4, $\mathcal{A}$ is unable to predict the response of a PUF to a given challenge, except with negligible probability. Thus, the response of the node is stored in a form that provides no sensitive information about the node's PUF in case of a later compromise of the AD. □

Lemma 5.6 (Message Modification)**.** *$\mathcal{A}$ is not able to modify an exchanged message and the modified message to be accepted by its intended recipient, except for negligible probability. This attack is commonly referred to as 'modification' or 'man-in-the-middle' attack.*

Proof. The exchanged messages contain public keys, PUF CRPs, or random nonces. All three kinds of information are either already known to the recipient, and thus their modification will be trivially detected, or they are signed with the private key of the sender and verified upon receipt. Thus, $\mathcal{A}$, due to Lemmas 5.1 and 5.3, is not able to modify the content of those messages unless she successfully guesses the corresponding private key, a task which she can achieve with only negligible probability.

Nevertheless, it should be noted that the ADS does not prevent $\mathcal{A}$ from modifying messages and causing their rejection on the receiving end. However, by consistently doing that, $\mathcal{A}$ would create an attack resembling the total obstruction of the communication channel (denial of service attack). The recovery from such a case would be the responsibility of a different class of protocols, possibly exploiting redundant communication paths. □

Lemma 5.7 (Correctness)**.** *A legitimate entity can authenticate to another legitimate entity, except for negligible probability.*

Proof. A legitimate entity would have access to its PUF thus being able to generate the required key pair and successfully participate in the authentication protocol Protocol 5.7. □

Lemma 5.8 (Replay Attacks)**.** *An eavesdropping adversary $\mathcal{A}$ can use previously transmitted data to successfully perform any of the protocols of the scheme, only with negligible probability. This attack is commonly referred to as a 'replay' attack.[20], [112].*

Proof. The data transmitted during the protocols comprises:

- Random tokens which are generated by a cryptographically secure random number generator. Thus, token values are not repeated over the course of the protocol operation and replaying of older messages containing a random token will fail.
- Public keys which do not provide any advantage if they are recorded and replayed, since all protocols involve the use of corresponding private key.
- Signatures generated with secret keys that are not permanently stored and never revealed to third parties.
- PUF CRPs involved in the Setup (Protocol 5.1) and Verification (Protocol 5.2) protocols.

Evidently, the only truly useful information exchanged between system entities are

the PUF CRPs. However, the Setup protocol is performed in a secure environment in the absence of adversaries and the Verification protocol discards used CRPs, thus replaying them would have no result for $\mathcal{A}$. Additionally, when random data are not included in the interactions, replay attacks are implicitly prevented by the protocols, as discussed in the relevant protocol descriptions. □

Lemma 5.9 (Impersonation)**.** *An adversary $\mathcal{A}$ is able to impersonate or clone any authority device or node, only with negligible probability.*

Proof. Before a node has been introduced to an AD with the Setup protocol, it is possible for it to be replaced with a malicious node. However, this case is not part of our application scenario in which the Setup is performed in a trusted environment, under the control of the system owner.

After the node has performed the Setup protocol, by virtue of Lemma 5.1, impersonating a system entity would require the compromise of said entity's private key, due to Lemma 5.2. Thus, the probability of the former being impersonated is the same as that of predicting the responses of the node's PUF. By Lemma 5.4 this probability is negligible. □

Lemma 5.10 (Damage Containment)**.** *In the event of a node compromise, an adversary $\mathcal{A}$ does not gain any advantage towards compromising the rest of the system.*

Proof. A node compromise can take place either during its normal operation or after it has been decommissioned. In the first case, the node does not have access to secret information of any entity but itself. At the same time, due to Lemma 5.2 the security of the node's PUF and secret key is guaranteed. In the second case of compromise after decommission, the above still holds, with the addition of Lemma 5.11. □

Lemma 5.11 (Decommission)**.** *An adversary $\mathcal{A}$ is able to enrol a decommissioned node, only with negligible probability.*

Proof. The authority device corresponding to a neighbourhood is always involved in the enrolment of new nodes in that neighbourhood. In addition, as shown in Protocol 5.5, the authority devices keep a list of nodes that were previously decommissioned and will disallow any re-enrolment attempts. Thus, the only options for $\mathcal{A}$ to enrol a decommissioned node are: to tamper with the AD's list or to compromise the private key of the AD. Both cases are prevented: the former due to Lemmas 5.1 and 5.2, and the latter due to Lemmas 5.2 and 5.3. □

Lemma 5.12 (Malicious Authority Device)**.** *An adversary $\mathcal{A}$ is able to use a malicious authority device to gain control the system, only with negligible probability.*

Proof. During the Enrolment protocol (Protocol 5.3 and Protocol 5.4), the nodes verify that the ADs attempting to enrol them are legitimate via a list of authorised ADs. This list is populated during either the Setup protocol or, in case of multiple enrolments, the

Enrolment protocol itself. The Setup is performed in a secure environment, disallowing the existence of adversaries. In addition, when attempting to enrol a node which has already been enrolled, an AD is required to obtain the approval of at least one of the ADs controlling the node's neighbourhood. This approval would fail for a malicious AD. □

Theorem 5.1 (Authority Device Control)**.** *Assuming the physical security of the appropriate ADs, there can be no change to the membership attributes of any node, whether the node is legitimate or malicious, except with negligible probability.*

Proof. By the above lemmas, it is evident that modifications of any form to the membership of a node in any neighbourhood require the participation and authorisation of the appropriate authority devices. Great care has been taken to minimise the trust relationships among the entities taking part in the ADS enabling those ADs to become the roots of trust for the system.

This is an important feature of the ADS since it allows for the verification of the state of the system at any point, as well as the assurance that there are no distortions to the neighbourhood memberships as long as the authority devices is under the control of trusted individuals. Providing further security for those ADs is mainly a social rather than technical issue but, due to the unclonability properties of our scheme, ADs are bound to be unique, existing at only a single place at any given moment, a property that guarantees the detection of their physical compromise. □

Theorem 5.2 (PUF Confidentiality)**.** *The Authority Device Scheme ensures the confidentiality of all private information used in the context of the scheme.*

Proof. The private information used by all entities throughout the operation of the ADS falls under two categories: PUF secrets, and private keys. Both are kept confidential on a physical level as shown in Lemma 5.2 and are never shared as shown in Lemma 5.8. Additionally, PUF secrets are protected as per Lemma 5.4 and Lemma 5.5, and private keys are unavailable to adversaries as shown in Lemma 5.3. □

Theorem 5.3 (ADS Security)**.** *The Authority Device Scheme provides methods for the establishment and maintenance of network clusters, in an unclonable and physically secure manner.*

Proof. Using the above lemmas, we can conclude that the Authority Device Scheme is both correct and secure.

By Theorem 5.2 legitimate nodes are able to join and leave neighbourhoods and by Lemmas 5.7 and 5.8 nodes are be able to interact in those neighbourhoods. At the same time, malicious attempts to access and modify the nature of the neighbourhoods are prevented and contained as shown in Lemmas 5.4, 5.6 and 5.9 to 5.12.

Additionally, by Lemma 5.2 and Lemma 5.3 we can conclude that private entity information is protected at a physical level. Finally, by Lemma 5.5, Lemma 5.8, and Theorem 5.2 we can be confident that the unclonability goals of the scheme are achieved. □

5.5 Formal Verification

The proposed protocols were formally verified in the applied pi calculus with the automatic cryptographic protocol verifier ProVerif[113].

Operation	Implementation	Model Source
PUF	Symmetric cryptography with dedicated private key.	-
RNG	Natively supported (random tokens).	ProVerif user manual.
HASH	Natively supported.	ProVerif user manual.
CONCAT	Implemented with native functions.	-
SIG	Natively supported.	ProVerif user manual.
VER	Natively supported.	ProVerif user manual.

Table 5.2: Basic operations in ProVerif

ProVerif allows the modelling of the majority of the operations used in our protocols, as illustrated in Table 5.2. Accurately modelling cryptographic operations in the applied pi calculus is not a straightforward task, thus we chose to use the models already provided in the user manual of the software, where possible. The only remaining operation, the PUF, was modelled as a symmetric cryptography primitive, with a random key that is used exclusively for the PUF and is not shared with any parties apart from the PUF owner.

In order to model the authentication of a party to another, we use the *correspondence assertions* available in the ProVerif. As a result, authentication is modelled as a correspondence assertion between two events, the start of the protocol and its successful completion. For example, in the encoding of the Mutual Authentication protocol, shown in Appendix B.1.5, the authentication of A to B is verified by ensuring that for each occurrence of the event 'authenticatedA' there is a previous occurrence of the 'startB' event.

It should be noted that the multiple ownership enrolment protocol was not verified, since the authentication of AD Y to node N is achieved by proxy: AD Y is visually authenticated by the holder of AD X. Furthermore, the authenticity of the public keys during the key exchange is attested by the associated AD signature, a fact that cannot be captured in the applied pi calculus model.

In addition to the verification of the authentication properties (summarised in Table 5.3) we used the native support for secrecy queries to verify that the private keys of all parties remain secret throughout the protocol executions. Finally, the secrecy of the PUF state was verified with secrecy queries regarding the secret key used for the each PUF. The detailed ProVerif queries and their corresponding results can be found in Table B.1.

Protocol	Security Properties	Verified With
Setup and Verification	Authentication of N to X.	Correspondence assertion.
Enrolment	Authentication of N to X.	Correspondence assertion.
Enrolment (multiple ownership)	Authentication of Y to N.	-
Decommission	Authentication of dec. request.	Correspondence assertion.
Key Exchange	Public key authenticity.	-
Mutual Authentication	Authentication of A to B.	Correspondence assertion.
	Authentication of B to A.	Correspondence assertion.

Table 5.3: Security properties as captured in ProVerif

5.6 Performance Discussion

Acquiring a PUF response is associated with low performance overhead since it is often as simple as a memory or I/O device access. Thus the performance of the ADS is directly affected by the selection and the implementation of the cryptographic primitives that enable it. Public key cryptography is commonly considered to be quite costly from the point of view of computation, as well as the size of ciphertexts and signatures. In this section we briefly discuss the different performance aspects of the proposed scheme.

Computation

Evidently, public key cryptographic operations, mainly signature generation and verification, constitute the main computational tasks of the ADS. A precise estimation of the computation costs is exceptionally hard to derive due to the dependence of these costs, not only on the selected algorithms but also on their implementation and its nature. Assuming that the required operations are supported (in hardware) by the cryptoprocessor included in the cryptocore (as discussed in Section 5.2), the computation overhead can be greatly reduced.

However, the algorithm selection also has an effect on other aspects of the scheme, discussed below. Two options are commonly available in the field of public key cryptography: the Rivest Shamir Adleman(RSA) cryptosystem[114] and systems based on Elliptic Curve Cryptography(ECC)[115]. A detailed comparison of these solutions is out of the scope of the ADS. Nevertheless, ECC is widely accepted to provide equivalent security to RSA with greatly improved performance and is thus recommended for use in resource-constrained systems[116].

Storage Requirements

ADs store lists of nodes that have been set up, enrolled, or decommissioned with a list size that scales linearly with the number of system nodes. Likewise, nodes store lists of their peers along with their public keys, with their size scaling linearly with the number of peers. In addition, during protocol executions the required storage is limited to a few kilobytes per peer and the stored information is discarded after the end of the protocols

unless it belongs to one of the aforementioned lists. As a result, the storage requirements on all devices scale linearly with the number of the entities with which they interact, and remain constant regardless of the number of protocol interactions.

The information stored in the aforementioned lists comprises PUF CRPs, public keys, and device identifiers. The raw CRP length is largely dependent on the PUF used but it is normally compressed to a few bytes, through a hash function (16 to 64 bytes in commonly used hash algorithms[117]–[120]).

On the other hand, the public key length is determined by the cryptographic algorithm used. For example, ECC provides equivalent security with up to 4 times shorter keys compared to RSA, a length that can be further reduced with key compression, due to the reliance on curve points[121].

Finally, device identifiers are only required to differentiate between devices of the same system leading to potentially very short identifiers depending on the system size (e.g. two bytes can represent over 65 thousand distinct devices). In many implementations, identifiers may be replaced by public keys for convenience, potentially sacrificing storage space, as was done in the reference implementation of Chapter 9.

Message Size

Similarly, the size of protocol messages is heavily dependent on the selected public cryptography algorithms, due to a big part of the payloads comprising public keys or signatures. Thus, reducing the size of both directly leads to shorter messages and decreased network overhead, calling for the use of primitives that support the smallest possible keys and signatures.

The second major component of protocol messages are random tokens. While the size of these tokens is not directly prescribed by the protocol, they are required to have sufficient length so as not to be repeated over the lifetime of the system.

5.7 Conclusion

Integrating the unclonability primitive in consumer devices would have a profound impact on the societal concepts of ownership, authority, and eventually trust. In this chapter we have presented a first step in that direction, the ADS, a collection of security protocols that allow the formation of network neighbourhoods, with the aim of exploiting the primitive of unclonability to provide novel features for networked systems, focused on Machine-to-Machine and Internet-of-Things scenarios.

Our work aims to integrate unclonability into the basis of modern networking scenarios while retaining and even improving the usability of existing security solutions. This is achieved in two ways: by designing protocols which guarantee their security without requiring excessive configuration, and by limiting the system authority operations to physical entities that can be secured and verified. The most important advantage of our solution is the combination of these two methodologies into a scheme that isolated the secrets

of all the devices of the system, irrevocably locking them into the physical domain, a feature that enables the users to apply, hand over, verify, and destroy the secrets as they would with a physical key. At the same time, this key cannot be duplicated and thus bears a much higher value than a 'clonable' counterpart.

6. Continuous Pairwise Authentication

In this chapter we discuss methods for establishing *unclonable links* (as they were defined in Section 2.4.4) via continuous pairwise authentication protocols. The term *continuous* signifies that the authentication state is established once and subsequently renewed periodically for an unbounded period of time. In practical terms, a balance is required in the definition of the renewal frequency, aiming to achieve the security objectives of the system for the entirety of its useful life.

Continuous authentication is able to provide persistent trust in the topology of the system, by repeatedly verifying the paths comprising the network graph. As a result, it is possible to detect distortions to the graph and, by extension to the topology of the system, as they are discussed in Section 6.2.1.

In practical applications, this detection capability enables the system to guard against unauthorised modifications in a traceable manner. Thus, the system administrator would be able to unquestionably verify the current and historical state of the system topology.

We can consider the following scenario, involving our example of smart metering applications. An energy provider installs smart meters in a number properties which are in range of each other, creating a mesh network. In current deployments, the provider relies on the physical security of the smart meters to prevent tampering, and ensure their correct and honest operation. However, the security potential of the mesh topology is not utilised.

Using continuous authentication, smart meters can establish peer-to-peer links which can in turn provide information about the status of each meter. These links will present a high level of redundancy, allowing the system to reach a consensus about the state of each meter in a manner that is robust against single failures or localised attacks. For example, while a malicious user could potentially control a small number of meters, it is unlikely that they would have access to a large number of residential or commercial properties without leaving a trace.

Continuous authentication based on PUFs is especially suitable for such an application, due to the fact that internal PUF secrets are easily destroyed by intrusion attempts. Consequently, tampering with the hardware will lead to the collapse of the unclonable links created between the smart meters, and thus information about physical attacks will be propagated to higher levels of the network stack where appropriate action can be taken.

6.1 Contributions

As we already mentioned, designing protocols that take advantage of unclonability properties is not a straightforward task. To that end, we begin by describing the purpose of a continuous, pairwise authentication protocol and we introduce the basic requirements and features of an ideal protocol. We then propose two practical protocol designs, aimed at different adversarial scenarios, and periodically taking place between nodes of a neighbourhood.

These protocols exploit the inherent unclonability of the nodes' PUFs to establish proof that each node is not impersonated by a different device, whether that device is legitimate or malicious. We view potential impersonations as *faults* which create the topology distortions discussed above.

The novelty of the work presented in this chapter is summarised in the following:

- An ideal PUF-based authentication protocol is formulated for the the peer-to-peer (P2P) setting, a context that, to the best of our knowledge, has not been focused on in previous works.

- The primitives of 'ratcheting' and PUFs are combined, providing break-in recovery and continuous renewal of unclonability.

- No assumptions are made regarding resource homogeneity among devices.

- Our P2P scenario does not allow for continued reliance on third parties that is common in networking scenarios. No third party is involved in the authentication process after its initialisation, reducing the attack surface and improving scalability.

- Only one CRP from each device is exchanged and stored in every round. An extensive CRP database or a model of the PUF behaviour are not required on the verifier side, disallowing impersonation and improving scalability.

- An authority action, in the form of manual administrator input, is required to start the protocol or reset it after the detection of a potential compromise.

Due to the above properties, the proposed protocols create *unclonable links*, where modifications to the involved nodes directly lead to a change in the state of the link between them. This is in contrast to traditional pairwise communication where the identity of the endpoints has a weaker representation at the link level.

6.2 Ideal Protocol

While defining the goals of PUF-based pairwise authentication protocols, it is useful to describe an ideal such protocol. In this section we discuss a number of points which need to be taken into account from different design perspectives. However, a protocol that

satisfies all of these requirements is an ideal, theoretical construction, and in practice trade-offs are required, as seen in our proposed protocols later in this chapter.

To begin with, the overarching goals of the protocols in question are to prevent adversaries: (a) from impersonating one or more nodes *without* physical access to the nodes, and (b) from cloning identifying secrets of the nodes *with* physical access to the nodes.

An ideal authentication protocol based on PUFs, ensures that the inherent unclonability provided by the unclonable functions is used as a *root of trust* for creating unclonable links. To successfully exploit this powerful primitive one needs to involve the PUFs as integral parts of the protocol, rather than as generators for static keys which will remain unchanged over the lifetime of the system and could be easily compromised.

6.2.1 Fault Taxonomy

It is useful for the development of the protocol to discern the following node faults:

Node removed and reinserted: A transient fault signifying that a node is temporarily unavailable, possibly removed by an adversary and tampered with.

Node replaced/cloned: A node has been replaced by a different device, with entirely different hardware. The new device is impersonating the legitimate node it replaced.

Node destroyed/missing: After a period of time, a node remains unresponsive to requests for authentication.

Node failed to authenticate: After a number of attempts, a node fails to authenticate successfully.

In a pairwise topology, the faults mentioned above can occur as: (a) a single fault, where only one of the nodes is affected, and (b) a double fault where both nodes exhibit irregularities. These faults directly lead to a distortion of the topology of the pair and its wider neighbourhood, as seen in the taxonomy introduced in Section 2.4.5. In the case of a double fault, the whole pair has been removed, replaced, destroyed or missing. This kind of fault is evidently out of the scope of a pairwise protocol and has to be mitigated by neighbourhood-level methods similar to the ones discussed in Section 10.2.1.

Faults can also be classified as malicious or benign. Security is mainly interested in the detection of malicious actors and thus related protocols do not normally take into account faults stemming from other causes (e.g. network conditions). In both cases however, appropriate action must be taken upon detection. Additionally, a fault can be transient, occurring for a short period, or permanent, having long-lasting effects on the topology of the system. While, at first thought, transient faults might be overlooked, they could also be the result of a compromise, and thus need to be detected nonetheless.

6.2.2 Adversary Taxonomy

From an unclonability point of view, an adversary aims to impersonate a legitimate node. We distinguish between two main types of adversaries:

Physically Passive: A physically passive adversary follows the modified Dolev-Yao model introduced in Section 5.2. She has the ability to observe and manipulate messages but does not have any physical access to the nodes, and is unable to break the security of the cryptographic primitives used.

Physically Active: In this case, the adversary is able to physically tamper with the nodes, giving her access to data stored in memory. On the contrary, by virtue of the cryptocore, the adversary in not able to access the data inside the cryptocore boundaries, replace the PUF of a node, directly access the PUF, or remove it and place it in a different device. In the context of topology, the adversary has the ability to replace nodes with different devices, remove nodes from the field, tamper with node hardware (except for the cryptocore) or completely destroy nodes. However, the adversary is not able to physically compromise a device indefinitely without being discovered by system operators.

A compromise can take place in the following node states:

Before initialisation: Before it is initialised for the authentication protocol, the node is unknown to its peer. As a result, the security of the node is handled by the methods introduced in Chapter 5.

After initialisation: Shortly after its initialisation, the node has been introduced to its peer and some initial identifying information is stored in its memory. Thus, appropriate measures should be taken for the protection of this identifying information or the information should be regarded as public.

After a number of protocol steps: A node in this state is trusted by its peer to the highest possible extend. As a result, compromising this node is of high importance, since a trust relationship has already been established between the two nodes.

6.2.3 Security Requirements

Remote Communication

In a scenario where attackers cannot access the node hardware, we have the case of a physically passive adversary who can eavesdrop on and modify authentication messages exchanged over the communication channel. As a result, the ideal authentication protocol provides the following properties to ensure that an adversary without physical access cannot impersonate the participating entities:

- Integrity: The integrity of authentication messages is ensured. This is analogous to signing the messages. (Feature *Integ* in Table 6.2)

- Confidentiality: Authentication messages containing secret protocol information should remain confidential from malicious entities. This is analogous to encrypting the messages. (Feature *Conf* in Table 6.2)

- Forward Secrecy/Replay protection: Since the adversary is able to record and replay messages, the ideal protocol provides a method of invalidating past messages as soon as they have served their purpose, in order to avoid replay attacks. In the context of continuous authentication this requirement can also be referred to as forward secrecy[122]. (Feature *ForwSecr* in Table 6.2)

- Backward secrecy: We adopt a modified definition of backward secrecy[122]: upon compromising a single authentication message, the adversary does not gain an advantage against future protocol rounds. This feature is also called *break-in recovery* in more practical terms[6]. (Feature *BackSecr* in Table 6.2)

Local Data

To protect against an adversary with physical access to the nodes, the ideal protocol avoids transferring or storing unprotected sensitive information in local buses or storage elements.

In practice, this requirement is difficult to satisfy efficiently, since data storage and transfer lies at the heart of every computing device. Two approaches are often used for local data security, either separately or combined. The first approach is based on physical containment methods similar to the ones used in our cryptocore reference implementation seen in Chapter 7. The second approach relies on cryptographic methods, typically symmetric cryptography, to ensure that sensitive data is protected when it leaves the boundaries of a secure hardware component. Such methods have been extensively studied with several proposals including PUFs[58], [59].

Nevertheless, both approaches incur additional costs, often even during the protocol execution e.g. when data is encrypted before being stored in memory. Thus, the ideal security protocol is designed to minimise data exposure without making overly ambitious assumptions about the physical security of the participating devices. (Feature *LocProt* in Table 6.2)

PUF secrets

The requirements and caveats outlined in Section 2.5 are also present in periodic authentication protocols. To make matters worse, this periodicity contributes to the depletion of available CRPs. As a result, the PUF CPRs are used at the lowest possible rate.

6.2.4 Operational Requirements

The overall goal of a continuous authentication protocol based on PUFs is the detection of single faults, as they were defined in Section 6.2.1. Since authentication protocols aim to detect rather than handle such faults, there is no distinction being made between malicious

or legitimate changes in topology. However, to ensure the robustness of the system, the ideal protocol operates in the middle ground between detecting every transient fault (high number of false positives), and overlooking malicious distortions (high number of false negatives). Evidently, this is a trade-off between security and availability of the system and varies across applications. As such, the authentication frequency is an implementation parameter defined by the application requirements.

Additionally, the ideal protocol supports symmetric operation, which means that any member of the pair can initiate a protocol round in order to authenticate its counterpart. In essence, this requirement implies mutual authentication support, as well as a periodic renewal of the authenticating state, thus providing continuous authentication. (Feature *MAuth* in Table 6.2)

The construction of a CRP database on the verifier is a common method of identification and authentication in many protocols proposed in literature, as seen in Section 6.3. However, IoT systems comprise a large number of devices, and our target scenarios involve mesh topologies where each node has a multitude of direct neighbours. Thus, the associated security and storage costs rise exponentially with the number of CRPs included in the aforementioned databases. As a result, it is desirable to reduce the number of CRPs that are required to be stored during the lifetime of the protocol. (Feature *NoDB* in Table 6.2)

For similar reasons, user interaction, whether they are an administrator responsible for the configuration of the system or a simple end-user, is kept to a minimum for the ideal protocol. This ensures both improved usability (as it does not require extensive skills), and security as it does not depend on the user making the right security choices. (Feature *NoInt* in Table 6.2)

On the other hand, a deliberate user intervention which we call *authority action* is mandatory for the protocol to be initialised again, after the detection of a fault. In other words, the protocol is designed to abort its operation in the event of a fault by destroying the protocol state. Consequently, compromise attempts are guaranteed to be noticed and attended to by the holder of the appropriate authority, or rather the appropriate authority device. (Feature *AA* in Table 6.2)

6.2.5 Specification

In summary, the ideal protocol is initialised with an 'authority action' (using a security token, a password etc.) and is executed periodically and indefinitely until an authentication failure occurs. Both participating nodes are equipped with a PUF which is used in every round to inject fresh unclonability into the protocol. To achieve this, the nodes store a private state between protocol rounds and make use of a transformation function that allows the establishment of a temporary session state in each round. The process is outlined in Protocol 6.1.

In the following, functions with the prefix *St* refer to operations involving the node

state. The function *ReGen* generates a PUF response based on the supplied challenge and encrypts it with the supplied state. Several of the functions include a PUF as an input, signifying that the specified PUF is involved in the operation.

Protocol 6.1 (Ideal PUF-based Authentication)**.** *Nodes A and B belonging to the same neighbourhood, both enrolled with AD X. At the end of the protocol, both nodes have authenticated each other. See Fig. 6.1 for an illustration of the protocol.*

1. *Nodes A and B have the initial state S_A and S_B respectively, from previous protocol rounds.*
2. *A extracts the last PUF challenge C_B from its state with $StExt(S_A)$, and generates a new random PUF challenge.*
3. *A sends the two challenges to B.*
4. *B also extracts the last PUF challenge for A and generates a new one.*
5. *B derives the shared state based on its private state S_B and the challenge C_B with $S = StDer(S_B, C_B, PUF_B)$.*
6. *B generates a new PUF response with the new challenge C'_B and protects it using the state S. Both operations are included in $ReGen(C'_B, S)$.*
7. *B sends the two challenges and the new PUF response to A.*
8. *A follows the same process to derive the state S and generates its own new PUF response C'_A.*
9. *A updates its private state with $S_A = StUpd(C'_B, R'_B, S)$ and discards S.*
10. *A sends its new PUF response to B.*
11. *B updates its private state with $S_B = StUpd(C'_A, R'_A, S)$ and discards S.*
12. *Both nodes repeat the process periodically.*

It is evident that the above protocol omits several practical issues (e.g. susceptibility to replay attacks) that will be made apparent in the protocols of the following sections.

6.3 Related Work

The strong advantages of PUFs in entity authentication have sparked a great number of related protocols. However, to the best of our knowledge, the solutions proposed in this chapter are the first to address the P2P scenario. In this section, we briefly discuss previous work on PUF-based authentication, referencing only representative examples due to the number of different variations.

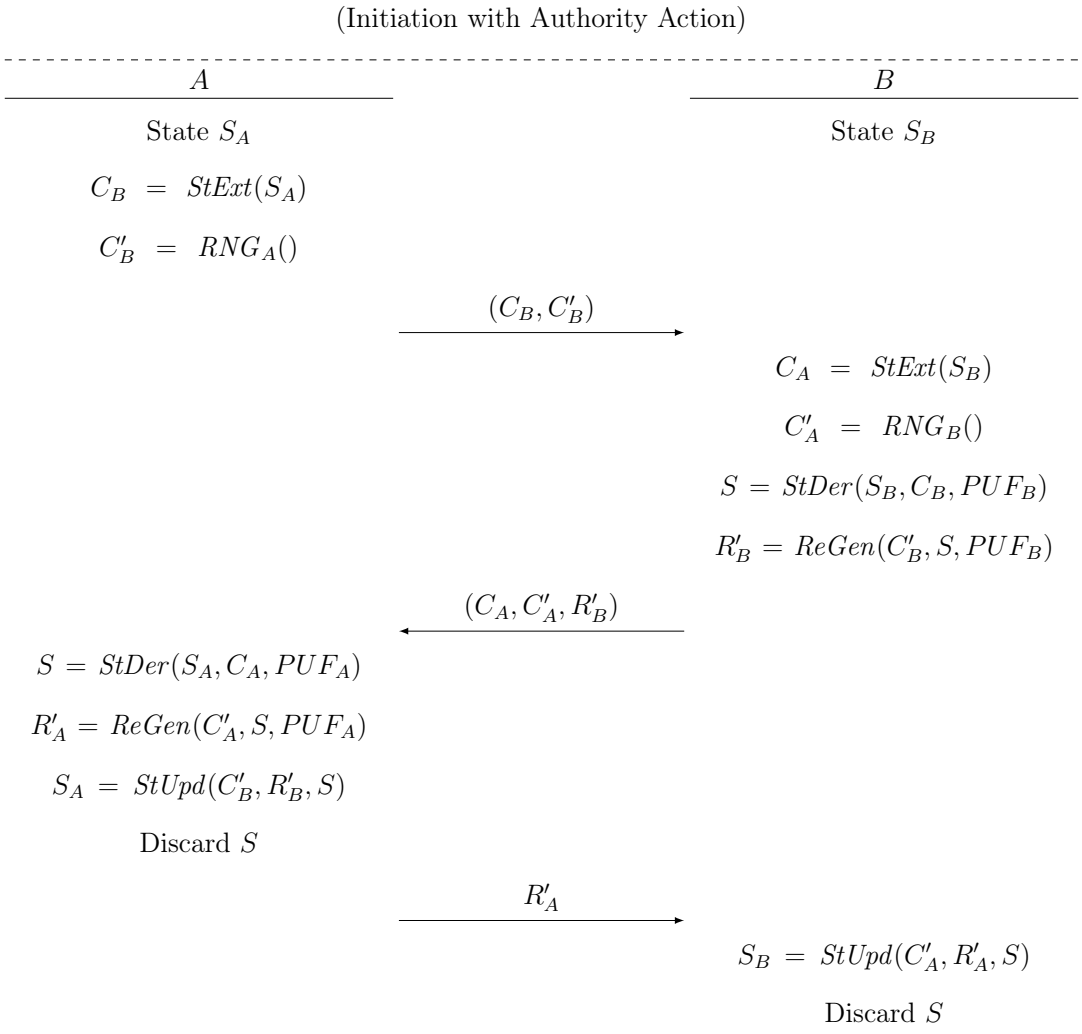


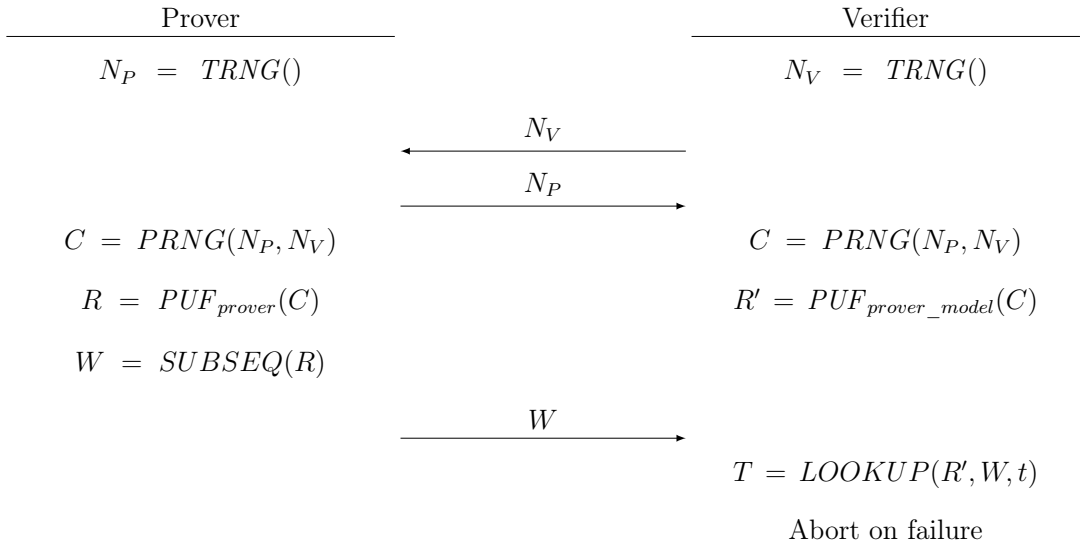
Figure 6.1: Ideal PUF-based authentication protocol

As expected, every proposal includes an enrolment phase during which an initial introduction is performed, and a verification phase which is executed every time authentication is required. The enrolment phase is used to exchange information that is subsequently used in the verification phase. In most cases, the verifier chooses a CRP that was previously exchanged, sends the challenge to the prover, and compares the received response to the expected value, to authenticate the prover.

A naive attempt to PUF-based authentication would be to simply replace the key generation component of a public key authentication protocol with a PUF-based alternative where the private key is sourced from the PUF. We are not aware of published work on this method, yet it is included for completeness.

The majority of existing protocols rely on the creation of a CRP database[54], [69], [123]–[125] or a model of the PUF on the verifier during the enrolment phase[126], [127]. This enrolment often uses an external PUF interface which is then physically disabled to prevent adversaries from accessing the PUF secrets. Fig. 6.2 illustrates the operation of the Slender protocol, one of the most sophisticated protocols making use of PUF models[126]. We think that these methods negate the unclonability advantage by effectively cloning

a big part of the PUF state and thus placing a great amount of trust on the verifier. Furthermore, storing multiple CRPs (along with error correction helper data) on the verifier involves storage costs which are unattainable for mutual authentication of IoT nodes with a large number of neighbours. Similarly, protocols involving costly operations similar to public key cryptography[16] are not suitable for our intended use case.



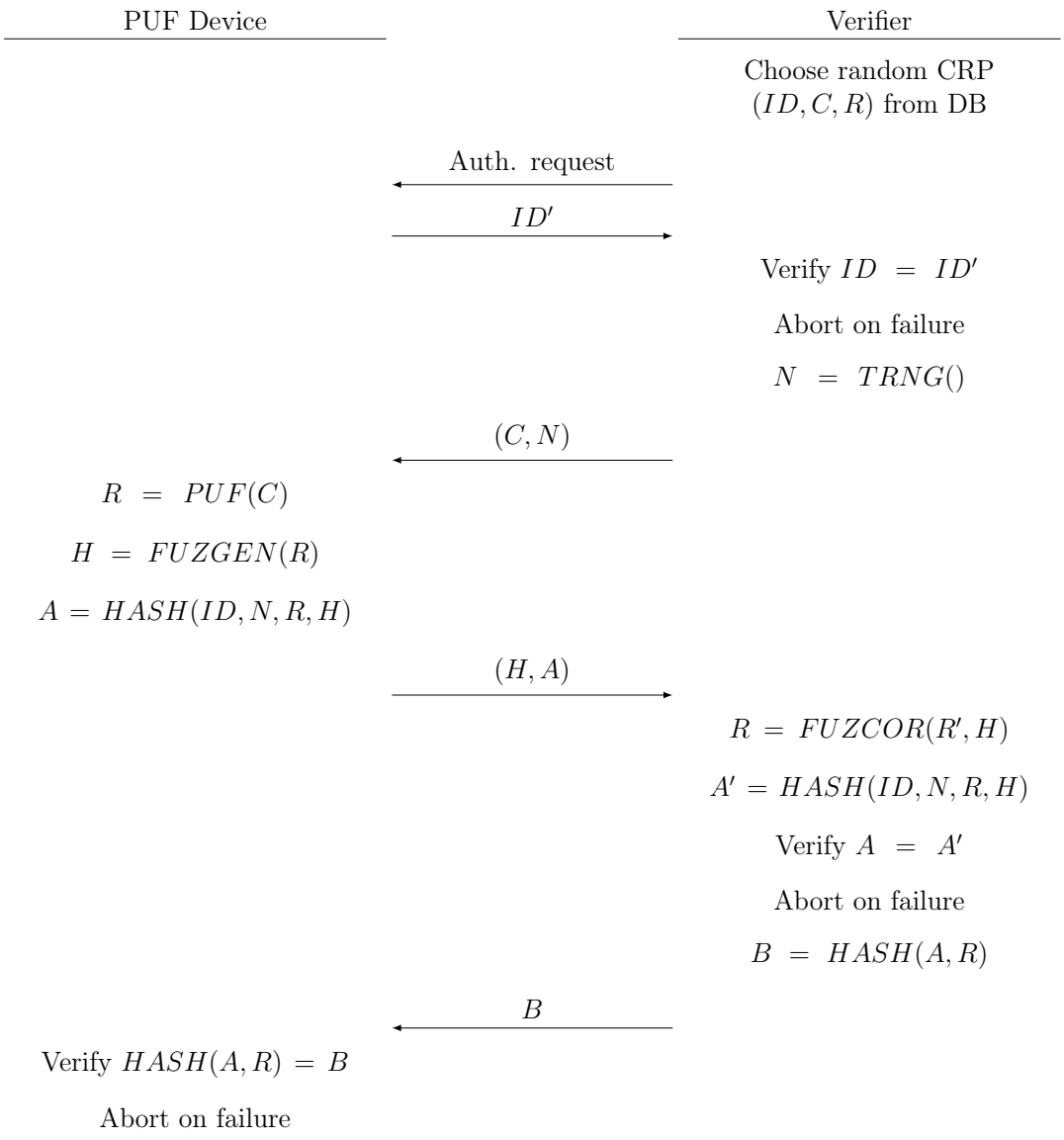
($PRNG$: generate pseudorandom output based on inputs, $SUBSEQ$: pick random subsequence, $LOOKUP$: look up subsequence in longer subsequence)

Figure 6.2: Slender PUF protocol [126]

Based on the asymmetry of resources between the prover and the verifier, 'reverse fuzzy extractors', were later introduced by Van Herrewege et al.[97]. As seen in Fig. 6.3, these protocols shift the highest complexity of the error correction overhead over to the verifier, since the prover's resources are often highly constrained (e.g. smartcards). Reverse fuzzy extractors provide significant performance gains for peers with highly asymmetric resources, and alleviate the issues of CRP databases. However, their inherent asymmetry and the additional cost they introduce on the verifier make them unsuitable for our scenario[124], [128].

The majority of the work discussed above includes the generation of one or more secrets based on the PUF which remain static after their generation[69], [97], [124]. In our view, this approach does not exploit the full unclonability potential as the PUFs are not constantly involved in the protocol but rather serve as an unclonable random number generator. On the other hand, proposals that aim to renew the authentication secrets often require several CRPs to be exchanged in each protocol round and do not include information from both participating entities, thus not achieving truly unclonable links and leading to faster exhaustion of the PUF responses[129], [130].

Finally, a number of other variants have been proposed with various aims including providing anonymity[131], removing the need for keeping the PUF state secret[132], or employing sophisticated PUF constructions to provide advanced features. These variants



($FUZGEN$: generate fuzzy helper data, $FUZCOR$: correct bit errors using helper data)

Figure 6.3: Reverse fuzzy extractor authentication [97]

provide valuable perspectives on unclonability protocols but nevertheless present similar challenges to the ones discussed above.

The different classes of solutions are summarised in Table 6.1, and in Table 6.2 the solutions are evaluated against the features of the ideal protocol of Section 6.2.

Method	**Advantages**	**Disadvantages**
Public key authentication	+ Straightforward implementation	- High computation overhead - Static secrets
CRP databases [54], [97], [123], [125], [126], [131], [133]	+ Native use of PUFs + Simple implementation + Low computation overhead	- High storage overhead - Static secrets
SIMPL systems[93] Matched PUFs [95], [134], [135] Commutative PUFs [96]	+ Native use of PUFs + Low computation overhead	- Not ready for implementation - High implementation overhead
PUF-Based zero knowledge identification [16]	+ Native use of PUFs + PUF responses not exposed	- High computation overhead - Static secrets
CRP Ratchet (Section 6.5)	+ Native use of PUFs + Low storage requirements + Low computation overhead + Supports authority devices + Refreshes secrets	- PUF responses exposed to prover
ZK CRP Ratchet (Section 6.6)	+ Native use of PUFs + PUF responses not exposed + Low storage requirements + Supports authority devices + Refreshes secrets	- High computation overhead

Table 6.1: Comparison of PUF authentication protocols

Method	**Integ**	**Conf**	**ForwSecr**	**BackSecr**	**LocProt**	**MAuth**	**NoDB**	**NoInt**	**AA**
Public key authentication	●	●	○	○	?	○	●	n/a	○
CRP databases [54], [97], [123], [125], [126], [131], [133]	○	n/a	○	○	○	○	○	●	○
SIMPL systems [93]	○	n/a	○	○	n/a	○	●	n/a	○
Matched PUFs [95], [134], [135]	○	n/a	?	?	n/a	●	○	n/a	○
Commutative PUFs [96]	○	n/a	?	?	n/a	●	○	n/a	○
PUF-Based zero knowledge identification [16]	○	n/a	○	○	●	○	●	○	●
CRP Ratchet (Section 6.5)	●	●	●	●	●	●	●	●	●
ZK CRP Ratchet (Section 6.6)	●	●	●	●	●	●	●	●	●

●: supported, ○: not supported, ?:unknown/unspecified, n/a: not applicable

Table 6.2: Feature evaluation of PUF authentication protocols

6.4 Preliminaries

The overarching goal of continuous authentication protocols is to establish a chain of trust via refreshing the authentication information in every protocol round, and injecting new unclonability to this information, sourced from the nodes' PUFs.

The proposed protocols are built on the renewal of the authentication secrets. Commonly known as 'ratcheting', refreshing the protocol secrets enables the system to recover from ephemeral secret compromise, a property referred to as 'post-compromise security'[136] or 'break-in recovery'[6]. Ratchets have been used extensively in various protocols, mainly concerning secure messaging [6], [13] and key exchange[137]. However, we are utilising the primitive in a somewhat different setting: our aim is to provide authentication rather than confidentiality.

We make use the term 'ratchet protocol' to describe authentication protocols which operate based on local state that has been established in previous protocol rounds and is refreshed with each new round. This refreshment is one-way, resulting in an inability to derive previous states from future ones and thus creating a temporal *authentication chain*. Additionally, we combine ratchets with PUFs, resulting in protocols with unclonability guarantees in addition to the forward secrecy provided by the renewal of secrets. Due to these properties, we refer to the proposed protocols as 'Challenge-Response Pair (CRP) ratchets'.

In line with Definition 3.1, the presented protocols do not provide security for application data but are focused on detecting distortions to the system topology. This allows us to make use of zero knowledge primitives in the second protocol variant. Nevertheless, the ephemeral common secrets established in the first variant can be used as session encryption keys for application data, and improved availability is also a positive side effect, due to the increased reliability which comes with distortion detection.

6.4.1 Notation and Definitions

In the following sections we use the standard notation found at the beginning of this thesis and summarised for convenience in Table 6.3. In addition, implementation details are omitted unless they are vital for the description of the protocols (a reference implementation information is provided in Chapter 9). Finally, it is assumed that the identity of the communicating parties is implicitly included in the exchanged messages by the protocols in charge of the low-level communication as is common e.g. in TCP/IP. This identity is only used to differentiate among peers and its integrity is not guaranteed.

6.4.2 Application Scenario

For the remainder of the chapter, we consider a pair of network nodes (A and B) which have been enrolled to the same neighbourhood with the methods described in Chapter 5. For simplicity, we only refer to one of the nodes as the initiator of the protocol. However, the protocols are designed to be symmetrically initiated and the roles of the participating

Symbol	Definition
AD_x	Authority Device x
N_x	Node x
ACK	Acknowledgement
ZK	Zero Knowledge
P_x	Public key of x
S_x	Private (secret) key of x
AC_x	Monotonic authentication counter for x
FC_x	Monotonic failure counter for x
$SIG_k(x)$	Signature of x with private key k
$VER_k(x, y)$	Verification of signature y of x with public key k
$ENC_k(x)$	Public key encryption of x with secret key k
$DEC_k(x)$	Public key decryption of x with public key k
ZKC(x)	Calculation of ZK commitment for value x
$ZKP_k(x, y)$	Generation of ZK proof with key k, value x and challenge y
$ZKV_k(x, y, z)$	Calculation of ZK commitment for value x
HASH(x)	Cryptographic hash of x
$PUF_x(y)$	Evaluation of the PUF of device x with challenge y
$RNG_x()$	Evaluation of the random number generator of device x
$HMG_k(x)$	Calculation of the HMAC of x with secret key k
$HMV_k(x, y)$	Verification of the HMAC y of x with secret key k
$\oplus$	Bitwise XOR operator
$\|$	Concatenation operator

Table 6.3: Summary of symbols

nodes are alternated to create a mutual authentication chain.

As in the previous chapters, the nodes contain a cryptographic core similar to the reference architecture described in Section 2.4.3. No assumptions are made for any other aspect of the software or hardware of the devices, or their homogeneity across the same or different node pairs. In addition, the nodes communicate directly and there is no third party involved in any part of their interactions.

Naturally, an initial introduction of the two nodes is required. Thus, both our designs comprise two separate phases: Initialisation and Ratchet Step. The first constitutes the authority action needed at the start of the system operation and, more importantly, after a failure is detected. The Ratchet Step phase enables the periodic part of the protocol and is repeated with a frequency which is application dependent.

6.4.3 Security Parameters

Both variants share the following security parameters which need to be chosen based on the individual application. These parameters aim to capture the realities of practical applications where transient faults are possible (e.g. networking issues) without being the result of malicious actions.

- The step interval t_s defines the time period between two successive, successful steps of the ratchet.

- The authentication threshold t_a defines the number of rounds or the time period after which a full public key authentication protocol (Protocol 5.7) is required.

- The failure threshold n_f defines the number of authentication failures after which a node is assumed to be compromised.

It is evident that the first two parameters have a direct effect on the security and resource consumption of the system. Decreasing either parameter causes increased communication and computation overhead, especially in the case of the authentication threshold. Due to the higher cost of public key authentication, t_a would typically be chosen in the range of tens or hundreds of rounds. Nevertheless, this cost is justified since it provides a method for the decommission of nodes via Protocol 5.5.

In many realistic scenarios, a single authentication failure can be regarded as a transient error and attributed to a number of factors including network failures. On the other hand, in a robust, wired network such errors have a much higher probability to be the result of security incidents. Thus, the threshold is an important aspect of the protocol, and a parameter that should be carefully calibrated depending on the application.

The above parameters also allow for the integration of more sophisticated primitives such as methods relying on the time between a challenge and a response to detect modelling attempts (e.g 'Simulation Possible but Laborious' work by Rührmair [93]). In fact, in one embodiment of our protocols a different relationship between t_s and n_f can exist: as n_f is approached, t_s is progressively reduced, thus closing the window of possible attacks at a rate that is proportional to the authentication failures that have already been detected.

In a different embodiment, t_s can be dynamically set based on information from higher layers of the stack and allow the system to transition between security levels depending on the nature of its operations at the time using variable values for the three security parameters. For example, the parameters can be set to a relatively high base value and decreased on demand shortly before highly sensitive data is expected to be transmitted through the nodes involved. Subsequently the parameters can be set back to their base values to conserve resources, until another data burst arrives.

6.4.4 Failure Procedure

A failure event can occur for two reasons: either a node fails to reply, or it replies with invalid information. In both the cases the incident is noted and, if the failure threshold has been exceeded, a *failure procedure* is initiated, aiming to irrevocably interrupt the normal operation of the ratchet, creating a condition where an authority action (in the form of repeating the initialisation phase) is required to restart the ratchet. In short, a node that detects an authentication failure:

1. Deletes the protocol state associated with its peer that is now in an unverified state.

2. Notifies the higher layers of the stack, making the neighbourhood aware of a security incident.

For clarity, in our protocol descriptions we assume the simplest and most secure case where $n_f = 1$, leading to an immediate initiation for the failure procedure upon detection of a single authentication failure.

6.4.5 Protocol States

Fig. 6.4 provides an overview of the protocols as a state machine, using the simpler case of the first variant as a reference. Based on the state diagram, it is easy to discern that every state is reachable from the initial state i. Additionally, the timeout transitions along with the failure threshold ensure that the protocol remains free of deadlocks.

6.4.6 Security Assumptions

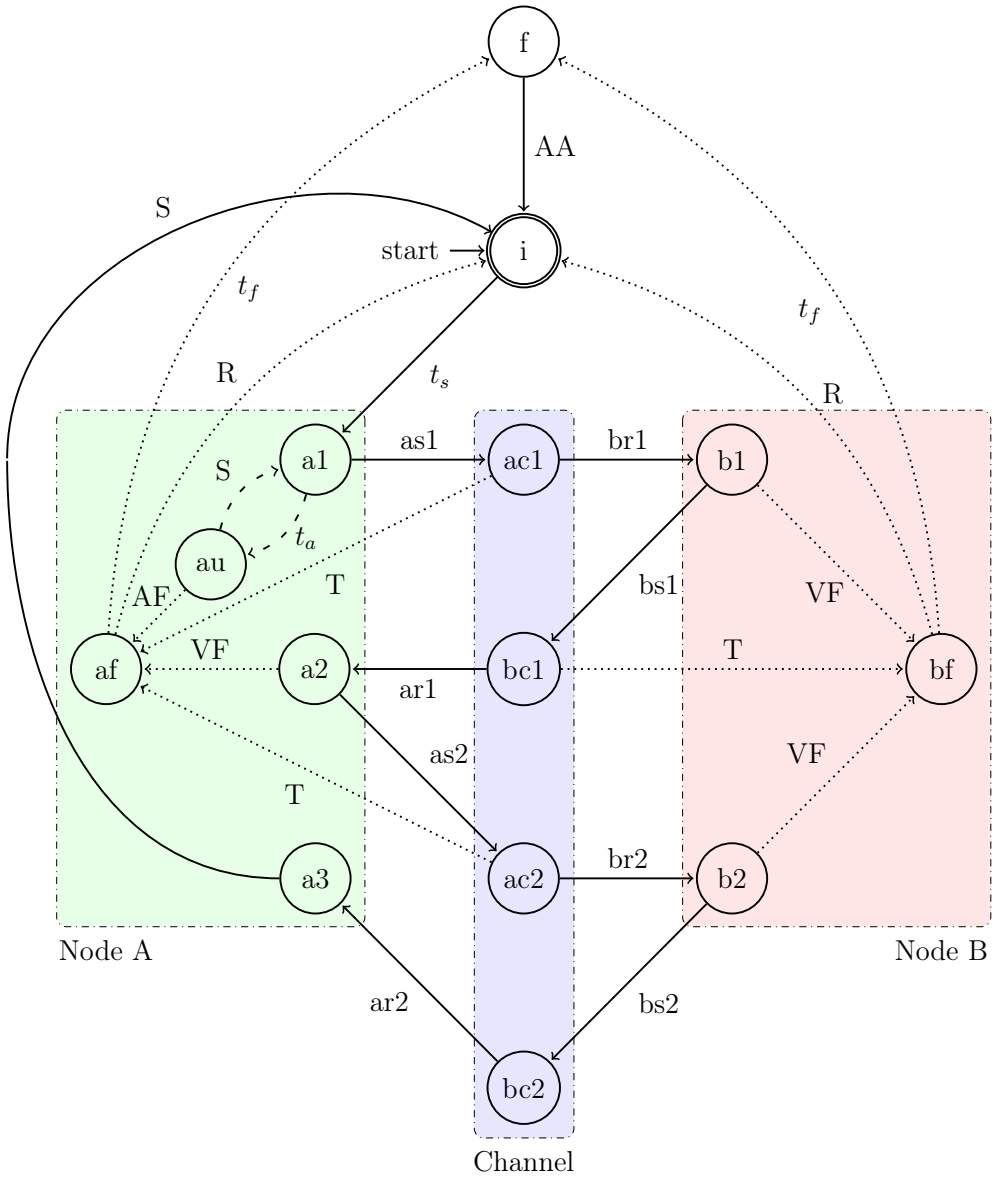
In addition to the capabilities of the adversary outlined in Section 6.2.2, the following assumptions are made regarding the security provisions of the system:

- An architecture similar to the one introduced in Chapter 7 is employed to prevent invasive attacks, protecting the PUF interfaces and the protocol state while it is executing. In essence, all the operations of the proposed protocols are performed within the boundary of the cryptocore and only their final results (which are stored or transmitted to peers) leave this boundary.

- While the protocol state that is stored between rounds (typically in non-volatile memory) does not need to remain confidential, its integrity is implicitly assumed, since PUF-based non-volatile memory protection methods can be easily envisioned and work to that extend already exists[59].

- Due to the unpredictability of the PUF behaviour and the size of the CRP space, it is deemed impractical for the whole set of CRPs to be enumerated. Thus no party, malicious or otherwise, can create a complete listing of CRPs for any of the involved PUFs.

- The same PUF unpredictability property removes the need for strict randomness requirements in the PUF challenge generation process. Thus, we refer to the employed random number generator simply as 'RNG' which can be a hardware TRNG if available, or simply a PRNG seeded with a PUF response (to ensure that the seed remains secret).



State	Action
f	Failure
i	Idle
ac1/ac2	A: Put message #1/#2 on channel
a1	A: Send auth. request and challenge
a2	A: Verify message, send auth. info
a3	A: Receive ack., increment auth. counter
af	A: Failure, increment failure counter
au	A: Execute public key auth.
b1	B: Verify message, send auth. info
b2	B: Verify message, send next round info
bf	B: Failure, increment failure counter
bc1/bc2	B: Put message #1/#2 on channel

Transition	Event
t_a	Auth. threshold exceeded
t_f	Failure threshold exceeded
t_s	Step interval expired
ar1/ar2	A received message #1/#2
as1/as2	A sent message #1/#2
br1/br2	B received message #1/#2
bs1/bs2	B sent message #1/#2
AA	Authority action
AF	Public key auth. failed
R	Reset, failure threshold not exceeded
S	Success
T	Timeout
VF	Verification failed

(auth.: authentication, info: information)

Figure 6.4: Ratchet protocol state diagram

6.5 CRP Ratchet

The first variant, *'Challenge Response Pair (CRP) Ratchet'* uses a PUF challenge-response authentication mechanism to support mutual authentication of the nodes. In essence, in each ratchet step, both nodes combine parts of their PUF secret state to identify each other. They subsequently use this combination to refresh their state, getting ready for the next ratchet step.

The strength of the protocol lies in the renewal of the authentication secrets through combining secrets from the participating entities. The unclonability provided by the PUFs is an integral part of the authentication protocol to continuously prove the existence of the PUF secrets, and the protocol described below is executed periodically to enable the establishment of trust between the participants. This feature achieves the goal of creating *unclonable links* between nodes and supports *break-in recovery*.

In comparison to the second design described later in this chapter, the focus of this variant is on achieving its security goals while retaining the smallest possible computation and energy footprint. For this reason, expensive cryptographic operations are avoided. It is however necessary to rely on the hardware security provisions of the cryptocore, to keep the state of the protocol safe between protocol steps (see Section 6.8).

The protocol comprises two phases, Initialisation and Ratchet Step, described in the following sections.

6.5.1 Initialisation

Explicit authority approval as part of the initialisation of the Ratchet is a basic feature of the proposed protocols. This feature ensures that after the failure threshold is exceeded the nodes stop the authentication loop until the appropriate AD is involved. Since the AD is removed from the field after the deployment of the nodes, requiring its involvement greatly decreases the likelihood of a security event being undetected as an adversary does not have access to the AD. In other words, the AD is in this case the second factor of authentication in addition to the node secrets.

Protocol 6.2 (Ratchet Authorisation)**.** *Nodes A and B have been enrolled into a neighbourhood with AD X. At the end of the protocol, both nodes have established a shared authorisation token that will be used in the remainder of the Initialisation phase. See Fig. 6.5 for the detailed interactions.*

1. *X generates a random authorisation token T_{AB} and encrypts it with the public keys of A and B, respectively.*
2. *X sends the public key of B to A.*
3. *A verifies that B is a peer and aborts on failure.*
4. *A generates a random nonce and sends to X.*

5. *X produces a signed authorisation request in the form of* $Q_{XA} = SIG_{S_X}(T_{AB} \| T_A \| P_A \| P_B)$.

6. *X sends the authorisation request to A.*

7. *A verifies the signature and aborts on failure.*

8. *A replies with an acknowledgement.*

9. *A similar process is repeated between X and B.*

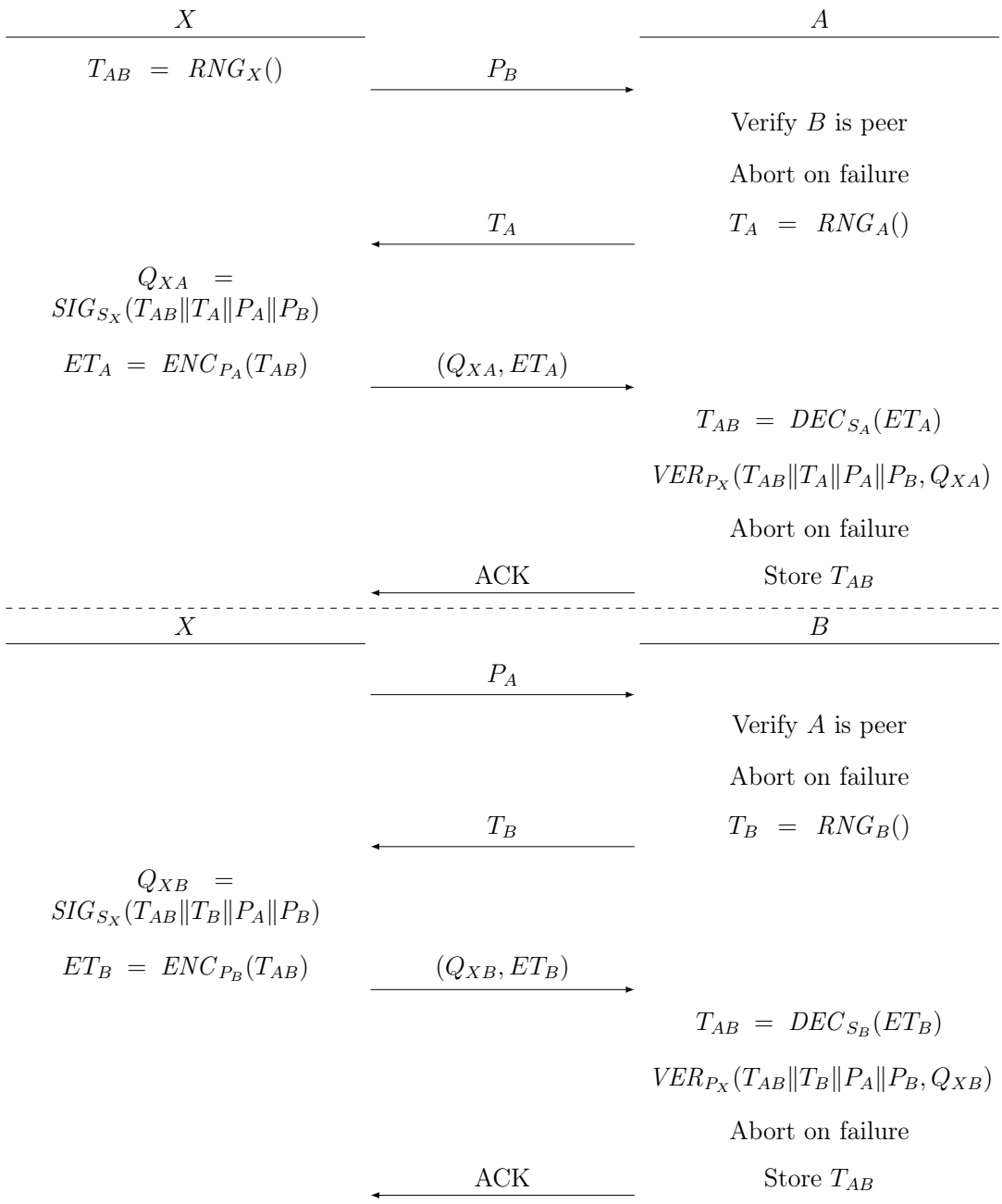


Figure 6.5: Ratchet Authorisation

The above Authorisation enables the 'bootstrapping' part of the Initialisation which introduces the nodes to each other in preparation for the periodic ratchet phase. It also allows the corresponding AD to approve the initialisation of the protocol. Thus, this

phase is required in two cases: (a) when the nodes are first introduced and (b) after the threshold of acceptable authentication failures n_f has been exceeded. While, seemingly, the second case negatively affects the robustness of the system, it is a deliberate design decision that satisfies the requirement for the protocol to cease normal operation in case of authentication failure (see Section 6.2).

Since the nodes have not been introduced yet, a temporary secure channel is established by means of public cryptography (with public keys exchanged previously). The requirement for a secure channel can be eliminated if the protocol is initialised in a trusted environment instead. In our framework, given that both nodes belong to the same neighbourhood, they make use of the provisions of the ADS (see Chapter 5) to exchange public keys and use them to sign their first CRPs. The Initialisation Phase takes place as follows:

Protocol 6.3 (CRP Ratchet Initialisation)**.** *Nodes A and B have been enrolled into a neighbourhood with AD X. At the end of the protocol, both nodes have established a state that will be used in subsequent Ratchet Step phases. See Fig. 6.6 for the detailed interactions.*

1. *A initiates the initialisation by authenticating B as described in Protocol 5.7.*
2. *A generates a random PUF challenge C_B^0 and signs it, including the authorisation token.*
3. *A sends the challenge and its signature to B.*
4. *B verifies the signature and aborts on failure.*
5. *B uses the challenge to produce a PUF response $R_B^0 = PUF_B(C_B^0)$.*
6. *B encrypts the response with the public key of A.*
7. *B generates a random PUF challenge C_A^0 and signs $C_A^0 \| R_B^0$ including the authorisation token.*
8. *B sends the encrypted response, the challenge, and the signature to A.*
9. *A verifies the signature and aborts on failure.*
10. *A decrypts the response of B and derives $K_A^0 = R_B^0 \oplus C_A^0$.*
11. *A generates a PUF response with the received challenge C_A^0: $R_A^0 = PUF_A(C_A^0)$.*
12. *A encrypts the PUF response and signs it, including the authorisation token.*
13. *A sends the encrypted response and the signature to B.*
14. *B verifies the signature and aborts on failure.*

15. *B decrypts the response of A and derives* $K_B^0 = R_A^0 \oplus C_B^0$.

16. *B stores the initial state* K_B^0 *and* C_A^0 *indexed by* P_A *and replies with an acknowledgement.*

17. *A stores the initial state* K_A^0 *and* C_B^0 *indexed by* P_B.

18. *Both nodes reset their initialisation and authentication counters.*

19. *Both nodes delete the authorisation token. They also discard any intermediate information used in this phase.*

6.5.2 Ratchet Step

The Ratchet Step phase serves a dual purpose: authenticating the remote node, and refreshing the secrets used for authentication. This phase is repeated continuously with the step interval t_s defined above. In the context of this phase, the nodes make use of one key each, which we refer to as 'ratchet key' and a common key which we refer to as 'round key'. The Ratchet Step phase for round $j, j \in \mathbb{Z}^+$ takes place as follows:

Protocol 6.4 (CRP Ratchet Step)**.** *Nodes A and B, performing the j-th ratchet step with A as the initiator. At the end of this phase, the nodes have authenticated each other and exchanged the necessary information to enable the next iteration. See Fig. 6.7 for the detailed interactions.*

1. *Node A has the state information* K_A^{j-1} *and* C_B^{j-1}. *Node B has the state information* K_B^{j-1} *and* C_A^{j-1}.

2. *If the authentication threshold has been exceeded A initiates Protocol 5.7. Upon successful authentication both nodes reset their authentication counters.*

3. *A generates a random PUF challenge* C_B^j *and calculates the HMAC tag of* $C_B^j \| C_B^{j-1}$ *with the ratchet key* K_A^{j-1}.

4. *A sends the challenges and the tag to B.*

5. *B derives the ratchet key of A as* $K_A^{j-1} = PUF_B(C_B^{j-1}) \oplus C_A^{j-1}$ *and validates the received HMAC tag. If there is a mismatch, B aborts and increments its failure counter. If the failure threshold has been exceeded, the failure procedure is followed.*

6. *B derives the round key* $K^j = K_A^{j-1} \oplus K_B^{j-1}$.

7. *B generates the PUF response* $R_B^j = PUF_B(C_B^j)$ *and a random PUF challenge* C_A^j.

8. *B encrypts the PUF response* $ER_B^j = R_B^j \oplus K^j$ *and calculates the HMAC tag of* $C_A^{j-1} \| C_A^j \| ER_B^j$ *with the round key.*

9. *B sends the two challenges, the encrypted PUF response and the HMAC tag to A.*

10. *A derives the ratchet key of B as* $K_B^{j-1} = PUF_A(C_A^{j-1}) \oplus C_B^{j-1}$ *and the round key* $K^j = K_A^{j-1} \oplus K_B^{j-1}$ *and decrypts the PUF response of B.*

11. *A validates the HMAC tag. If there is a mismatch, A aborts and increments its failure counter. If the failure threshold has been exceeded, the failure procedure is followed.*

12. *A generates the PUF response* $R_A^j = PUF_A(C_A^j)$*, encrypts it with the round key and calculates the HMAC tag of the PUF response with the round key.*

13. *A sends the encrypted response and the tag to B.*

14. *B decrypts the response and validates the HMAC tag. If there is a mismatch, B aborts and increments its failure counter. If the failure threshold has been exceeded, the failure procedure is followed.*

15. *B replies with an acknowledgement and stores* $K_B^j = R_A^j \oplus C_B^j$ *and* C_A^j *indexed by* P_A*.*

16. *A stores* $K_A^j = R_B^j \oplus C_A^j$ *and* C_B^j *indexed by* P_B*.*

17. *Both nodes advance their authentication counters and reset their failure counters.*

18. *Both nodes delete the ratchet keys and challenges from the previous round. They also discard any intermediate information used in this phase.*

6.6 Zero Knowledge CRP Ratchet

The main drawback of the CRP Ratchet is that PUF responses, or rather their hashes, are exposed to the verifying nodes. In order to alleviate this issue we need to rely on the more complex cryptographic primitive of *zero knowledge proofs(ZKP)*. Discussed in Section 3.3, zero knowledge proofs allow the prover to demonstrate the possession of some information without ever revealing that information. In this section, we combine zero knowledge proofs with PUFs into a *Zero Knowledge(ZK) CRP Ratchet*, inspired in part by the work of Kerr et al. in [16].

The ZK variant is designed to operate in threat models where nodes are not fully trusted. In those models, nodes would be able to masquerade as their peers if they had access to the PUF responses (or their hashes as was the case in the 'plain' CRP Ratchet). In addition, if the hashed responses are not exposed directly, the requirements for a secure cryptocore can be relaxed to some extend, leading to lower hardware costs. Finally, the exhaustion rate of the PUF responses can be significantly reduced since the burden of proof is handled by the ZKP primitive and PUF responses can be repeated without enabling replay attacks.

However, these advantages come at a cost. Firstly, ZKPs require increased compu-

tational resources since they are based on primitives similar to public key cryptography. Furthermore, due to their ZK nature, these proofs require multiple interactions (and thus increased bandwidth) for the verifier to be convinced that the prover possesses the authenticating information.

6.6.1 Initialisation

The zero knowledge Initialisation phase is analogous to that of Section 6.5 with the PUF responses now replaced by commitment values derived from the responses. The AD authorises the initialisation via Protocol 6.2 in same manner as above. Additionally, the security parameters defined in Section 6.4 are also used and a temporary secure channel is again established during initialisation. The Initialisation protocol is outlined below:

Protocol 6.5 (ZK CRP Initialisation)**.** *Nodes A and B have been enrolled into a neighbourhood with AD X. At the end of the protocol, both nodes have established a state to be used in subsequent ZK Ratchet Step phases. See Fig. 6.8 for the detailed interactions.*

1. *A initiates the initialisation by authenticating B as described in Protocol 5.7.*
2. *A generates a random PUF challenge C_B^0 and signs it, including the authorisation token.*
3. *A sends the challenge and its signature to B.*
4. *B verifies the signature and aborts on failure.*
5. *B uses the challenge to produce a PUF response and the corresponding ZK commitment $V_B^0 = ZKC(PUF_B(C_B^0))$.*
6. *B generates a random PUF challenge C_A^0 and signs $C_A^0 \| V_B^0$ including the authorisation token. The challenge C_B^0 is also included to ensure freshness avoiding replay attacks.*
7. *B sends the challenge, the commitment, and the signature to A.*
8. *A verifies the signature and aborts on failure.*
9. *A generates a PUF response with the received challenge and calculates the corresponding ZK commitment $V_A^0 = ZKC(PUF_A(C_A^0))$.*
10. *A signs the commitment including the authorisation token and the challenges C_A^0, C_B^0.*
11. *A sends the commitment and the signature to B.*
12. *B verifies the signature and aborts on failure.*
13. *B stores the initial state C_A^0 and V_A^0 indexed by P_A and replies with an acknowledgement.*

14. *A stores the initial state C_B^0 and V_B^0 indexed by P_B.*

15. *Both nodes reset their initialisation and authentication counters.*

16. *Both nodes delete the authorisation token. They also discard any intermediate information used in this phase.*

6.6.2 Ratchet Step

In the zero knowledge Ratchet Step phase of the protocol, the nodes mutually authenticate each other by exchanging challenges and the corresponding zero knowledge proofs. This exchange takes place λ times where $\lambda \in \mathbb{Z}^+$ is a security parameter, and the probability of an adversary successfully guessing the correct ZK proofs decreases as λ increases.

After both nodes are satisfied regarding the authenticity of their peer, they exchange commitments to enable the next protocol round, and rounds are executed with a step interval t_s as above. The Ratchet Step phase for round $j, j \in \mathbb{Z}^+$ takes place as follows:

Protocol 6.6 (ZK CRP Ratchet Step)**.** *Nodes A and B, performing the j-th ratchet step with A as the initiator. At the end of this phase, the nodes have authenticated each other and exchanged the necessary information to enable the next round. See Fig. 6.9 for the detailed interactions.*

1. *Node A has the state information C_B^{j-1} and V_B^{j-1}. Node B has the state information C_A^{j-1} and V_A^{j-1}.*

2. *If the authentication threshold has been exceeded, A initiates Protocol 5.7. Upon successful authentication both nodes reset their authentication counters.*

3. *A generates a random ZK challenge and a random PUF challenge.*

4. *A sends the new challenges and the PUF challenge from the previous round C_B^{j-1} to B.*

5. *B regenerates the PUF response $R_B^{j-1} = PUF_B(C_B^{j-1})$ and the corresponding ZK proof using the received ZK challenge.*

6. *B generates a random ZK challenge and a random PUF challenge.*

7. *B sends the new challenges, the ZK proof, and the PUF challenge from the previous round C_A^{j-1} to A.*

8. *A verifies the received ZK proof. If the verification fails, A aborts and increments its failure counter. If the failure threshold has been exceeded, the failure procedure is followed.*

9. *A regenerates its own PUF response R_A^{j-1} and the corresponding ZK proof using the received ZK challenge.*

10. *A also generates a new random ZK challenge and sends it to B along with the ZK proof from the previous step.*

11. *B verifies the ZK proof and replies with its own proof.*

12. *A in turn verifies the ZK proof sent by B.*

13. *The nodes repeat steps 9 to 12 $\lambda - 2$ times for both of them to be satisfied of the identity of their peer.*

14. *In the λ-th repetition, B generates a PUF response with challenge C_B^j and the corresponding commitment $V_B^j = ZKC(PUF_B(C_B^j))$, in addition to the ZK proof and challenge described in the previous rounds.*

15. *B then sends the proof, the commitment, and their hash $HASH(Q_B^{j\lambda} \| V_B^j)$ to A.*

16. *A verifies the received hash value against the received commitment and challenge. It also verifies the received ZK proof. If the verification fails, A aborts and increments its failure counter. If the failure threshold has been exceeded, the failure procedure is followed.*

17. *A in turn generates the PUF response R_A^j and its commitment V_A^j, in addition to the ZK proof and challenge described in the previous rounds.*

18. *A sends the proof, the commitment, and their hash to B.*

19. *B verifies the received hash value against the received commitment and challenge. This hashing mechanism binds the values of the commitment and the proof, ensuring that modifications of the commitment require modifications of the proof which will lead to a verification failure in the next step.*

20. *B also verifies the received ZK proof. If either verification fails, B aborts and increments its failure counter. If the failure threshold has been exceeded, the failure procedure is followed.*

21. *B replies with an acknowledgement and stores C_A^j, V_A^j indexed by P_A.*

22. *A stores C_B^j, V_B^j indexed by P_B.*

23. *Both nodes advance their authentication counters and reset their failure counters.*

24. *Both nodes delete the challenges and commitments from the previous round. They also discard any intermediate information used in this phase.*

(Ratchet Authorisation with X, A, B. See Protocol 6.2)

A		B
	Mutual Authentication (Protocol 5.7) $\longleftrightarrow$	
$C_B^0 = RNG_A()$		
$Q_A^C = SIG_{S_A}(C_B^0 \| T_{AB})$		
	$(C_B^0, Q_A^C) \longrightarrow$	
		$VER_{P_A}(C_B^0 \| T_{AB}, Q_A^C)$
		Abort on failure
		$R_B^0 = PUF_B(C_B^0)$
		$ER_B^0 = ENC_{P_A}(R_B^0)$
		$C_A^0 = RNG_B()$
		$Q_B = SIG_{S_B}(C_A^0 \| R_B^0 \| T_{AB})$
	$\longleftarrow (ER_B^0, C_A^0, Q_B)$	
$R_B^0 = DEC_{S_A}(ER_B^0)$		
$VER_{P_B}(C_A^0 \| R_B^0 \| T_{AB}, Q_B)$		
Abort on failure		
$K_A^0 = R_B^0 \oplus C_A^0$		
$R_A^0 = PUF_A(C_A^0)$		
$ER_A^0 = ENC_{P_B}(R_A^0)$		
$Q_A^R = SIG_{S_A}(R_A^0 \| T_{AB})$	$(ER_A^0, Q_A^R) \longrightarrow$	
		$R_A^0 = DEC_{S_B}(ER_A^0)$
		$VER_{P_A}(R_A^0 \| T_{AB}, Q_A^R)$
		Abort on failure
		$K_B^0 = R_A^0 \oplus C_B^0$
	$\longleftarrow$ ACK	
Store (P_B, K_A^0, C_B^0)		Store (P_A, K_B^0, C_A^0)
$AC_{AB} = 0$		$AC_{AB} = 0$
$FC_{AB} = 0$		$FC_{AB} = 0$
Delete T_{AB}		Delete T_{AB}

Figure 6.6: CRP Ratchet Initialisation

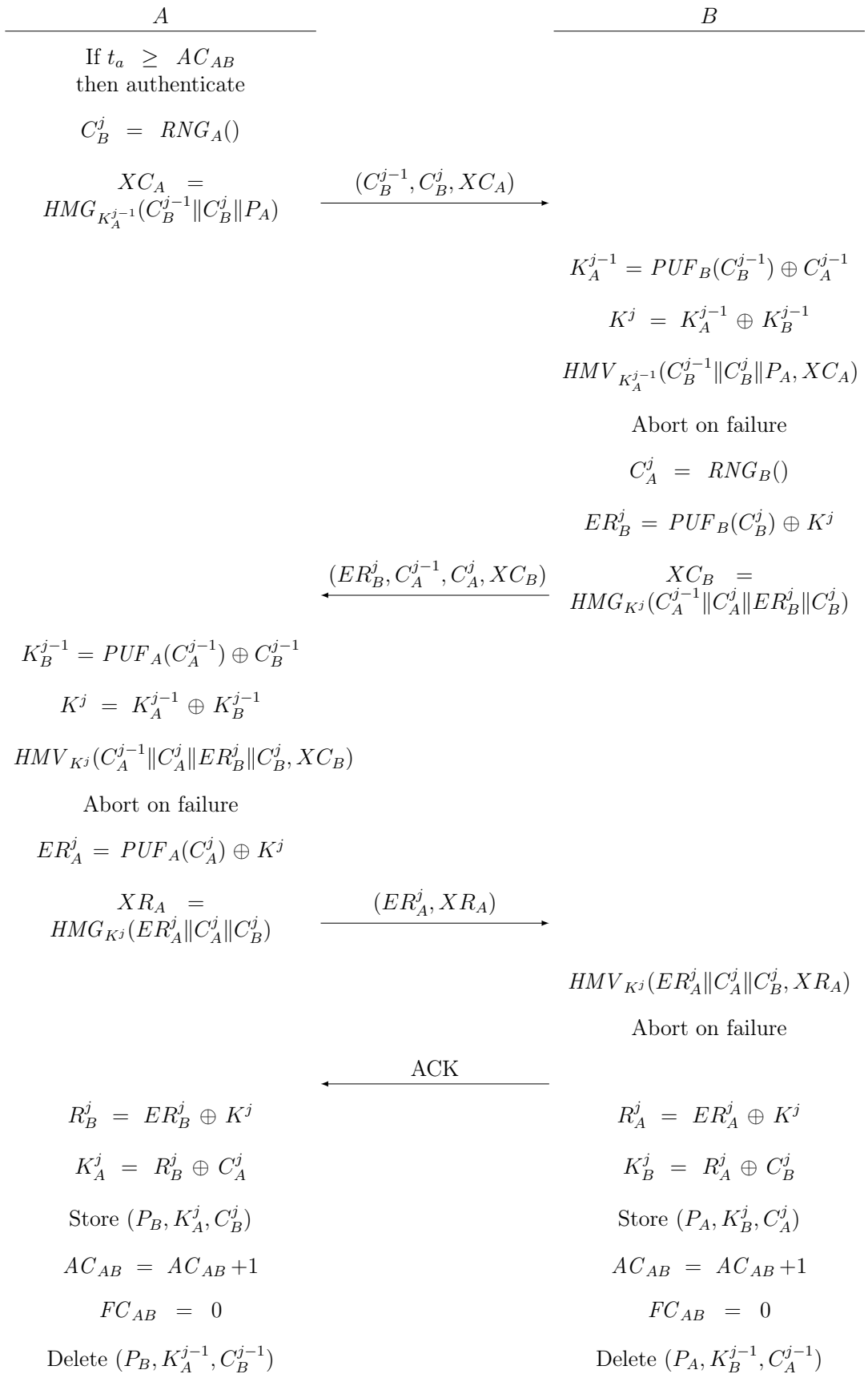


Figure 6.7: CRP Ratchet Step

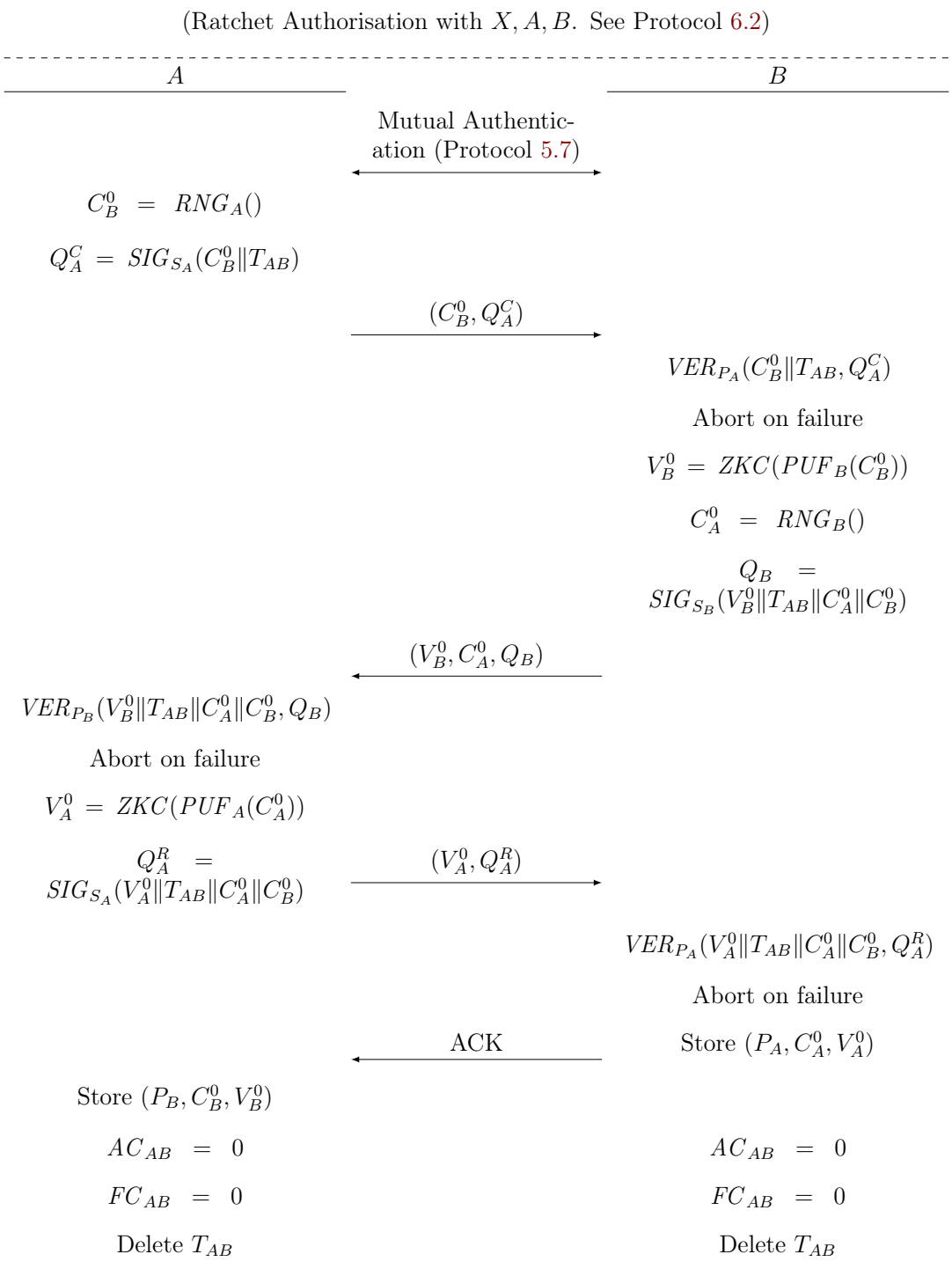


Figure 6.8: ZK CRP Ratchet: Initialisation

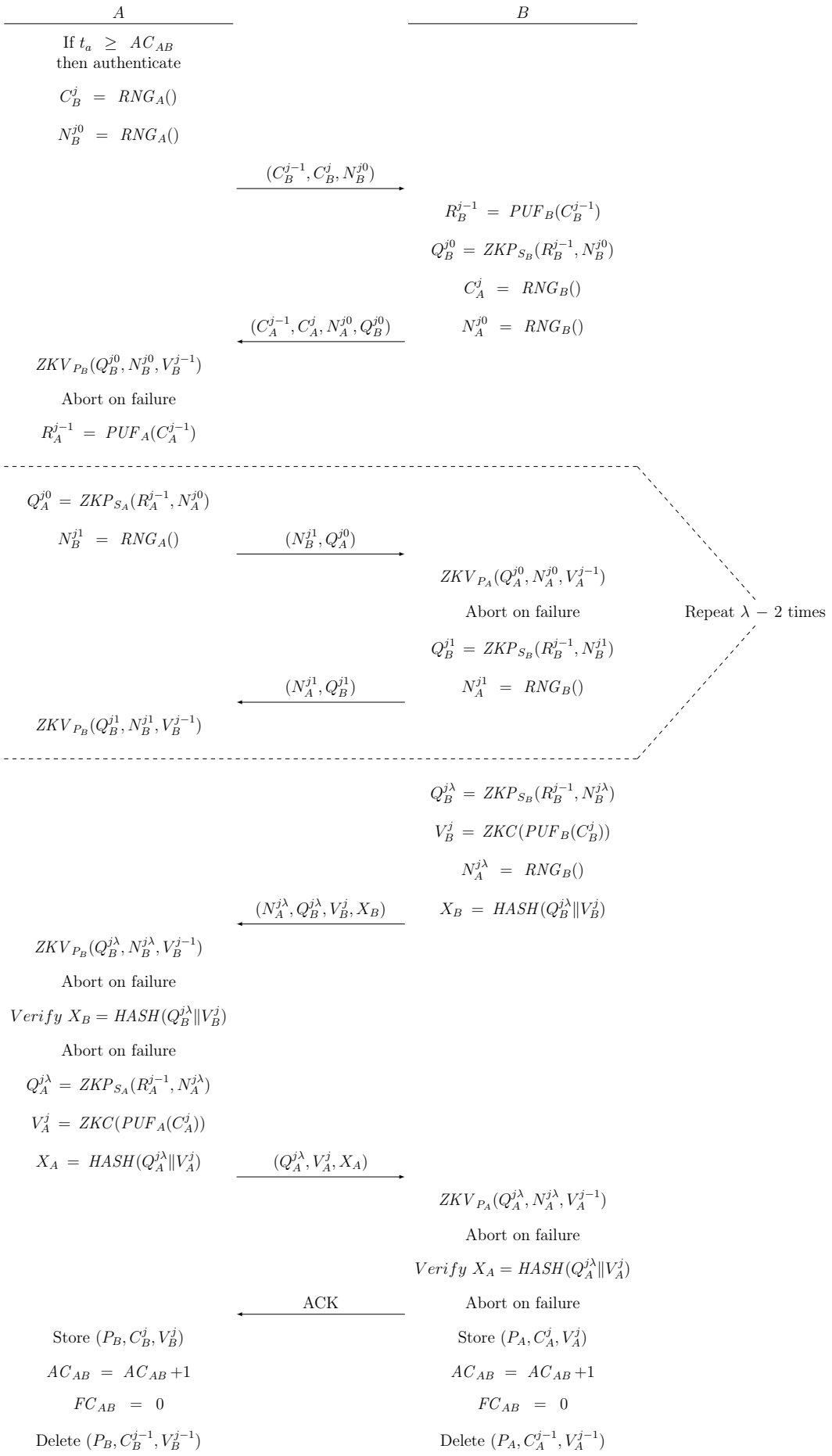


Figure 6.9: ZK CRP Ratchet: Ratchet Step

6.7 Performance Discussion

In this section we discuss the performance characteristics of the two ratchet variants. This discussion is informed by the software reference implementation presented in Chapter 9. Our analysis focuses on the Step phase of both variants since it occupies the majority of the protocol runtime.

Computation

The operations involved in a single 'plain' CRP Ratchet step are summarised in Table 6.4. Regardless of the hardware specifics, a single evaluation of the PUF is assumed to have negligible cost, as it often comprises simple 'read' operations. In addition, XOR and HMAC are efficiently computed in modern processors and can be further accelerated in hardware.

Protocol Operation	**Executions on A**	**Executions on B**
RNG	1	1
PUF	2	2
XOR	5	5
HMAC (generation or verification)	3	3

Table 6.4: CRP Ratchet operations (step phase)

On the contrary, the ZK CRP Ratchet involves more complex mathematical operations, similar to the ones used in public key cryptography. The Schnorr ZK proof[138] is often used in resource constrained environments, offering both a finite field and an elliptic curve variant. The use of ECC, in addition to the advantages discussed previously, provides the added benefit of being able to reuse the same implementation logic for encryption, signature, and ZK proofs. The operations involved in a single round of this variant are detailed in Table 6.5.

Protocol Operation	**Executions on A**	**Executions on B**
RNG	$\lambda + 1$	$\lambda + 1$
PUF	2	2
HASH	2	2
ZKP - modular multiplication over EC	λ	λ
ZKV - multiplication over EC	λ	λ
ZKC - scalar multiplication	1	1

Table 6.5: ZK CRP Ratchet operations (step phase)

The Initialisation phase and the occasional public key authentication for both variants make use of symmetric and asymmetric cryptography respectively, which have high

computational cost. For this reason, these phases are designed to be used infrequently and can also be accelerated in hardware.

Security Parameters

The t_s and n_f parameters also have an important effect on the overhead of the proposed protocol. The failure threshold n_f needs to account for application-dependent issues including network latency, dropped packets etc. Similarly, the minimum step interval t_s depends on two implementation variables: the maximum throughput and the maximum number of PUF responses. The latter is discussed in Section 6.8. The throughput of the implementation refers to the rate at which the hardware is capable of performing the necessary operations. As discussed above, the operations involved in the CRP Ratchet are lightweight and can be efficiently implemented in hardware, thus the throughput is a theoretical rather than a practical limitation to the minimum step interval. However, the ZK Ratchet requires more complex operations which can lead to reduced throughput.

On the other hand, the maximum step interval is determined by the human factor, as the Step phase is required to be repeated at a rate sufficient to prevent an adversary from accessing and modifying the participating nodes. Thus, t_s can range from milliseconds to a few seconds or even minutes, depending on the particular deployment, greatly reducing the overhead without necessarily harming the overall security of the system.

In the case of the ZK variant, an additional parameter λ is used. Since λ directly affects both the number of interactions and the number of operations required, the energy and bandwidth overhead are directly proportional to λ. However, as discussed in Section 6.8 λ is critical for the security of the protocol and is required in most cases to remain larger than 5, signifying five ZK proof interactions per ZK ratchet round.

Storage Requirements

The storage space required by either variant remains constant with the number of authentications and scales linearly with the number of peer nodes.

Both variants store device identifiers to differentiate among peers. Similarly to the ADS, two bytes are sufficient to identify up to 65536 peers, which is adequate for most practical applications. Although not included in the PUF model used in this chapter, helper data is also stored to enable the correction of bit errors in the PUF responses. The size of these data can amount to up to 100 bytes per response, depending on the correction methods. However, since our protocols use a single PUF response per round, only one such piece of data is required to be kept at any given time. In addition, Chapter 8 includes practical methods for reducing the BER of SRAM PUFs leading to smaller helper data sizes.

For the first variant, only PUF CRPs are effectively stored, in different forms. Challenges and responses have the same length as discussed in Section 6.8 and thus, for a response length of l bits and a number of peers n, a node is required to store a total of $s = 3ln$ bits of data. The final PUF responses are the result of a hash function, with

a typical digest length in the range of 128 to 512 bits, thus 384 $bits \leq s \leq 1536$ $bits$ (48 $bytes \leq s \leq 192$ $bytes$)[117]–[120]. For currently available PUF constructions, 64 bits of output are sufficient for the entirety of the raw PUF response space and thus higher length values are normally not necessary.

The ZK variant storage requirements are harder to estimate. Nodes in this case store pairs of PUF challenges and ZK commitments. While the former have the length mentioned above, the size of the latter varies with the implementation. For the ECC Schnorr ZK identification used in our reference implementation (Chapter 9), the commitments are simply curve points represented with 32 bytes for each coordinate, or 64 bytes in total.

Message Length

The size of the exchanged messages depends on the length of the information discussed above, with the addition of HMAC or hash digests used for verification. In the reference implementation, the maximum message size is 128 bytes for the first variant and 160 bytes for the ZK variant, with a mean message size of 101 bytes for both variants. Table 9.2 provides an overview of the message size for every protocol interaction.

6.8 Security Analysis

This section includes an analysis of the security of the proposed protocols, via several lemmas and theorems. Once again, we assume a single PPT adversary, referred to as $\mathcal{A}$ with capabilities discussed in Section 6.2.2.

Similarly to Chapter 5 we use the taxonomies of protocol attacks from [20] and [112] as a starting point and extend them to include the physical security and unclonability aspects. Since the goal of the protocols is to establish unclonable links, our analysis focuses primarily on the (in)ability of the adversary to impersonate nodes. A secondary but crucial goal is the protection of the PUF secrets since they lie at the root of our security paradigm. The majority of our analysis is shared by both ratchet variants and thus we specifically point out issues applying to one of the variants only when needed, to avoid repetition. For the same reason, the reader is referred to Lemmas 5.2 and 5.3 which also apply to the protocols of this chapter.

The following analysis proves the *correctness* and *completeness* of both protocol variants. Additionally, it is shown that $\mathcal{A}$ is confined to brute-force attacks which can be prevented by means of sufficiently long secrets.

Lemma 6.1 (Cryptographic Primitive Security)**.** *The security of the cryptographic primitives used is guaranteed against $\mathcal{A}$, with high probability.*

Proof. According to the adversary model of Section 6.2.2, $\mathcal{A}$ is unable to directly bypass the security of the involved cryptographic primitives without compromising the corresponding keys. Additionally, the proposed protocols do not undermine the security of the aforementioned primitives (e.g. by exposing parts of the keys) and thus do not facilitate cryptanalysis efforts. □

Lemma 6.2 (XOR Encryption Security)**.** *$\mathcal{A}$ is not able to recover the content of XOR encrypted messages, except with negligible probability.*

Proof. In the first ratchet variant, the encryption of the PUF responses is based on XORing them with the round key. Therefore, to ensure the security of this method, the PUF challenges and round keys are required to have the same length. Since the round keys are in turn a combination of PUF challenges and responses, essentially the requirement is that PUF challenges and PUF responses have the same length. In addition, both the keys (round keys) and the plain texts (PUF responses) are random and only used once, preventing *chosen plaintext attacks*. Thus, we conclude that $\mathcal{A}$ is limited to guessing the round or ratchet keys. In the reference implementation, 32 bit long PUF responses are used, leading to a probability for correctly guessing the keys of 2^{-32}. □

Lemma 6.3 (Real or Random Secrecy)**.** *Message contents appear random to an eavesdropping $\mathcal{A}$.*

Proof. The data contained in messages exchanged by both variants can be classified in three categories: (a) PUF and ZK challenges, (b) PUF responses, ZK responses, and ZK commitments, and (c) digests of combinations of the first two categories. In the case of (a), PUF challenges are random. Data of categories (b) and (c) are derived from data of the first category and are thus also random. Additionally, the PUF challenges are only transmitted once. Thus we conclude that $\mathcal{A}$ has no ability to differentiate between random data and real messages. □

Lemma 6.4 (Replay Attacks)**.** *$\mathcal{A}$ is unable to successfully authenticate via replaying previously captured messages, except for a negligible probability.*

Proof. Thanks to the random challenge involved in the ZK proofs, replay attacks are prevented, given that the random challenges are not repeated. Similarly, in the 'plain' CRP Ratchet, randomly generated PUF challenges are included in the messages. In addition, round keys are derived from a combination of the state of both participants (which is continuously refreshed) and are used as the HMAC secret. This directly prevents replay attacks since the HMAC verification for older messages will fail and $\mathcal{A}$ cannot bypass the security of HMAC due to Lemma 6.1. In the communication with the AD during the Initialisation phase, a monotonic counter is used to prevent similar attacks. □

Lemma 6.5 (Authentication)**.** *In the CRP Ratchet, a legitimate node can successfully authenticate, and a malicious node cannot authenticate, except for negligible probability.*

Proof. In the 'plain' CRP Ratchet, the authentication is performed by means of HMAC tags using keys derived from PUF CRPs. These keys are inherently protected by the properties of PUFs and the cryptocore, and they are refreshed in every round. In addition, by Lemma 6.1 the security of the HMAC primitive is guaranteed.

Thus $\mathcal{A}$ is limited to guessing the HMAC keys which have a length equal to the length of the PUF responses. For example, for a response length of 32 bits, $\mathcal{A}$ has a probability

of 2^{-32} of guessing the correct key and successfully authenticating, and has to repeat this guess in every round due to Lemma 6.8.

On the contrary, the probability of a legitimate node failing to authenticate amounts to the probability of error in either the PUF response regeneration or the network transmission. Both issues are covered by the assumptions outlined in Section 6.4. □

Lemma 6.6 (Zero Knowledge Authentication)**.** *In the ZK CRP Ratchet, a legitimate node can successfully authenticate, and a malicious node cannot authenticate, except for negligible probability.*

Proof. In the ZK protocol variant, the probability of $\mathcal{A}$ successfully guessing the correct proof value for a given challenge is affected by the number of proof rounds λ and the length of the PUF responses. As an example, our reference implementation uses $\lambda = 5$ and 32-bit PUF responses, resulting in a probability of a correct guess 2^{-37}. Evidently, much shorter PUF responses can be used if the number of repetitions is increased, partially alleviating the issue of response exhaustion in exchange for higher overhead.

For the authentication of legitimate nodes, the same as Lemma 6.5 applies. □

Lemma 6.7 (Authority Action)**.** *After a failure, the ratchets can only be restarted with an 'authority action' by the appropriate AD.*

Proof. The Initialisation phase of both variants involves the AD responsible for the neighbourhood of the nodes and this phase is required after protocol failures. Thus, the proposed protocols can only be (re)started with the appropriate authority device. □

Lemma 6.8 (Break-in Recovery/Post-compromise Security)**.** *Both variants are able to recover from temporary node compromise.*

Proof. Every protocol exchange of the CRP Ratchet is signed and/or encrypted with a ratchet key or a round key. Furthermore, these keys are derived locally from the PUFs and are protected from invasive attacks since raw PUF responses never leave the cryptocore.

More specifically, the round key of round j is derived as $K^j = K_A^{j-1} \oplus K_B^{j-1} = R_B^{j-1} \oplus C_A^{j-1} \oplus R_A^{j-1} \oplus C_B^{j-1}$. Assuming that $\mathcal{A}$ has captured C_A^{j-1} and C_B^{j-1} when they were transmitted, she needs to recover both R_A^{j-1} and R_B^{j-1} to compromise K^j. Since neither of the nodes stores this information, physically compromising one of the nodes does not expose the round key and $\mathcal{A}$ would need to compromise both nodes, essentially rendering the whole authentication protocol futile. This feature is also assisted by the cryptocore which performs all the necessary operations internally, protecting the raw PUF responses.

It should also be noted that part of the ratchet key of each node is a PUF response of *its peer*. This is an important detail of the protocol, since it exploits the unpredictability of PUFs, given that the security of the HMAC primitive is guaranteed by Lemma 6.1. For example, in the first interaction, if $\mathcal{A}$ compromises node A she will be able to construct an HMAC tag with the ratchet key of A. However, B will derive the ratchet key of A from the contents of the received message and since $\mathcal{A}$ cannot predict PUF responses, she

is unable to generate an appropriate challenge that would lead to the appropriate ratchet key needed to verify the fake HMAC tag.

Due to the above properties, an adversary who compromises a node for a limited time does not gain access to future protocol secrets (since those secrets can only be derived via the appropriate PUFs). One protocol round is sufficient to refresh the secrets after the adversary's access has been removed. Therefore, we can conclude that the CRP Ratchet provides *break-in recovery* or *'post-compromise security'*[136].

The ZK Ratchet provides similar recovery, in a different manner: PUF responses do not leave the generating nodes and their possession is proven with an interactive challenge-response sub-protocol. There exist thus no secrets that can be compromised and leveraged to break future protocol rounds. □

Lemma 6.9 (Message Modification/Man-in-the-Middle Attacks)**.** *Modifications to protocol messages by $\mathcal{A}$ are detected with high probability.*

Proof. The first ratchet variant utilises HMAC tags which ensure the integrity of the message contents. All messages exchanged during the Step phase contain the corresponding tag which can be calculated only by entities in possession of the corresponding secret. During the Initialisation phase, public key signatures are employed to provide message integrity.

Similarly, the ZK variant uses public key signatures during Initialisation and hashes during the Step phase. $\mathcal{A}$ would only aim to tamper with the messages containing a ZK commitment, to enable her to compromise the next authentication round. Thus, $\mathcal{A}$ would attempt to modify the commitment with her own value. However, a modification to the commitment would invalidate the message hash. Similarly, generating a modified digest would require an alteration of the ZK proof, leading to authentication failure.

Therefore, since $\mathcal{A}$ is unable to bypass the above cryptographic primitives by Lemma 6.1, we conclude that $\mathcal{A}$ is unable to modify messages or perform man-in-the-middle attacks. □

Lemma 6.10 (PUF Modelling)**.** *The proposed protocols do not allow modelling attempts against the PUFs used in the protocols.*

Proof. As seen in Lemma 5.4, due to the properties of the cryptocore and the PUFs themselves, an adversary is unable to predict future responses from past CRPs. Nevertheless, certain PUF classes are susceptible to modelling attacks via machine learning methods based on a large number of CRPs. For this reason, the proposed protocols aim to minimise the exposure of PUF responses.

It is clear that PUF responses are never exchanged during the ZK CRP Ratchet. In the 'plain' CRP Ratchet, the interactions are designed to authenticate the remote node before sending over any PUF responses. Furthermore, the inclusion of an HMAC tag in every message prevents adversaries from trying all the PUF challenges until the receiving node

eventually produces the right ratchet key. Of course, with a sufficient challenge length the probability of a successful guess is negligible, regardless of the HMAC protection. □

Lemma 6.11 (PUF CRP Confidentiality)**.** *$\mathcal{A}$ is not able to recover the CRPs corresponding to any node, except with negligible probability.*

Proof. In the first variant, PUF responses are fully protected with the XOR method and responses are only decrypted inside the cryptocore which is assumed secure. Thus, $\mathcal{A}$ can only access PUF responses with the same probability of reversing the XOR encryption (see Lemma 6.2). For the ZK variant, raw PUF responses never leave the cryptocore of their generating node since ZK commitments are used in their place, and the security of the ZK proofs is guaranteed by Lemma 6.1. □

Lemma 6.12 (Topology Distortion Detection)**.** *The replacement or complete removal of a node (a topology distortion) is detected with high probability.*

Proof. As discussed in Section 6.2.1, a node can either be removed temporarily, removed permanently, or replaced by a different device altogether. By Theorem 6.1, in the replacement case the authentication will fail since the new device will not possess the necessary authentication information.

Temporary or permanent node failures on the other hand, are covered by the failure threshold parameter n_f, which ensures that a number of failed authentication attempts are allowed, to take into account possible communication errors. If the threshold is exceeded, the unresponsive node is assumed to be missing.

Thus, we conclude that with a carefully chosen failure threshold, *topology distortions* are detected with high probability. □

Theorem 6.1 (Node Unclonability)**.** *The use of PUFs in every round proves their continued possession, creating unclonable nodes.*

Proof. By Lemmas 6.5 and 6.6 it is evident that the PUF of each node is a fundamental element of each protocol interaction, for both variants. This feature, which is one of the major goals for the proposed protocols, essentially creates unclonable nodes since adversaries are unable to derive the protocol secrets, due to the properties of PUFs.

It is vital however for the PUF responses to have a certain length that will prohibit brute-force attacks and allow the proposed protocols to operate for extended periods of time without considerable repetition. In most practical applications a response length that is over 64 bits is sufficient. For example, a 64-bit response and a step interval of 100ms would support over *26 years* of continued protocol operation without repetition of CRPs; a time period which exceeds the typical lifetime of modern electronic systems. Still, in practice, certain PUF classes are unable to provide responses of the required length, or exhibit a high correlation between bits of different responses, and thus additional methods are needed to improve the number and entropy of the available responses.

Nevertheless, neither of the protocols is directly impaired by a repetition of the PUF CRPs, thanks to their chaining nature. The only case where CRP reuse presents an issue is when every single CRP is used in the same order. Thus, assuming that PUF challenges are never generated in the same order, PUFs with a somewhat limited CRP space do not negatively affect the security of the protocols.

Of course, t_s can be chosen with an aim at decreasing the depletion rate of PUF responses. However, the ratcheting feature of the protocols is only valuable when the Step phase is executed with a frequency sufficient to ensure the containment of potential attacks through prompt detection. The definition of 'sufficient' frequency is based on the application and the assumed capabilities of potential adversaries. □

Theorem 6.2 (PUF Confidentiality)**.** *The proposed protocols ensure the confidentiality of the PUF secrets of the participating entities.*

Proof. By Lemma 6.11 the raw PUF responses are protected in both protocol variants. Additionally, the modelling of the PUF behaviour is prevented by virtue of Lemma 6.10. Finally, when PUF responses are shared (in the first variant) they are never stored in clear text and do not leave the cryptocore. Hence we conclude that the confidentiality of the PUF secrets is safeguarded. □

Theorem 6.3 (Correctness)**.** *The proposed protocols are* correct*.*

Proof. The *correctness* property of the proposed protocols is proven by Lemma 6.5 and Lemma 6.6 which show that both protocols provide authentication for legitimate nodes and disallow authentication for malicious entities. □

Theorem 6.4 (Link Unclonability)**.** *The proposed protocols achieve their goals of providing mutual PUF-based authentication and creating unclonable links.*

Proof. By Theorems 6.1 and 6.2 and Lemmas 5.2 and 5.3 adversaries are unable to clone legitimate devices after the proposed protocols have been initialised, and nodes involved in the protocols cannot be removed or replaced due to Lemma 6.12.

Additionally, the proposed protocols provide continuous authentication due to Theorem 6.3 and are able to withstand a number of common attacks as proven in Lemmas 6.2 to 6.4, 6.8 and 6.9. Finally, by Theorem 6.2 the proposed protocols preserve the confidentiality of the underlying PUF secrets, hence ensuring future protocol security.

Thus, we conclude that the proposed protocols achieve the security goals outlined in Section 6.2 and provide methods for the establishment of unclonable links. □

6.9 Formal Verification

The protocols introduced in this chapter were also formally verified with ProVerif. In Table 6.6 we summarise the models of the basic cryptographic operations used in our

Operation	Implementation	Model Source
PUF	Symmetric cryptography with key which is never shared.	-
RNG	Natively supported (random tokens).	ProVerif user manual.
HASH	Natively supported.	ProVerif user manual.
HMAC	Implemented with native functions.	[139]
XOR	Implemented with native functions.	[140]
CONCAT	Implemented with native functions.	-
SIG	Natively supported.	ProVerif user manual.
VER	Natively supported.	ProVerif user manual.
ENC	Natively supported.	ProVerif user manual.
DEC	Natively supported.	ProVerif user manual.
ZKC	Not modelled, unsupported by ProVerif.	-
ZKP	Not modelled, unsupported by ProVerif.	-
ZKV	Not modelled, unsupported by ProVerif.	-

Table 6.6: Basic operations in ProVerif

protocols, where it is evident that the majority of operations are modelled in an identical manner as in Chapter 5.

Similarly to Section 5.5, we used correspondence assertions and secrecy queries to verify the security properties of the proposed protocols, as illustrated in Table 6.7. Unfortunately, ProVerif does not provide native support for zero knowledge assertions due to the nature of the applied pi calculus. The same is true for alternative software solutions that are currently available and thus we were unable to verify the zero knowledge variant of the CRP Ratchet. Nevertheless, formal verification of this variant can be achieved in future work, since there exist efforts to extend the modelling framework with a 'zero knowledge compiler'[141], [142].

Protocol	Security Properties	Verified With
CRP Ratchet Authorisation	Secrecy of T_{AB}.	Secrecy query.
	Authentication of authorisation request.	Correspondence assertion.
CRP Ratchet Initialisation	Secrecy of T_{AB}.	Secrecy query.
CRP Ratchet Step	Authentication of A to B.	Correspondence assertion.
	Authentication of B to A.	Correspondence assertion.

Table 6.7: Security properties as captured in ProVerif

6.10 Conclusion

In this chapter, we presented two protocol alternatives for mutual authentication in peer-to-peer scenarios, based on the periodic exchange PUF secrets. We described the protocol

operations and discussed their suitability for different applications, their practicality, as well as their security through a comprehensive analysis.

The main limitation of the methods proposed in this chapter is the assumption of ideal PUFs. This is a reasonable assumption for the reasons discussed in Section 4.3.2, especially in the context of this chapter. In reality, PUF ICs exhibit non-zero bit error rates (BER), requiring additional logic and storage for error correction, and adding overhead to any PUF-based protocol. In Chapter 8 we discuss methods for reducing the effect of bit errors in such protocols, with the help of experimental data of SRAM PUF behaviour.

Despite the protocols' allowance for CRP reuse, the PUF response space remains finite, limiting the number of protocol interactions that can be performed before CRPs are repeated. While so-called 'strong PUFs' can improve this situation, they have other disadvantages including susceptibility to modelling. For the 'weak PUFs', solutions exist on both protocol and hardware level. In hardware, reconfigurable PUFs[90] allow for one-way reconfiguration of their logic when their responses are exhausted. PUFs without this feature can be combined with block ciphers to expand the number of available responses[143]. Protocol level countermeasures exploiting the inherent instability of PUFs are included in Chapter 8. Several of the methods can be extended to other PUF classes besides SRAM.

Finally, authentication protocols of any kind, including the proposed ones, do not provide any guarantees for the actual actions of the authenticated entities. In other words, a node can be under the control of an adversary but still successfully authenticate. However, due to the node architecture, the adversary cannot access cryptographic or other secrets without invasive attacks to the hardware which are likely to destroy the PUF secrets and halt the authentication process. In addition, the unclonable links established by the ratchet protocols ensure that the participating nodes are not removed from the field and can thus be inspected periodically by human operators. In future iterations, the ratchets can also include distance bounding methods for verifying the physical distance of the nodes, some of which have already been proposed in literature[144].

Part III

Practical Considerations

7. Cryptographic Core

7.1 Introduction

The protocols and methods described in Chapters 5 and 6 have varying aims but are based on the same set of cryptographic primitives. For the correct, efficient and, most importantly, secure operation of the protocols, these underlying primitives are best served by a hardware construction that we call a cryptographic core or 'cryptocore'. Since the aim of this chapter is to sketch a practical architecture, we move away from information-theoretical requirements that are often discussed in literature as these requirements are rarely achievable, much less guaranteed in practice.

The main motivation behind a hardware implementation stems from the very nature of the PUFs and the increased protection that is required for their responses. To ensure that these responses are not exposed before being transformed by post-processing operations, it is important for the post-processing to be performed in hardware, leaving no interface for exposure. At the same time, a hardware implementation greatly reduces the attack surface against the PUF for two reasons: (a) invasive physical attacks are much harder to perform than software attacks, and (b) the effectiveness of security provisions is higher when it is applied before the protected information is dispersed through multiple paths, especially given the complexity of modern software.

In this chapter we present a reference architecture for the cryptocore highlighting its required features and the underlying rationale. The cryptocore comprises components which are commonly used in embedded systems and have been thoroughly studied and optimised; certain implementation details are therefore omitted. As such, the cryptocore can be implemented in an FPGA or as an ASIC depending on the application, as long as the overall architecture remains on the same IC, to allow for the advantages mentioned above. Additionally, physical security of said components can be provided, albeit with varying success, by one or several methods similar to the ones reviewed in Section 3.2.

Our design discussion is driven by the aim of creating a single IC which can be integrated or even 'plugged' into existing architectures and perform its prescribed purpose without exposing any of its internal secrets. To allow for a reduced attack surface and facilitate verification, the cryptocore is not (re)programmable and thus changes in underlying cryptographic primitives would require it to be replaced. Despite these restrictions, our reference architecture is modular and can be easily adapted or extended in light of future developments in cryptography or PUFs. In an effort to highlight the universal nature of the architecture, we leave our cryptographic implementation choices for Chapter 9.

7.2 Instruction Set

From a functional viewpoint, the cryptocore performs all the operations required by the higher layers of the unclonability stack. Based on the protocols of Chapters 5 and 6 we summarise the main operations below:

- PUF CRP generation: receiving a challenge and returning a response.
- Generation of cryptographic hashes.
- HMAC generation and verification.
- ECC public key operations: key pair generation, encryption, decryption, signature generation, signature verification, ZK commitment, ZK proof generation, ZK proof verification.
- Execution of the above operations as required by the protocols, in order to preserve the confidentiality of PUF responses. The cryptocore receives a PUF challenge and produces the post-processed (and thus obfuscated) PUF response.
- Random token generation.
- Volatile data storage: holding temporary data and intermediate values.
- Non-volatile data storage: the proposed protocols do not permanently store any secret information and thus this storage can be shared with the rest of the system.

It is evident that the above operations are particularly useful in the majority of embedded devices. As a result, it is easy to envision additional features and interfaces making use of the provided operations. However, as the features of the core multiply, so does its complexity and the corresponding attack surface. Therefore, the focus should be on creating a self-contained, highly restricted hardware block that accepts the instructions outlined in Table 7.1.

In the table, all the operations involving encryption, decryption, and signature are asymmetric cryptographic operations. When a private key is required, it is used implicitly and thus is not included in the input list. Additionally, the PUF instructions which receive a PUF challenge, generate the corresponding PUF response (with the required error correction) and apply the requested operation on the response before returning it. This design ensures that PUF responses are not exposed beyond the cryptocore boundary.

Operation	**Instruction**	**Inputs**	**Output**
Random number generation	RNG	-	Random data
Hash digest generation	HASH	Message	Hash digest
HMAC generation	HMG	Message, key	HMAC tag
HMAC verification	HMV	Message, key, HMAC tag	Pass/fail
Signature generation	PKSG	Message	Signature
Signature verification	PKSV	Message, signature, public key	Pass/fail
Encryption	PKE	Message, public key	Ciphertext
Decryption	PKD	Message	Plaintext
Key pair generation	PKG	Key seed	Public key
Decryption and XOR	PKDX	Ciphertext, XOR mask	XORed plaintext
Encryption	PUFE	PUF challenge, public key	Encrypted PUF response
XOR	PUFX	PUF challenge, XOR mask	XORed PUF response
Hash digest generation	PUFH	PUF challenge	Hashed PUF response
PK signature generation	PUFS	PUF challenge, padding data	PUF response signature
ZK commitment generation	PUFZC	PUF challenge	PUF response ZK commitment
ZK proof generation	PUFZP	PUF challenge, ZK challenge	PUF response ZK proof
ZK proof verification	PUFZV	Proof, ZK challenge, commitment, public key	Pass/fail

Table 7.1: Cryptocore instruction set

7.3 Architecture

The proposed protocols and the reference architecture are designed to make use of PUF features with the aim of minimising the need for physical security. Thus, the only data path that needs to be protected is the raw PUF output. As soon as the raw PUF output is protected with an additional operation (e.g. XORed with a secret key), the resulting data are considered public. The internal state of the PUF is assumed to be protected by the inherent properties of the PUF which make it impossible to perform physical attacks on it without destroying the underlying secret.

For the reasons discussed in the previous section, it is desirable for the cryptocore to reside on a separate IC than the rest of the system. In the following descriptions and diagrams, components performing the same function can appear multiple times. This is for the benefit of clarity, and practical implementations can opt for duplication or reuse of components as dictated by the relevant area and performance constraints. From a security standpoint, the concentration of the cryptocore's logic in a smaller silicon area also hinders physical attacks. Nevertheless, component reuse contributes to the creation of points of failure which would affect large portions of the cryptocore.

To begin with, Fig. 7.1 shows a high-level block diagram of the cryptocore architecture, serving as a roadmap for the following sections. The control unit receives instructions, decodes them and generates the appropriate control signals for the components. Upon completion of their operations, the components place the necessary output data on the internal data bus to be output to the rest of the system.

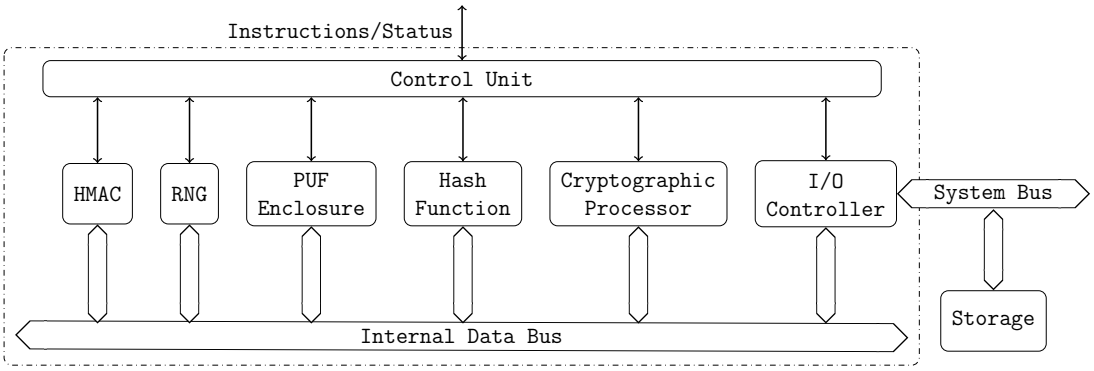


Figure 7.1: Block diagram of the cryptocore reference architecture

7.3.1 Communication and Input/Output

Component communication is achieved via an internal data bus which is kept isolated at all times from the system bus. When input or output is required, the I/O controller transfers the data to and from the internal data bus and consequently resumes bus isolation.

To enable effortless integration with existing systems, the I/O controller (in collaboration with the control unit) performs a memory mapping of the internal cryptocore components. As such, a software API for the cryptocore can be easily developed, interacting with the core by simply using the appropriate memory addresses.

7.3.2 Storage

The data kept in permanent storage comprises 'key challenges' (used as PUF input to (re)generate asymmetric key pairs), error correction helper data, and protocol state. As per the adversary models of Chapters 5 and 6, these data types are considered public and do not need to be protected. Hence, the storage can be shared with the rest of the system and the communication between the cryptocore and the storage device can be handled by the CPU, as would be the case for any other application.

Volatile storage required for the operation of the cryptocore components is implicitly present in the core architecture. The components can either share a core-wide storage element or include their own local registers. A common storage element would make efficient use of the available silicon area but it would also require additional logic for access instrumentation. Additionally, avoiding the dispersion of sensitive data over long data paths is advantageous for physical security provisions.

7.3.3 Hash-based Message Authentication Code

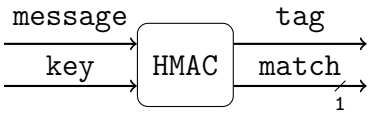
Hash-based Message Authentication Code (HMAC) is a widely used message authentication code, developed by Bellare et al.[145] to mitigate attacks against earlier MAC constructions. It is based on the combination of XOR and a cryptographic hash function, and thus can be implemented very efficiently and with minimal overhead compared to simpler but less secure MAC alternatives. For the cryptocore in particular, it can be implemented with a simple combination of the existing hash function and XOR components. In addition, as proven in later work[146], HMAC's only security requirement is for the underlying hash function to be a Pseudorandom Function Family (PRF). The block diagram of the cryptocore's HMAC component is shown in Fig. 7.2, where 'opad' and 'ipad' refer to outer padding and inner padding constants respectively.

7.3.4 Cryptographic Processor

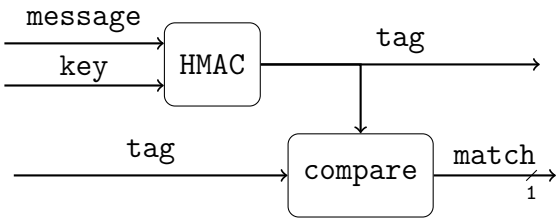
The cryptographic processor (cryptoprocessor) is tasked with the operations involving public key cryptography and zero knowledge proofs. A hardware implementation of this component provides both the security guarantees discussed above, and considerable acceleration of the cryptographic operations which undoubtedly constitute the most costly part of our protocols.

In order to reduce the number of required functions it is advisable for the algorithm selection to be such that both primitives (asymmetric cryptography and zero knowledge proofs) share the same underlying algorithm. For example, in our reference implementation we used Elliptic Curve Cryptography (ECC) for all the required cryptographic schemes (ECDSA, ECIES, and ECC Schnorr ZKP).

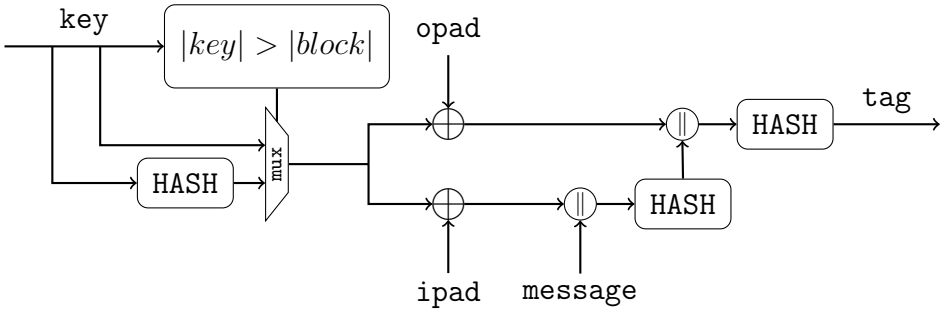
The cryptographic processor bears the highest complexity and cost of the cryptocore components but, assuming a correct implementation, no secrets are shared between distinct cryptoprocessor operations. Thus, an interface is provided for the rest of the system



(a) Block view



(b) Component architecture



(c) HMAC internal architecture

Figure 7.2: Block diagram of the HMAC component

to take advantage of the available cryptographic operations. However, the cryptoprocessor remains on the same IC as the rest of the cryptocore to reduce the attack surface on any data buses connected to the PUF.

Hardware implementations of the required cryptographic primitives have been extensively studied and optimised [31], [32], [147]–[149], and have even been realised as commercial products[150]. The specifics of the architecture are thus beyond the scope of our work, and we consider the cryptoprocessor an abstract block providing the required features. A brief discussion on cryptographic processors is included in Section 3.2.

7.3.5 Cryptographic Hash Function

Hash functions play a central role in our protocols, as is often the case in cryptographic schemes. Due to their prevalence, these functions have been the focus of many research efforts aiming at enhancing their security properties as well as reducing their footprint.

From a hardware viewpoint, the choice of a hash function requires a trade-off between complexity (and the associated area and performance costs) and security. In some cases, a construction aimed specifically at resource constrained systems is preferable [117], [151]. On the other hand, the cost gains of these lightweight primitives rely on selecting parameters that adversely affect their security level, at least in an information theoretic context.

For the purposes of this chapter, the hash function component is viewed as an abstract block with the simple input-output behaviour seen in Fig. 7.3. The proposed protocols make use of hash functions in several cases: for HMAC, entropy accumulation, protection of the raw PUF responses, and even to provide lightweight message authentication. In the following sections, the hash function block appears multiple times for clarity, while in a practical implementation the same logic block would be reused as much as possible to reduce the silicon area required by the cryptocore.

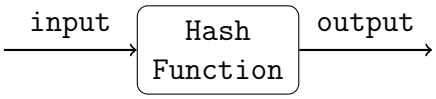


Figure 7.3: Block diagram of the Cryptographic Hash Function component

7.3.6 Random Number Generator

Randomness is employed in our protocols as a seed for cryptographic key pairs, as PUF challenges, and as random nonces. In the first case, the entropy required to ensure the security of the public key cryptographic schemes is obtained through the combination of a PUF with the appropriate key generation algorithm of the chosen public key scheme. The key generation algorithm usually involves a PRNG which alleviates some of the entropy 'imperfections' of the PUF.

When it comes to PUF challenges and random nonces, entropy requirements can be relaxed to some extent, since passing the challenges through a PUF amplifies their entropy. Additionally, random nonces are only used for a limited time period, in contrast to the asymmetric key pairs.

While a separate hardware TRNG component can be used if available, we once again exploit the existing PUF component to construct an RNG. A number of proposals have suggested the use of PUFs as True Random Number Generators (TRNGs) either directly or after post-processing[152]–[154]. Utilising PUFs in this context is motivated by the fact that they successfully mitigate a number of technical challenges regarding TRNGs. Firstly, PUFs are capable of generating high entropy seeds, without the need for secure seed storage. Additionally, TRNGs are often sensitive to environmental conditions and this sensitivity can be exploited by adversaries to force the random output to a lower entropy content. On the contrary, the entropy of many PUFs has been proven to be robust in a number of different conditions, as can be seen for example for SRAM PUFs in Section 8.5.3 and in literature[10], [153], [155]. Finally, PRNG algorithms typically absorb their seed into an internal state in an irreversible manner. This property is a significant advantage since it enables the use of the physical entropy that the PUF provides over a long time period, while only exposing the PUF response for a limited time.

However, PUFs on their own are unable to provide a long sequence of random outputs to satisfy the requirements of periodic protocols. To counteract this issue, we combine the PUF block with a cryptographically secure PRNG, following the best practices described

in RFC4086[156] and the relevant NIST guidelines[51]. Seeded by the PUF, the PRNG expands the entropy extracted from the PUF into a long sequence of random numbers. A similar method was proposed in [57], where the practicality of the method was also proven.

We propose an extended version of the method of Van Der Leest et al.[57], including a reseeding procedure, designed to refresh the entropy pool of the RNG. The initial PRNG seed is derived from the response of the PUF to a random challenge (possibly predefined during manufacturing). Periodically, the entropy of the PRNG is refreshed with a new PUF response generated by providing the PUF with output of the PRNG as a challenge, creating a reseeding loop.

Extensive literature supports the robustness of PUF entropy across various conditions [152]–[154]. Additionally, in Chapter 8 we demonstrate this robustness for SRAM PUFs through the analysis of experimental data. Yet, in certain applications of high value or where it is expected that the system will operate in adverse conditions, various statistical methods can be applied to the RNG output to detect low quality randomness[157]–[160]. Solutions specific to PUFs and supporting online testing have also been proposed[161], [162].

Seed generation

Given the unpredictability of PUF responses, the challenges used to generate the seed responses do not need to be secret or unique, and thus their initial values can be shared among devices. Using a challenge which is defined during manufacturing has significant efficiency gains for provisioning large numbers of devices as it allows the use of the same hardware and firmware across devices. Since each cryptocore IC contains a distinct PUF block, the resulting seed will be unique for every cryptocore instance. It should also be noted, that raw PUF responses are not used in any other part of the architecture, thus the same PUF can be employed for both random number generation and the rest of the operations required by the cryptocore, without affecting its security.

However, care should be taken to avoid a simple attack where the device reverts to its initial, statically defined challenge after a power cycle. In that case, the deterministic PRNG would end up producing the same sequence of random numbers as it did after the last power cycle. To avert this scenario we could rely on the partial instability of PUF responses. Unfortunately, practical PUF implementations exhibit a limited number of entropy per bit, often in the range of 5%[57]. Thus, relying on this property to produce a high-entropy seed would either require the accumulation of a large number of responses, or the characterisation of the PUF to pre-select bits that are highly unbiased. Both solutions introduce additional costs, and characterisation is only possible for certain PUF classes.

Instead, we prefer a simpler solution which is part of the reseeding process mentioned above: the last random bitstring before reseeding is not used as an output but is instead fed back to the PUF as a challenge to produce the new seed. At the same time, the feedback challenge replaces the previous PUF challenge and is stored in (unprotected)

non-volatile memory to be recovered in case of a device power cycle. In order to ensure that a seed is never used more than once, the reseeding process is triggered on every power-on, regardless of how much randomness has been output up to that point.

An entropy accumulator is used to gather and compress the entropy of the PUF responses. As described by Kelsey et al. in [163], entropy accumulators compress the entropy of the input stream regardless of how it is distributed throughout the stream. Additionally they resist attempts from adversaries with temporary access to the input stream to weaken the accumulated entropy. Cryptographic hash functions are commonly used as entropy accumulators due to their ubiquity and strong security properties[51]. All common hash functions support incremental updates to their state, a feature that can be applied to entropy accumulators allowing for the processing of large amounts of input data without the need for the equivalent storage space.

Generation Process

In our architecture, the HMAC-DRBG (Deterministic Random Bit Generator) algorithm[51] enabled us to use the HMAC and HASH components that were already included in the cryptocore. Additionally, HMAC-DRBG is hardened against prediction attempts and is resilient against occasional entropy failures, while being fairly simple to implement correctly. For its instantiation, besides the random seed, the algorithm also requires a random nonce albeit with lower entropy guarantees. In our case, additional PUF responses are generated as needed to supply the nonce value. This is in accordance with the algorithm specification, allowing the use of the same entropy source for seeds and nonces (see [51]).

According to [57], the HASH-DRBG algorithm (as it is proposed in [51]) can be efficiently implemented in hardware, providing a throughput of 400Mb/s. Since we are using the HMAC variant, we estimate the throughput of our RNG architecture to be close to 200Mb/s due to the inclusion of two hash function steps in a single HMAC step. While increased throughput is possible (e.g. via parallelisation of the hash operations) such measures are not necessary in the context of our protocols: over 6 million 256-bit nonces can be generated every second at the 200Mb/s rate. The impact of reseeding on throughput is negligible, due to the long reseeding period (2^{67} bits) offered by HMAC-DRBG.

The complete generation process is outlined in Algorithm 7.1 and the corresponding block diagram of Fig. 7.4. In the descriptions, *entropyBitsPerResponse* and *pufChallengeLength* are defined by the properties of the underlying PUF construction. The *securityStrength* parameter is defined by the application.

The $PUF()$ function used in the entropy generation passes the supplied challenge to the PUF Enclosure and returns the received *hashed* PUF response. This leads to an additional hashing operation per response, adding overhead. Alternatively, a series of challenges can be sent to the PUF Enclosure which will internally generate the responses, concatenate them and return the hash digest of the concatenation. Equivalent entropy is generated by either method, and thus in this chapter the former, simpler (but more costly) alternative is used.

Algorithm 7.1 (Random Number Generation).

```
1:  procedure RngInitialise(securityStrength)
2:      s = RngGenerateEntropy(securityStrength, True)
3:      n = RngGenerateEntropy(securityStrength / 2, True)
4:      HmacDrbgInstantiate(s, n)                          ▷ as specified in [51]
5:      reseedCounter = reseedInterval
6:
7:  procedure RngGenerate(requestedLength)
8:      if reseedCounter = 0 then
9:          s = RngGenerateEntropy(securityStrength, False)
10:         HmacDrbgReseed(s)                              ▷ as specified in [51]
11:         reseedCounter = reseedInterval
12:     reseedCounter = reseedCounter - 1
13:     return HmacDrbgGenerate(requestedLength)           ▷ as specified in [51]
14:
15: procedure RngGenerateEntropy(requestedLength, isStored)
16:     responseCount = ⌈requestedLength/entropyBitsPerResponse⌉
17:     entropy = [ ]
18:     for all i ∈ [1, responseCount] do
19:         if isStored = True then
20:             C_i = RecallChallenge(i)
21:         else
22:             C_i = RngGenerate(pufChallengeLength)
23:             StoreChallenge(i, C_i)
24:         entropy = entropy ‖ PUF(C_i)
25:     return HASH(entropy)
```

Entropy Density

The entropy density of a PUF varies, depending on its construction and implementation. For a block of data with length n and entropy density μ, the total bits of entropy are $n \times \mu$. When an error correction code is employed, producing m bits of helper data, the entropy is reduced to $(n \times \mu) - m$[10]. This entropy loss leads to a significant overhead in the required PUF size, which in turn increases the silicon area of the implementation.

Fortunately, error correction is not required for the generation of RNG seeds and thus the aforementioned overhead can be reduced and the whole entropy provided by the PUF can be used. Thus, despite the low entropy density of popular PUFs, seeds with full entropy can be realistically generated. As an example, we refer to SRAM PUFs which in our analysis showed a worst-case entropy density of approximately 4%, using a pessimistic estimation (see Section 8.5.3). In order to generate a full-entropy 256-bit seed (as required

by the HMAC-DRBG when using a SHA-256 hash function) 5689 PUF bits are required:

$$\left\lceil \frac{seed\ bits}{entropy\ per\ bit} \right\rceil = \left\lceil \frac{256}{0.045} \right\rceil = 5689\ PUF\ bits \tag{7.1}$$

In practice, a conservative entropy estimate is used, often slightly lower than the measured entropy, to account for variations in operating conditions and hardware. In our demonstration implementation, a 3% density value is assumed, increasing the required bits to 8534. Further experimental data regarding the entropy of various PUF classes, albeit in a key generation scenario, can be found in [10].

Given the impact that both the PUF implementation and the operating conditions have on entropy density, a conflict of design requirements is evident: a PUF with low error rates is suitable for authentication protocols, while one with high error rates achieves an improved entropy density. One solution would be the use of two separate PUFs, individually designed for their respective goals, but also leading to higher costs. Alternatively, the same PUF can be used in conjunction with specialised circuitry which allows for 'online' modifications of the PUF operating conditions. For example, in Section 8.5.3 we show that the supply voltage ramp-up time directly affects the entropy density of SRAM PUFs, with faster ramp up leading to higher entropy. Of course, the additional circuitry is also an overhead and thus further investigation into the trade-offs would be required. For clarity, a single PUF component is used in this chapter.

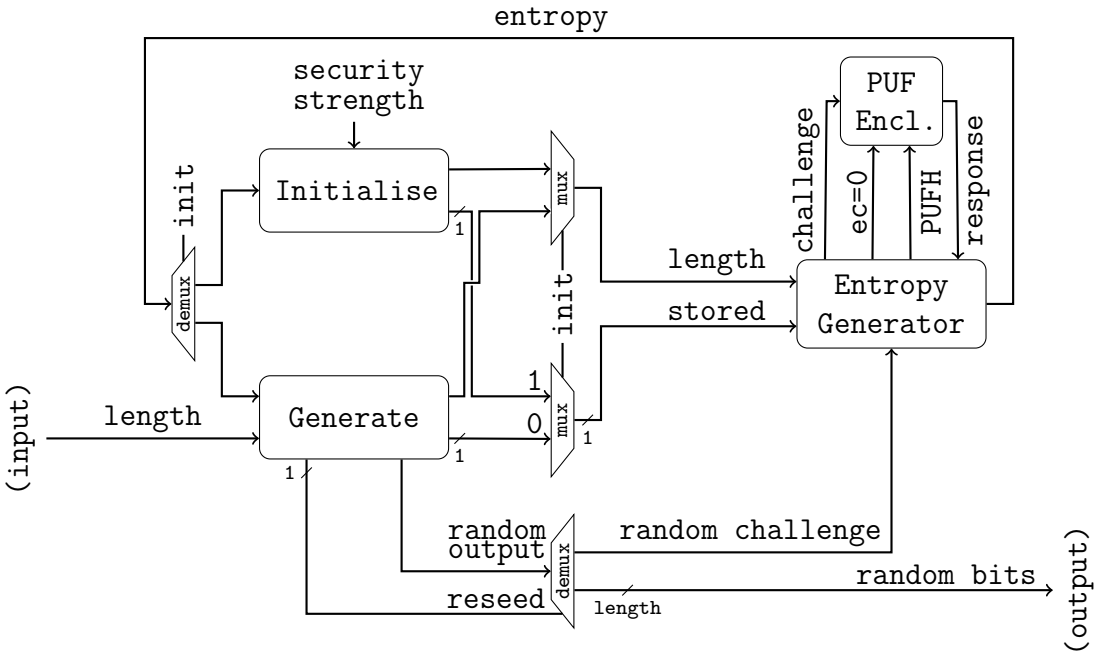


Figure 7.4: Block diagram of the RNG component

In Chapter 9 we perform a statistical randomness evaluation of the seed material as well as the output of the proposed RNG design.

7.3.7 PUF Enclosure

The PUF enclosure acts as a wrapper around the PUF block and ensures the protection of the raw responses. In order to achieve this protection, a number of operations are offered by the enclosure, to satisfy the needs of the protocols and methods discussed in previous chapters. An overview of the enclosure is shown in Fig. 7.5, where it is evident that the overall architecture can be extended with additional operations if required by future development of the protocols. A 3-bit `operation` input is used to differentiate among the operations of the PUF enclosure, and the individual bits of this input are labelled $\texttt{op}_0$, $\texttt{op}_1$, and $\texttt{op}_2$ respectively. The error correction of the PUF output is toggled via the single bit input `ec`.

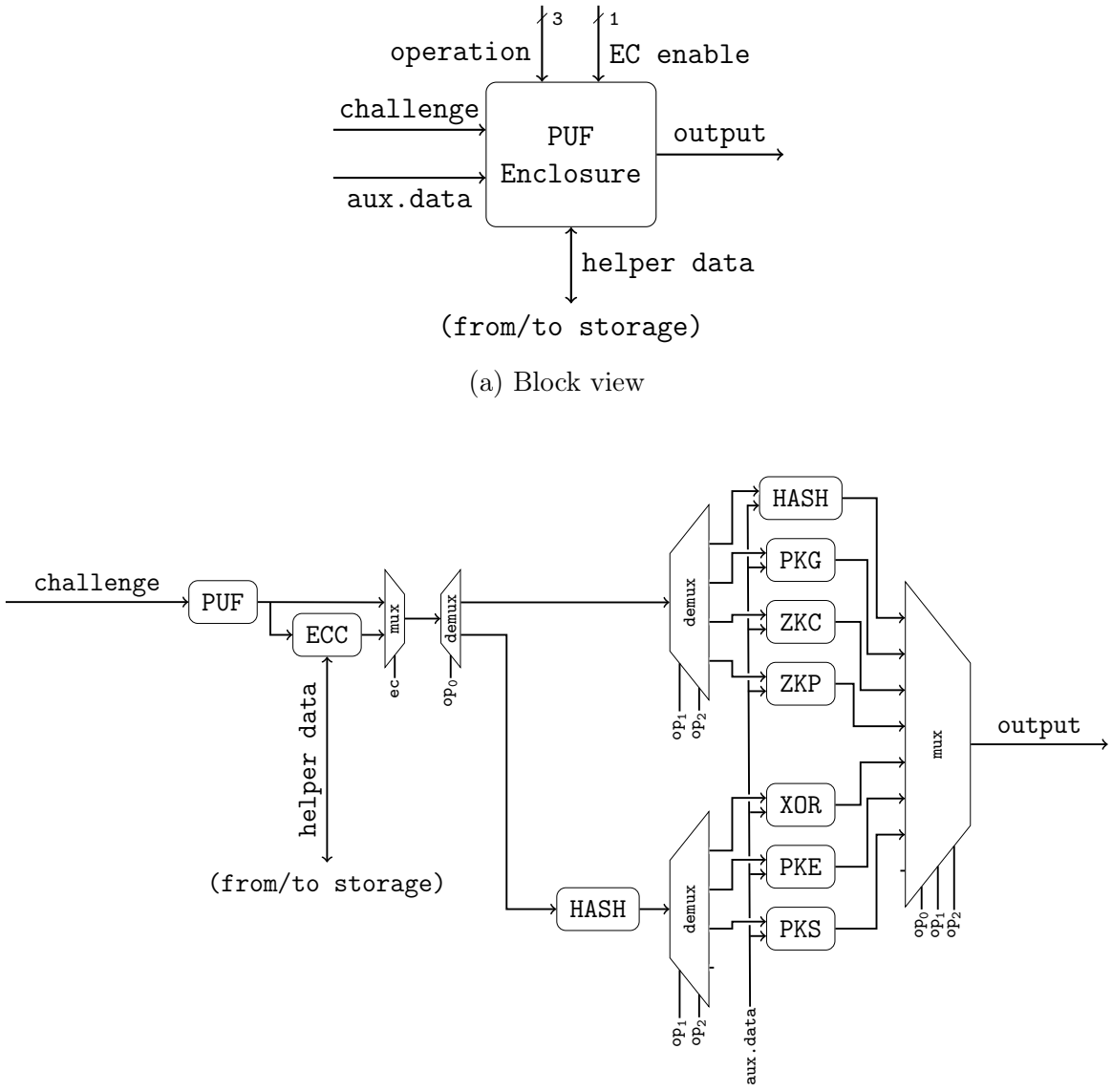
The variety of PUF classes, as it was reviewed in Chapter 4, makes the selection of a specific implementation particularly challenging. Throughout our work, our reasoning is based on SRAM PUFs due to their prevalence and strong characteristics, some of which are experimentally shown in Chapter 8. However, we see no reason preventing the use of any other class, provided that it abides by the models of Section 4.3.

From the inclusion of the PUF component in the cryptocore, it follows that the PUF is not used for other purposes, even in the case of memory-based constructions. Thus, data remanence and similar attacks are implicitly avoided[100], [164]–[166]. In this chapter, we treat the PUF block as a black box, accepting challenges and providing the corresponding responses with a degree of instability.

While the rest of the operations are fairly straightforward, the asymmetric key pair generation is of particular interest. In contrast to symmetric encryption primitives, a simple binary number cannot be used directly by the cryptoprocessor as a private key. Thus, the PUF is supplied with a random set of 'key challenges' to generate a high entropy output which is used as a seed for the key pair generation algorithm of the cryptoprocessor. Regeneration of the same seed is achieved by storing the key challenges in permanent storage.

The key generation process is outlined in Algorithm 7.2 where *seedEntropy* is defined by the cryptographic primitives used in the cryptoprocessor. The parameters *entropyBitsPerResponse*, *pufChallengeLength*, and *RngGenerate* follow the same definition as in Section 7.3.6.

The unpredictability of PUFs ensures that the resulting seed remains secret even if the key challenges are compromised. Additionally, due to the architecture seen in Fig. 7.5b, there is no need for the PUF enclosure to keep track of the key challenges. If the same challenge is used for a purpose other than key generation, then the corresponding response will be obfuscated by either a hash or a ZK operation.



(a) Block view

(b) Internal architecture

Figure 7.5: Block diagram of the PUF enclosure

Algorithm 7.2 (Key Seed Generation)**.**

```
1:  procedure GenerateKeySeed
2:      responseCount = ⌈seedEntropy/entropyBitsPerResponse⌉
3:      entropy = [ ]
4:      for all i ∈ [1, responseCount] do
5:          if firstRun = True then
6:              C_i = RecallChallenge(i)
7:          else
8:              C_i = RngGenerate(pufChallengeLength)
9:              StoreChallenge(i, C_i)
10:         entropy = entropy ‖ PUF(C_i)
11:     return HASH(entropy)
```

7.3.8 Error Correction

The error correction block seen in Fig. 7.6 ensures the reproducibility of PUF responses by correcting (up to) a number of bit errors. To enable error correction, redundant 'helper' data are produced and stored during the 'enrolment' phase, when a specific PUF challenge is encountered for the first time. The next time the same challenge is applied, the 'reproduction' phase takes place and the corresponding helper data is retrieved to be used as an input to the error correction algorithm.

Due to its complexity, the error correction block has a relatively high footprint, adding overhead to the overall implementation. Unfortunately, it is also an indispensable part of any PUF realisation and thus a significant body of research focuses reducing or correcting bit errors[102], [103], [167]–[170]. Recent work, however, has suggested that BCH codes, which are the most common error correction method in the PUF domain, can in fact be implemented in a fairly efficient manner[171].

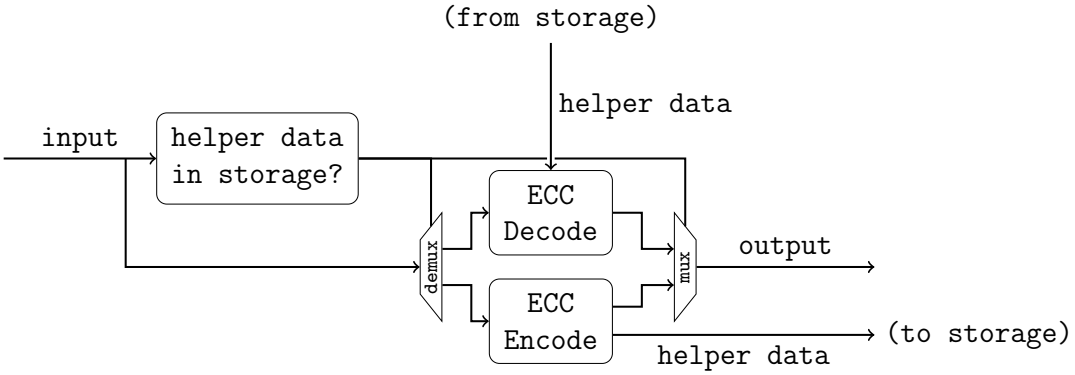


Figure 7.6: Block diagram of the Error Correction Code (ECC) component

The high BER exhibited by certain PUFs would make the implementation of the error correction code exceedingly complex. One of the most straightforward solutions is given by Bösch et al. in [172], where a binary repetition code is used to drastically reduce the BER before employing a Reed-Muller code to correct the remaining errors. This

method allows for practical applications with even the most unstable PUFs (which often exhibit BERs up to 20%) without an excessive error correction overhead. The reference implementation of Chapter 9 employs a variation of this method. Binary block codes such as the above can be implemented very efficiently in hardware, making them an excellent choice for resource-constrained devices[10].

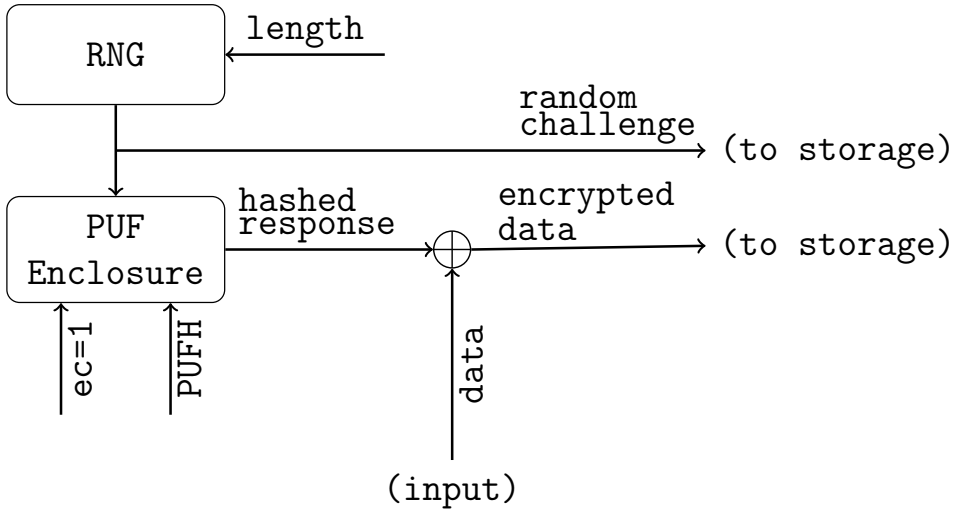
In the majority of existing research, the number of error bits between the original and subsequent PUF responses is assumed to remain fairly stable over time, for the same operating conditions. However, for certain PUF classes ageing effects may affect the BER. Ageing is especially pronounced in SRAM ICs where regular operating stress temporarily skews the bias of individual cells, influencing their power-up value and thus the PUF responses[173]. In the simplest case, the reference response and the associated helper data can be periodically refreshed, alleviating the effect of ageing. This approach is particularly suitable for periodic authentication protocols similar to the ones presented in Chapter 6. Alternatively, the drift phenomenon can be exploited to detect the age of the IC and take appropriate action. Further investigation would however be needed in the latter case, since ageing effects can often be, partially or fully, reversed by simply powering off the IC[174].

7.4 Optional Extensions

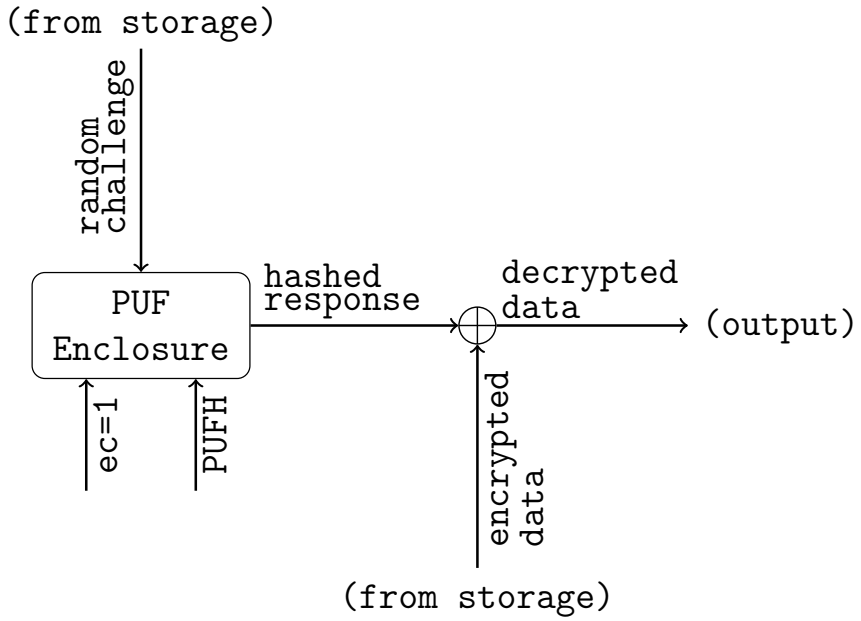
It is clear that the components included in the cryptocore can be used in a variety of applications in addition to the protocols described in the previous chapters. As a result, some of the supported operations can be exposed to the rest of the system via an I/O interface similar to the one described above. One can easily envision the value of asymmetric cryptography, zero knowledge, error correction and even XOR operations, especially since hardware implementation will most certainly have performance advantages as well. Unfortunately, the extend to which this extension can be securely achieved is unclear. Allowing external control and supply of data without appropriate controls expands the attack surface of the system, partially defeating one of the initial goals of the cryptocore. For example, an adversary would be allowed to perform a chosen plaintext attack by repeatedly instructing the cryptocore to perform the encryption. Consequently, the verification complexity for the security of the core also rises exponentially due to the multiplicity of possible attacks, and additional methods would be required to ensure that the cryptocore design does not inadvertently leak internal information.

Similarly, the implemented PUF logic can be used in a variety of other applications as seen in the numerous PUF-based solutions referenced throughout this thesis. One of the most interesting applications in the 'low-level' context of the cryptocore is local storage encryption. Several sophisticated methods for encrypting off-chip volatile and non-volatile storage through the use of PUFs have been proposed[58], [59], [175]. In Fig. 7.7, we demonstrate a simple architecture for the protection of off-chip storage contents. The encryption key is generated randomly and only used once, in a way similar to one-time pads, under

the assumption that the random output of the RNG component is not repeated. After the data is decrypted, the random challenge and the associated helper data are discarded. Due to the required error correction, this method involves some computation and storage overhead. Thus, it is not suitable for high frequency access but rather constitutes a secure and lightweight alternative for providing confidentiality of infrequently accessed data.



(a) Encryption



(b) Decryption

Figure 7.7: Block diagram of local storage encryption/decryption component

7.5 Conclusion

In this chapter we demonstrated the practical feasibility of the methods proposed in the previous chapters via the presentation of a cryptographic core architecture. This was also supported by the discussion of the various practicalities that arise during its design and implementation. We described in detail the operations supported by the core, including a secure PUF interface, random number generation and several cryptographic primitives, and showed how these operations can be combined with minimal overhead. We also showed that the employed primitives are guided by established research and industry practices, which is conducive to both security validation and the ability to make use of highly optimised implementations.

8. SRAM PUFs

8.1 Introduction

While the validity of the rest of this thesis does not depend on a specific PUF class, we selected Static Random Access Memory (SRAM) PUFs as the basis of our demonstrator system, in order to establish the feasibility of our methods. This decision was driven by the many advantages of SRAM PUFs: they are a readily available Commercial Off-the-Shelf (COTS) component, often already included in many system architectures. They are also cost-efficient and their I/O can be easily manipulated in the digital domain without additional circuitry. These advantages have made SRAM PUFs popular in industry and research applications, reinforcing the reasoning for their use in our work.

From a hardware point of view, SRAM PUFs are simply SRAM ICs, the contents of which are accessed before initialisation, as discussed in Section 4.2. These uninitialised contents comprise the power-up values of individual SRAM cells, shown in Fig. 8.1. The susceptibility of the cells to manufacturing variations but also variations of environmental conditions leads to stochastic processes that affect the power-up values. These processes are complicated even further by physical phenomena including ageing, cell interdependencies, stress-induced bias etc. Thus, SRAM PUFs, while deceptively simple to implement, provide a wealth of investigation opportunities and have the potential to support novel security paradigms.

In this chapter, we discuss the behaviour of SRAM PUFs based on experimental data. The experiment was designed and developed by Michael Walker and Dr. Alex Bystrov as part of a PUF research project of which our work was also part. The data acquisition process was performed by Dr. Alex Bystrov. The data analysis and interpretation was solely the work of the author of this thesis. All parties were members of the uSystems Group, Newcastle University at the time of their involvement.

We start by defining the appropriate metrics and describing the hardware and the conditions of the data acquisition process. Subsequently, we demonstrate how SRAM behaviour follows the PUF models that we defined in Section 4.2, and we look into the instability of SRAM cells in different conditions. While, to the best of our knowledge, current research work has been focusing on removing and reducing the aforementioned instability, we view it as a feature that guarantees the randomness of PUF outputs even in the presence of environmental influences.

The experimental data is used to assess the stability and randomness of the SRAM behaviour, as well as its suitability for PUF applications. While the experiment itself was motivated by goals separate to the work presented in this thesis, it was deemed necessary

to examine the collected data in the context of our work, in order to validate that SRAMs exhibit the behaviour we assumed in previous chapters. Additionally, and perhaps more importantly, the data provides empirical proofs of said behaviour while the SRAMs are subjected to different combinations of temperature, stress and ramp-up time.

Several researchers examined their power-up behaviour, mainly while SRAM PUFs were still a fairly new concept. The seminal work in the field of SRAM PUFs was carried out by Guajardo et al.[70] and Holcomb et al.[153] independently but at approximately the same time. Later notable works include [155] and [176] where SRAM power-up behaviour was examined in both simulations and hardware experiments. These publications served as a solid basis for our analysis.

The main drawback of these sources in the context of our research was our lack of access the raw data, leading to an inability to evaluate it from a different perspective. In our evaluation of SRAM PUFs we focus on their cell-level behaviour instead of their general uniqueness and stability characteristics. Additionally, due to the use of custom hardware specifically designed for the experiment, we were able to examine conditions that, to the best of our knowledge, were not part of the existing literature.

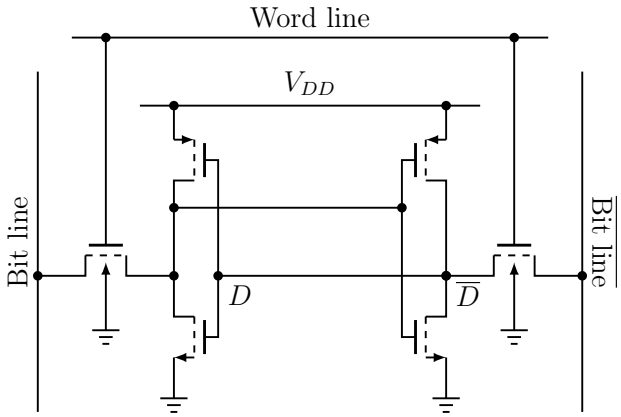
8.2 Physical Behaviour

SRAM is a highly common digital memory technology included in almost every electronic device. Modern SRAMs comprise thousands or millions individual memory cells capable of storing a single bit each, and implemented in a CMOS six-transistor (6T) architecture. This architecture, shown in Fig. 8.1a, uses two inverters (four transistors) to store the digital value, and two additional transistors to allow for read and write operations. The inverters are cross-coupled in a feedback loop which allows the cells to retain their stored value while they are powered on (see Fig. 8.1b).

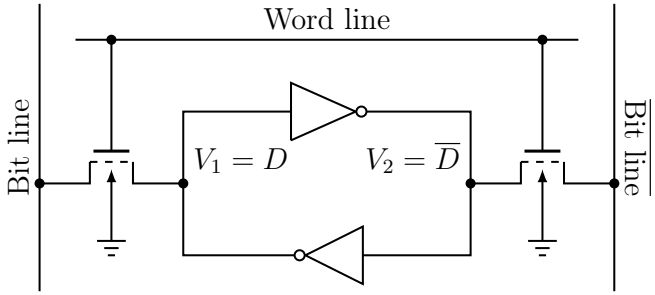
This architecture leads to a bistable circuit with three operating points: stable low, stable high, and metastable, illustrated in Fig. 8.1c. When a cell is in one of its stable points, external interference of any form is unlikely to affect the output of the circuit. However, in the metastable condition, the opposite is true: relatively minor influences are amplified by the tendency of the circuit to assume a stable state[177]. The final logic value is independent for each cell, and is determined by the relative strength of the corresponding transistors.

In theory, SRAM cells are designed to be fully symmetrical, mainly for performance reasons. However, random process variations lead to slight asymmetry in the sizes of the inverters, giving each cell a preferred state. The extend of the aforementioned mismatch also determines the bias of the corresponding cell: a significantly larger inverter will strongly 'pull' the cell to one of the stable points. Cells with highly asymmetric inverters are more sensitive to variation such as changes in the supply voltage and thermal noise (see Fig. 8.4).

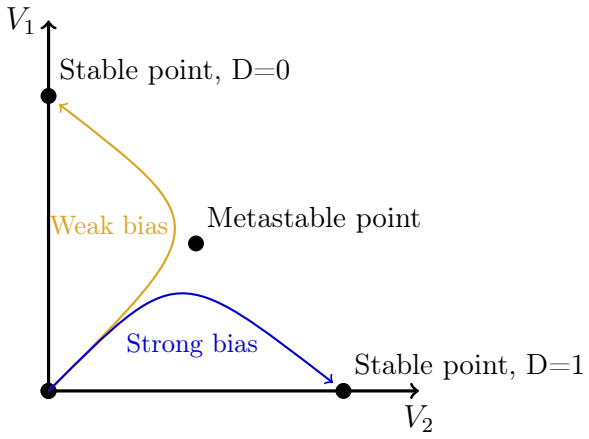
Current SRAM implementations exhibit a high percentage of stable cells, normally



(a) SRAM cell circuit[177]



(b) SRAM cell bistable cross-coupled inverters[177]



(c) SRAM cell voltage transfer curves

Figure 8.1: Architecture and behaviour of CMOS SRAM cells

over 95%, with the remaining cells showing different degrees of unpredictable behaviour (as seen in previous work [153], [155] and in our analysis in the following sections). Despite referring to them as stable and unstable, in reality, cells occupy the whole spectrum between these two categories. Thus, a more appropriate designation includes the extend of their stability, commonly termed cell *bias*. We further discuss bias in the next sections.

For the use of SRAMs as PUFs, both stable and unstable bits are important. Due to the physical effects governing cell behaviour, their preferred power-up values are highly unique for each IC produced. Thus, by collecting the power-up values of several cells, a unique bitstring can be formed. However the instability of certain cell means that repeated measurements of the power-up values after a power cycle will produce slightly different results. In many cryptographic scenarios, even a single bit of difference leads to a failure and thus appropriate error correction is required, as discussed in previous chapters. On the other hand, the eventual value of each unstable cell is highly unpredictable and mostly depends on random environmental influences, thus making the SRAM PUFs a good source of entropy.

To generate a response from an SRAM PUF, one needs to supply a challenge in the

form of one or more memory addresses which in turn select the corresponding memory cells. The combination of the values stored in these cells constitutes the PUF's response to the specified challenge. In practice, cells are not individually addressable in off-the-shelf SRAM ICs and thus Challenge-Response Pairs (CRPs) are formed by the concatenation of memory words. After initially supplying the SRAM IC with power, the cells will retain their values until the power is removed, or a write operation in performed. This allows for the recovery of the PUF responses on demand, without the need to keep them in storage, increasing exposure of the PUF state. In addition, the volatile nature of the cells ensures that, in the general case, the PUF state is not available when power is not supplied to the SRAM.

It is evident that since SRAMs contain a finite number of cells, their potential for generating CRPs is also finite, categorising them in the 'weak' PUF group. Nonetheless, published attacks on SRAMs have been limited to 'traditional' methods of side-channel analysis[178]. Invasive methods in the physical domain have also been proposed, although such attacks normally require specialised equipment and skill[179]. For example, Anagnostopoulos et al.[100], [166] recently proposed methods exploiting data remanence effects in very low temperatures to access the SRAM data without applying power to the IC. Another interesting class of attacks, in the sense of a denial-of-service, was proposed by Roelke et al. [180], using wearout effects to permanently modify the power-up values of SRAM ICs, thus forcing future cryptographic operations to fail. Of course, without additional protection, adversaries could simply read the whole contents of an SRAM IC, reducing the unpredictability of future responses. As a result, the raw PUF responses require protection from exposure, a fact which was one of the main motivating factors for the cryptocore architecture of Chapter 7.

The above offers only a simplified view of SRAM behaviour. As expected in such designs, correlations exist between quantities that are otherwise assumed to be independent. For example, electromagnetic noise emitted by a cell can affect neighbouring cells, biasing their power-up values. This is further complicated by the geometry of the IC, where cells are arranged in rectangular blocks and thus the distance between cells and the actual number of neighbouring cells vary. Evidently these issues require further research and we are not aware of published work on this respect.

Furthermore, SRAM ICs exhibit ageing under prolonged normal operation, or under temperature or voltage stress. Arguably, the most pronounced of these stress effects is Negative Bias Temperature Instability (NBTI), a phenomenon where the threshold voltage of (mainly PMOS) transistors grows with time while continuous high temperature stress is applied[181]. This creates a tendency in the cells to shift away from the value they were previously holding, thus creating an 'oscillation' effect that was also apparent during our experiments. The cells slowly recover after the stress is removed, although a full recovery is not always guaranteed[153], [182]. Active techniques have also been proposed which either prevent ageing to some extend, or speed up the recovery from distortions similar

to NBTI[104], [183], [184].

8.3 Metrics

Several metrics have been established to evaluate the behaviour of PUF classes and implementations. This behaviour is due to stochastic processes which are hard to fully model. As a result, the following metrics attempt to characterise various properties of the physical unclonable construction based on the experimental observations of its state.

While all the metrics apply to the majority of PUF classes, we discuss them with a view to the particulars of SRAM PUFs. In this context, a response is the concatenated contents of the memory cells specified by the challenge. Thus, we are able to simplify the calculation of certain metrics by directly comparing the whole array of PUF ICs. Besides its obvious practical benefits, this strategy also allows for the alleviation of the effect of individual cells in the results, allowing for a more generalised picture of the PUF behaviour. For the remainder of this chapter, the individual bits of SRAM PUF responses are also referred to as 'cells'.

Inter-distance

Inter-distance is defined as the hamming distance between two responses generated from two different PUF instances x and y when the same challenge C is applied:

$$d_{x,y}^{inter} = HD[PUF_x(C), PUF_y(C)] \tag{8.1}$$

Inter-distance describes the uniqueness of a PUF. In practice, a high inter-distance is desirable in order to successfully differentiate among PUF instances. Inter-distance is often reported as a fractional hamming distance, signifying the percentage of differing bits (assuming that all instances generate the responses of the same length):

$$d_{x,y}^{inter} = \frac{100}{|PUF_x(C)|} HD[PUF_x(C), PUF_y(C)] \tag{8.2}$$

Intra-distance

Intra-distance is defined as the hamming distance between two responses (observations i and j) generated from the same challenge applied to the same PUF instance x:

$$d_{x,i,j}^{intra} = HD[PUF_x^i(C), PUF_x^j(C)] \tag{8.3}$$

Intra-distance describes the instability of a PUF, which is the probability of mismatch between responses to the same challenge. Thus, in practice, a low intra-distance is desirable in most applications. Similarly to the inter-distance, intra-distance is reported as a

fractional rather than an absolute hamming distance:

$$d_{x,i,j}^{intra} = \frac{100}{|PUF_x(C)|} HD[PUF_x^i(C), PUF_x^j(C)] \tag{8.4}$$

We can also derive the Bit Error Rate (BER) of a PUF, by taking the mean intra-distance over multiple observations m:

$$BER_x = \frac{100}{m} \sum^{m} \frac{HD[PUF_x^i(C), PUF_x^j(C)]}{|PUF_x(C)|} \tag{8.5}$$

Cell Bias

In contrast with the previous metrics, the cell bias examines individual bits of PUF responses. The bias of a cell i over multiple observations m of the same response is defined as the number of observations where the specified cell has a value of 1:

$$bias_i = \sum^{m} [i = 1] \tag{8.6}$$

Since the interpretation of the above value is dependent on the number of observations, the bias is often represented as a percentage of the observations:

$$bias_i = \frac{100}{m} \sum^{m} [i = 1] \tag{8.7}$$

The bias of a PUF cell is the probability that the cell will have the logical value of 1 in future observations. A *strongly* biased cell (bias close to 0% or 100%) will have a value of 0 or 1, respectively, with a high probability, in future observations. In the same sense, a *weakly* biased cell (bias close to 50%) can be easily affected by external conditions and as a result its value in future observations is more unpredictable.

Thus, it becomes evident that knowledge of the bias allows the prediction of future values of the corresponding response bit to a degree that is often more accurate than random guessing. This is especially the case for strongly biased cells. At the same time, the examination of bias statistics provides valuable insights into the behaviour of the PUF as a whole.

Min-entropy

The min-entropy metric can be calculated for individual bits as well as whole PUF responses and it represents the intrinsic unpredictability of the corresponding random variable:

$$minentropy_i = -log_2(max\{bias_i, 1 - bias_i\}) \tag{8.8}$$

In order to estimate the min-entropy for PUF cells, we must assume that the power-up value of each SRAM cell is independent from the values of other bits. While, given

the architecture of SRAM ICs, dependencies between cells are bound to exist, this is considered a reasonable assumption, especially since inter-bit dependencies cannot be easily characterised by adversaries. A potential adversary with access to such information would be able to fully predict the PUF responses and thus min-entropy would be of no interest in that case. The mean min-entropy per bit for a PUF response j with length m bits is defined as:

$$\overline{minentropy_j} = \frac{1}{m}\sum^{m} minentropy_i \tag{8.9}$$

Unstable Cell Relative Distance Deviation

We have established that certain SRAM cells exhibit a tendency to alternate between power up states, and this tendency is quantified by the cell bias. While, intuitively, the aforementioned cells are expected to be fairly uniformly spread across the whole SRAM, this may not always be the case, especially when taking into account the effects of IC geometry. In an attempt to quantify the dispersion of unstable cells, in comparison to the uniform case, we make use of a new metric, based on the distance between them.

Assuming a sequential arrangement where cells are addressed by their index, the distance between cell i and j is:

$$D_{ij} = j - i \qquad for\ i > 0,\ j > i \tag{8.10}$$

When m unstable cells are uniformly distributed over an IC with size n bits, their *mean distance* is defined as:

$$\overline{D} = \frac{n}{m} \tag{8.11}$$

Based on the above, we can derive the *relative distance deviation* for cell j (with previous cell i) as:

$$RDD_j = \frac{D_{ij} - \overline{D}}{\overline{D}} \qquad for\ i > 0,\ j > i \tag{8.12}$$

The RDD values for a given IC of size m bits, lie between -1 and $\frac{m-\overline{D}}{\overline{D}}$. Values that are less than 1 signify unstable cells 'clustered together', while values exceeding 1 signify a larger spread of unstable cells. Evidently, since there are typically thousands of unstable cells in an SRAM IC, we make use of aggregate statistics that allow us to obtain a summative view of the unstable cell distribution. A particularly useful metric is the *median relative distance deviation.*

In the above, we deliberately did not define the exact criteria for what constitutes an unstable cell, since its definition varies. Theoretically, any cell with a bias that is not 0% or 100% can be considered unstable. However, to maximise the entropy provided by a cell, we would opt for a more strict definition, such as the one used in Section 8.5.3, with unstable cells having a bias between 49% and 51%.

8.4 Experimental Setup

A customised testing apparatus was designed to minimise the effects of noise and interconnection variations while the SRAM data was acquired. Twenty instances of the selected SRAM IC, Cypress CY7C1041DV33, were arranged in array and placed on a dedicated PCB board which provided the necessary communication logic. As seen in Fig. 8.2, the power-up values were read through a dedicated USB communication component (enabling high-speed acquisition) and the required ramp-up waveforms were provided by a waveform generator. The acquisition process was controlled by an FPGA development board which generated the necessary control and address signals.

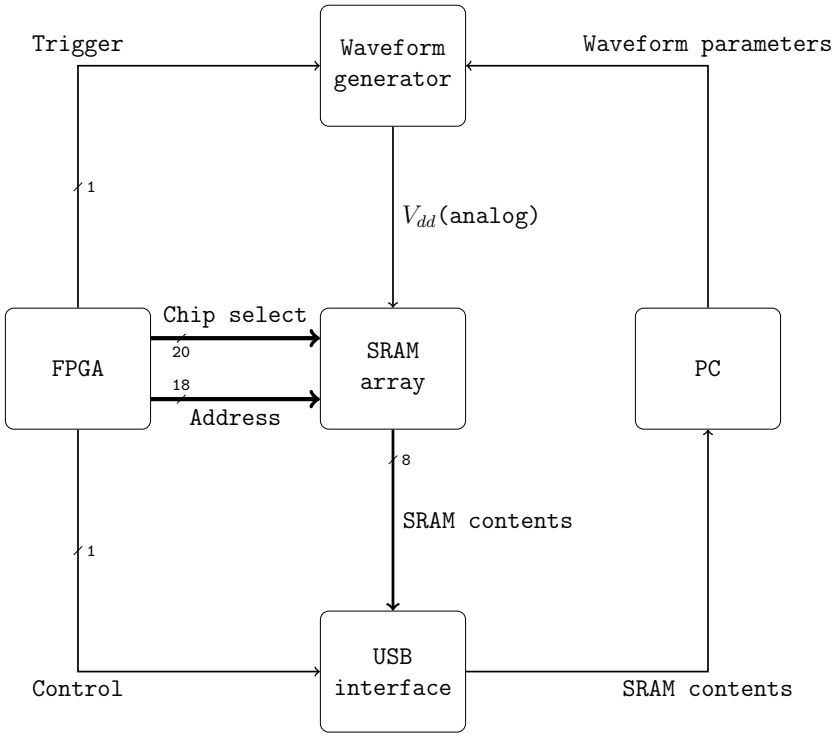


Figure 8.2: Experiment hardware setup

Fig. 8.4, extended from [155], illustrates the factors affecting the power-up values of SRAM ICs along with the studies in which such effects are discussed. Three of the factors were evaluated in the experiment: supply voltage ramp-up time t_{ramp}, ambient temperature, and IC age. The first two were externally applied to the ICs, while the latter was achieved by a 'hot ageing' process: the ambient temperature was raised to 100°C and 1 million power cycles were performed before returning the board to the nominal temperature. Applying this type of stress to the ICs increased their ageing rate, allowing the acquisition of measurements representing months of actual use. The selected operating conditions are summarised in Table 8.1, generating 28 operating corners where measurements were taken. The metrics discussed in the following sections cover the SRAM behaviour in single operating corners as well as across conditions.

Each SRAM IC had a size of 4Mbits bringing the total size of the array to $20 \times 4Mbit = 80Mbit(10Mb)$. The complete array was powered up with the selected ramp-

up curve (8.3), and its contents were read and saved to a binary file for analysis. This process was repeated 100 times in each operating corner (comprising a combination of an environmental condition and a ramp-up time) resulting in over 2 billion data points describing the individual cell values.

Fig. 8.3 shows the voltage curve for a single test cycle, with $t_{ramp} = 50ms$. The ICs were kept at zero voltage for 10ms, followed by a progressive increase of the supply voltage to 300mV. After reaching this operating point, the metastability of the cells was resolved and thus the voltage was sharply increased to the nominal supply voltage of 3V. After the appropriate time period, the supply voltage was actively pulled down to zero before the next test cycle.

Label	**Condition**
25°C (nominal)	25°C
100°C	100°C
25°C aged	25°C (after hot ageing)
100°C aged	100°C (after hot ageing)
fast t_{ramp}	500ns
	5us
	50us
nominal t_{ramp}	500us
	5ms
	50ms
slow t_{ramp}	500ms

Table 8.1: Selected operating conditions

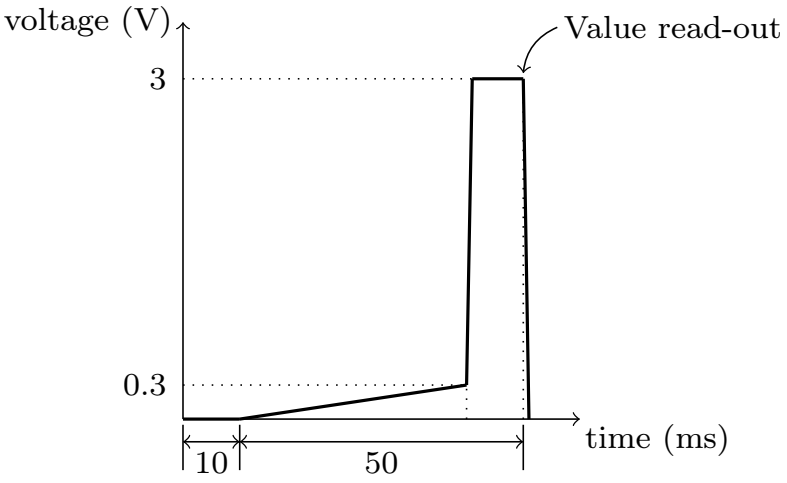


Figure 8.3: Voltage ramp up curve for $t_{ramp} = 50ms$

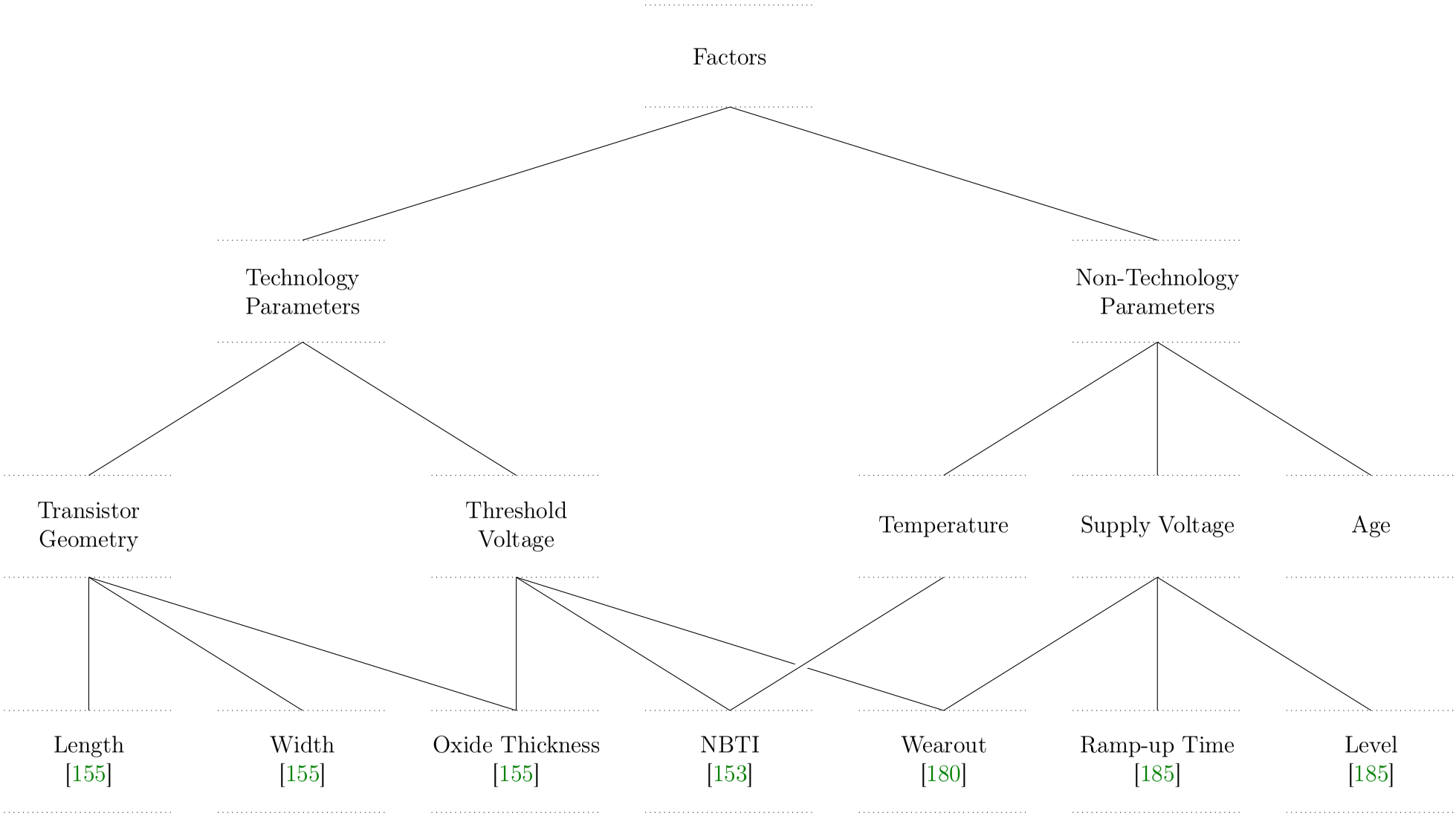


Figure 8.4: Taxonomy of influencing factors for SRAM power-up state

8.5 Behaviour as PUF

8.5.1 Uniqueness

As discussed in Section 8.3, the uniqueness of a PUF class is determined by its inter-distance distribution. We calculated the inter-distance by taking the fractional hamming distance between the power-up values of every IC combination. This process was repeated for every operating corner as well as across operating corners, yielding a total of almost 1.5 billion data points[1]. Subsequently, the measurements were divided into categories (by operating corner, IC etc.) and summary statistics were calculated per category.

The optimal inter-distance, which would represent fully uncorrelated responses, is 50%. In practice, values that are approximately 50% are considered sufficient. Overall, the SRAM ICs under evaluation exhibit fairly high uniqueness. As seen in Fig. 8.5 and Table 8.2, the inter-distance (and thus the uniqueness) varies quite significantly depending on the operating conditions. In every case, higher ambient temperature leads to a higher inter-distance, especially when the voltage ramp-up time is short. The effect of temperature is much less pronounced for slower voltage ramp-ups, becoming virtually non-existent for times longer than 500us.

We attribute this behaviour to the fact that cells are only sensitive to noise while their supply voltage is kept at levels just under the nominal supply voltage of 300mV[186]. While a cell is in this sensitive state, minor amounts of noise are sufficient to induce a transition of the cell to its favoured state. Thus, greater ramp-up times increase the probability of cells assuming their favoured state by prolonging their exposure. In other words, by increasing the ramp-up time, one can force the SRAM ICs to produce power-up values that are highly unique, as shown by our measurements. Similarly, the magnitude of thermal noise increases with temperature and forces the cells to their favoured states even faster than at lower ambient temperatures. On the contrary, ageing seems to have quite a limited effect (less than 2%) on inter-distance, and its effect diminishes with the increase of either ambient temperature or t_{ramp}. Tables A.1 to A.3 provide a more detailed view into the inter-distance measurements.

In summary, the effect of ramp-up time is fairly constant regardless of the ambient temperature. On the other hand, ambient temperature has a decreasing effect on the inter-distance as ramp-up times increase. Additionally, Fig. 8.5 shows a saturation effect for the ramp-up time: after the threshold of 5ms, further increase in t_{ramp} has little effect on the inter-distance[2]. The above observations allow us to conclude that higher temperatures and higher ramp-up times would be beneficial for SRAM PUF applications where high uniqueness is needed. However, in practice it is quite unusual for an IC to operate at an ambient temperature of 100°C thus leaving hardware designers with the only option of increasing t_{ramp}.

In Table 8.2 we focus on the minimum (mean) inter-distance in every operating corner.

[1] To be exact, 1,489,600,160 data points were acquired.

[2] Each observation refers to a specific IC and the ICs are sorted in ascending order for each t_{ramp} value. For example, the values for IC no.1 are found at $x \in \{0, 20, 40, 60, 80, 100, 120\}$.

	500ns	5us	50us	500us	5ms	50ms	500ms
25°C	34.27 %	34.80 %	40.23 %	45.64 %	49.11 %	49.50 %	49.47 %
25°C aged	29.08 %	34.59 %	39.64 %	44.57 %	48.94 %	49.05 %	49.46 %
100°C	41.78 %	43.01 %	47.80 %	49.37 %	49.14 %	49.46 %	49.46 %
100°C aged	41.20 %	42.71 %	47.58 %	49.46 %	49.51 %	49.47 %	49.23 %

Table 8.2: Minimum inter-distance in all operating corners

In comparison to other PUF classes, our data confirms the uniqueness of SRAM PUFs and their superiority in this respect. This is true in all the conditions that we examined which, to the best of our knowledge, have not been examined before in this context. An excellent reference of similar metrics for several PUF classes is the work by Maes[10].

Additionally, our measurements show that the inter-distance remains sufficiently high, even across different operating conditions. In many identification scenarios the PUF responses are compared to a previously recorded value, and a decision is made depending on whether or not their difference is below a certain threshold. While adversaries are potentially able to negatively influence the uniqueness of SRAM responses (by controlling the supply voltage or the ambient temperature), appropriate selection of the threshold is highly effective against such attacks.

Furthermore, as we will see in the next section, the typical intra-distance values in a certain operating corner are two to three times smaller than the respective inter-distance values. Thus, there is a very low probability of false positives in authentication or key generation scenarios. Fig. 8.6 illustrates the distribution of inter-distance and intra-distance values in every operating corner and includes measurements between operating corners as well. The apparent (almost) overlapping section between the two metrics is discussed in the next section.

The histograms of Figs. 8.6 to 8.8 contain the following data points:

- Fig. 8.6: All available data points.
- Fig. 8.7: Data points referring to the nominal conditions: 25°C, $t_{ramp} = 500us$.
- Fig. 8.8: All available data points excluding values referring to distances *between* operating corners.

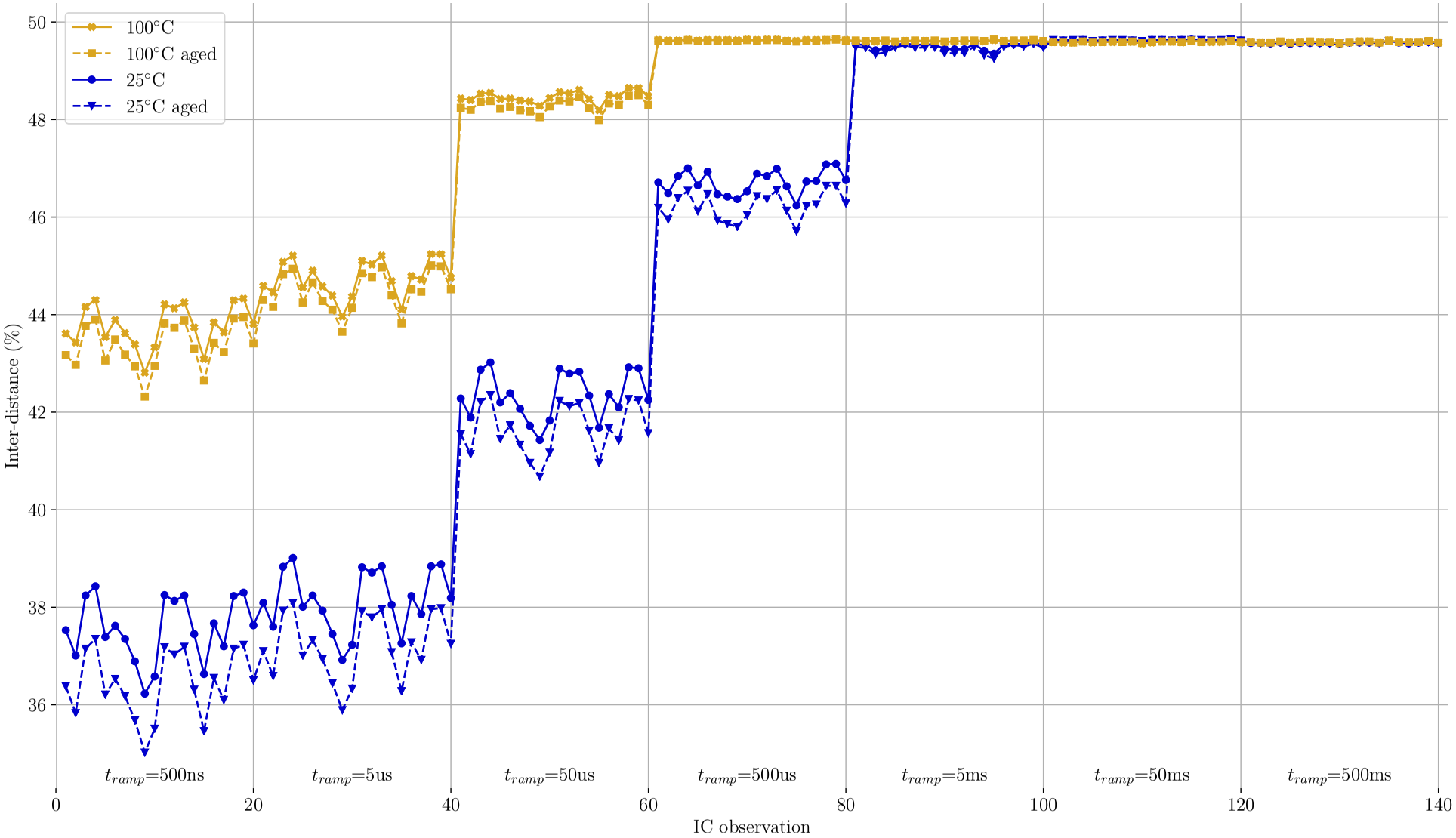


Figure 8.5: Mean inter-distance by IC and operating corner

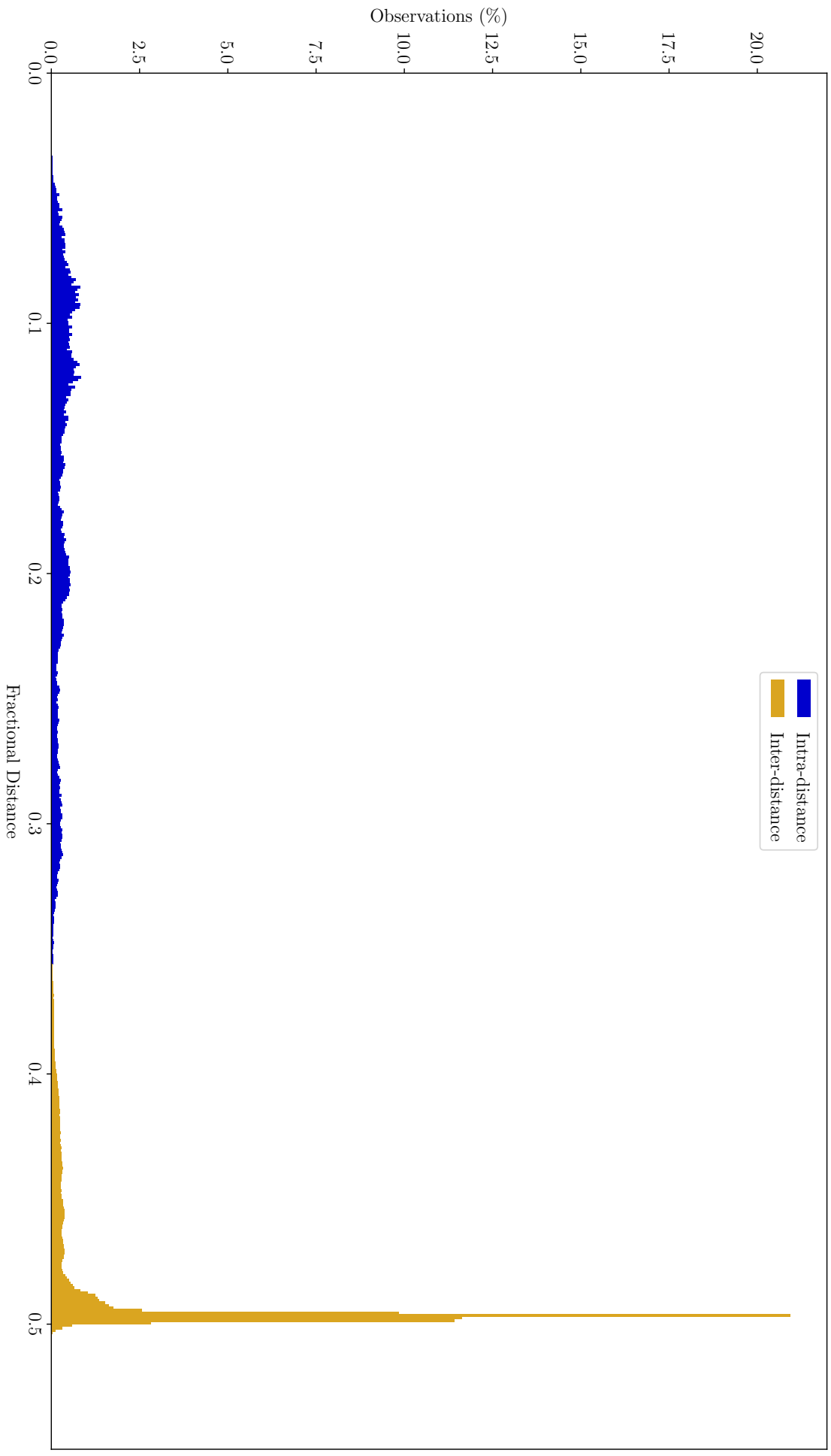


Figure 8.6: Distance distribution, all measurements

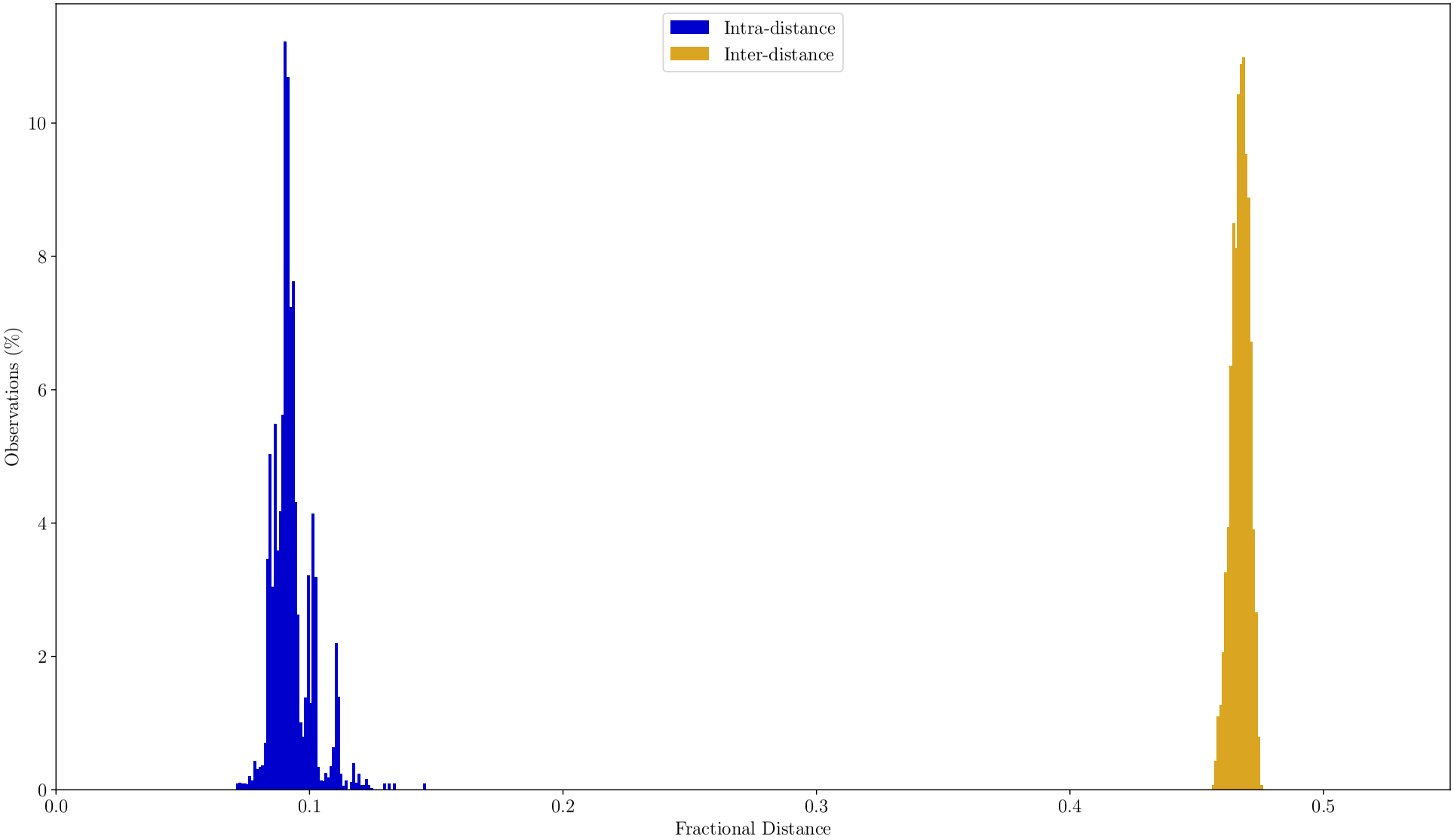


Figure 8.7: Distance distribution at nominal conditions (25°C, $t_{ramp} = 500us$)

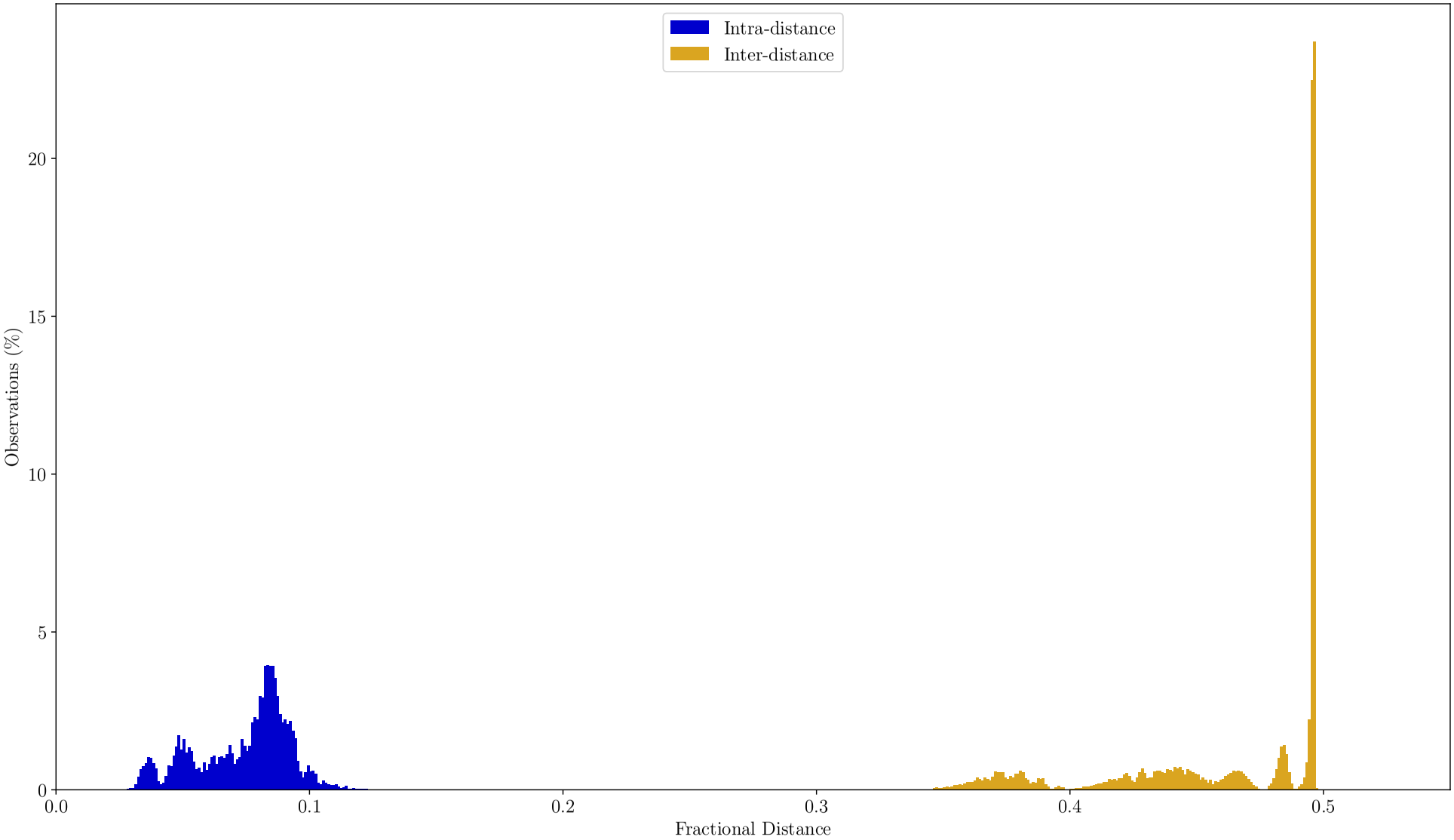


Figure 8.8: Distance distribution in every operating corner

8.5.2 Reproducibility

Contrary to the uniqueness metric discussed above, reproducibility is described by intra-distance, the distance between distinct responses to the same challenge. In this case, the optimal intra-distance value is zero, although practical PUF implementations are rarely able to achieve such a result. Table 8.3 summarises the maximum intra-distance in every operating corner of the experiment. Evidently, taking the maximum value of the metric provides a pessimistic estimation of intra-distance, which is nevertheless useful in designing resilient systems. A more realistic view of the intra-distance distribution is given in Figs. 8.7 and 8.8, showing that its value rarely exceeds the 10% threshold. Since the intra-distance is closely related to the bit error rate, its magnitude is usually far more important than that of inter-distance. This is mainly because even a slight variation in intra-distance is caused by a change in a large number of SRAM cells, making the correction of such errors far more challenging.

	500ns	**5us**	**50us**	**500us**	**5ms**	**50ms**	**500ms**
25°C	12.79 %	13.58 %	15.32 %	15.38 %	14.01 %	11.24 %	7.68 %
25°C aged	12.18 %	14.24 %	16.34 %	15.05 %	14.02 %	11.78 %	8.64 %
100°C	13.21 %	13.76 %	13.26 %	10.53 %	9.00 %	8.47 %	5.87 %
100°C aged	14.43 %	14.52 %	13.64 %	12.64 %	10.80 %	8.95 %	5.58 %

Table 8.3: Maximum intra-distance in all operating corners

The variations in PUF responses are the direct result of noise affecting the SRAM cells as described above. Due to this correlation, intra-distance varies significantly with the operating conditions. As mentioned in Section 8.5.1, the increased thermal noise in higher ambient temperatures forces the cells to assume their favoured state, which results in lower intra-distance values. The same effect is achieved by slower ramp-up of the supply voltage. In contrast to inter-distance metrics, Fig. 8.9 shows no signs of ramp-up time 'saturation'.

However, we can make a few observations regarding Fig. 8.9. Firstly, the effect of IC ageing seems to be amplified as the ramp-up becomes slower. Unfortunately, the experimental data is not sufficient in producing a clear image of this effect, and further investigation would be required in that respect. An additional interesting trend can be discerned in the 25°C operating corners. While, in general, the intra-distance decreases monotonically with higher ramp-up times, in those corners its magnitude seems to follow a different course, increasing to its maximum value at the 'middle point' ramp-up time before starting to decrease.

Moreover, the repeated pattern in every operating corner verifies our earlier intuition that each IC exhibits an intra-distance baseline around which all intra-distance measurements are centred. For example, IC no.9 of the experiment (easily identified at every 'spike' of the curves) consistently presents a higher intra-distance than the rest of the ICs

and seems to be more sensitive to condition variations[3]. Tables A.4 to A.6 provide a more detailed view into the inter-distance measurements.

From the point of view of security protocols, two main observations can be made. Firstly we refer to the overlapping histograms of Fig. 8.6: an intra-distance of about 30% would be devastating for any authentication or identification protocol. Nevertheless, such high values are only obtained when comparing PUF responses *across* different operating corners. Thus, rather than a disadvantage, this behaviour can be used to provide the novel feature of detecting changes in operating conditions. The value of this feature becomes clear if one reflects on the operating conditions of many digital systems: the voltage supply is designed to be fairly constant, and dramatic changes in ambient temperature are rare. As a result, such a change is likely to indicate a physical attack in certain applications.

Finally, the data of this and the previous section show that an adversary who is capable of tampering with the operating conditions of the system, might be able to induce a high number of false negatives, but cannot do the same with false positives. We found no operating corner where the inter-distance and intra-distance magnitudes overlap or are even remotely close to each other. Even with precise control of either metric through modifying the external conditions, there exists no direct correlation between those conditions and the actual values of the PUF responses. Thus, the adversary would be unable to produce two matching responses from distinct ICs. We can therefore conclude that the examined condition variations do affect the security of the system in authentication and key generation scenarios, but only minor provisions (i.e. appropriate threshold selection) are needed to prevent false positives.

Error Rate

The bit error rate (BER) is derived directly from the intra-distance and thus follows a similar distribution. The BER values in this section are defined as the mean intra-distance over all ICs for a certain operating corner.

Perhaps counter-intuitively, the lowest BER is exhibited in the highest ambient temperature, as seen in Table 8.4. We can also see that the effect of temperature is much less significant for lower ramp-up times, to the point that it even seems to be reversed. However, in the lowest ramp-up times, the measurements are within the standard error[4] and thus this reversal is not considered statistically significant. The influence of both the temperature and the ramp-up time parameters can be associated with the same physical phenomena discussed above.

SRAMs exhibit a relatively low BER, with a maximum of 9.5% in the worst case, a value which is one of the lowest in comparison to other PUF classes[10]. However, security protocols often involve algorithms with avalanche effects, where a single bit variation in the input produces radically different outputs. As a result, bit errors need to be completely

[3]See Footnote 2.

[4]The standard error of the mean (SEM) for a sample size n is defined as $SEM = \frac{s}{\sqrt{n}}$ where s is the sample standard deviation.

eradicated for SRAMs to be a viable security building block. Error correction methods were discussed in Chapter 7, but their implementation complexity increases exponentially with the number of bits to be corrected. It is therefore in our best interest to achieve the lowest possible BER, before employing any error correction.

An interesting side-effect of NBTI can be observed in Fig. 8.11: the tendency of the cells to shift away from their previous value drives the intra-distance between sequential evaluations of the SRAM to its *maximum magnitude*. The intra-distance (and thus the BER) between observations i and j (with $j > i$) was found to be 30% lower when $j-i = 2$ compared to cases of $j - i = 1$. For $j - i > 2$ a similar behaviour continued, with the intra-distance 'oscillating' between higher values for $j-i = 2k+1, k \in \mathbb{Z}$, and lower values for $j - i = 2k, k \in \mathbb{Z}$, as summarised in Table 8.5. As a result, simply skipping a single observation (or an odd number of observations) when generating SRAM PUF responses would substantially improve the BER of the responses, regardless of the operating corner.

	500ns	**5us**	**50us**	**500us**	**5ms**	**50ms**	**500ms**
25°C	8.28 %	8.41 %	9.24 %	9.30 %	8.35 %	6.75 %	4.83 %
25°C aged	8.18 %	8.43 %	9.43 %	9.51 %	8.81 %	7.28 %	5.24 %
100°C	8.60 %	8.60 %	8.41 %	7.07 %	5.91 %	4.85 %	3.60 %
100°C aged	8.71 %	8.70 %	8.53 %	7.43 %	6.30 %	5.12 %	3.72 %

Table 8.4: BER in all operating corners

j-i	**Mean Intra-distance**	**Absolute Difference**	**Relative Difference**
1	12.07 %	-	-
2	7.92 %	-4.1 %	-34.3 %
3	9.91 %	-2.1 %	-17.8 %
4	8.69 %	-3.3 %	-27.9 %
5	9.52 %	-2.5 %	-21.0 %
6	8.93 %	-3.1 %	-26.0 %
9	9.33 %	-2.7 %	-22.6 %
10	9.09 %	-2.9 %	-24.6 %
30	9.24 %	-2.8 %	-23.4 %
31	9.28 %	-2.7 %	-23.1 %
49	9.30 %	-2.7 %	-22.9 %
50	9.29 %	-2.7 %	-23.0 %

Table 8.5: Sequential intra-distance in nominal conditions (25°C, $t_{ramp} = 500us$)

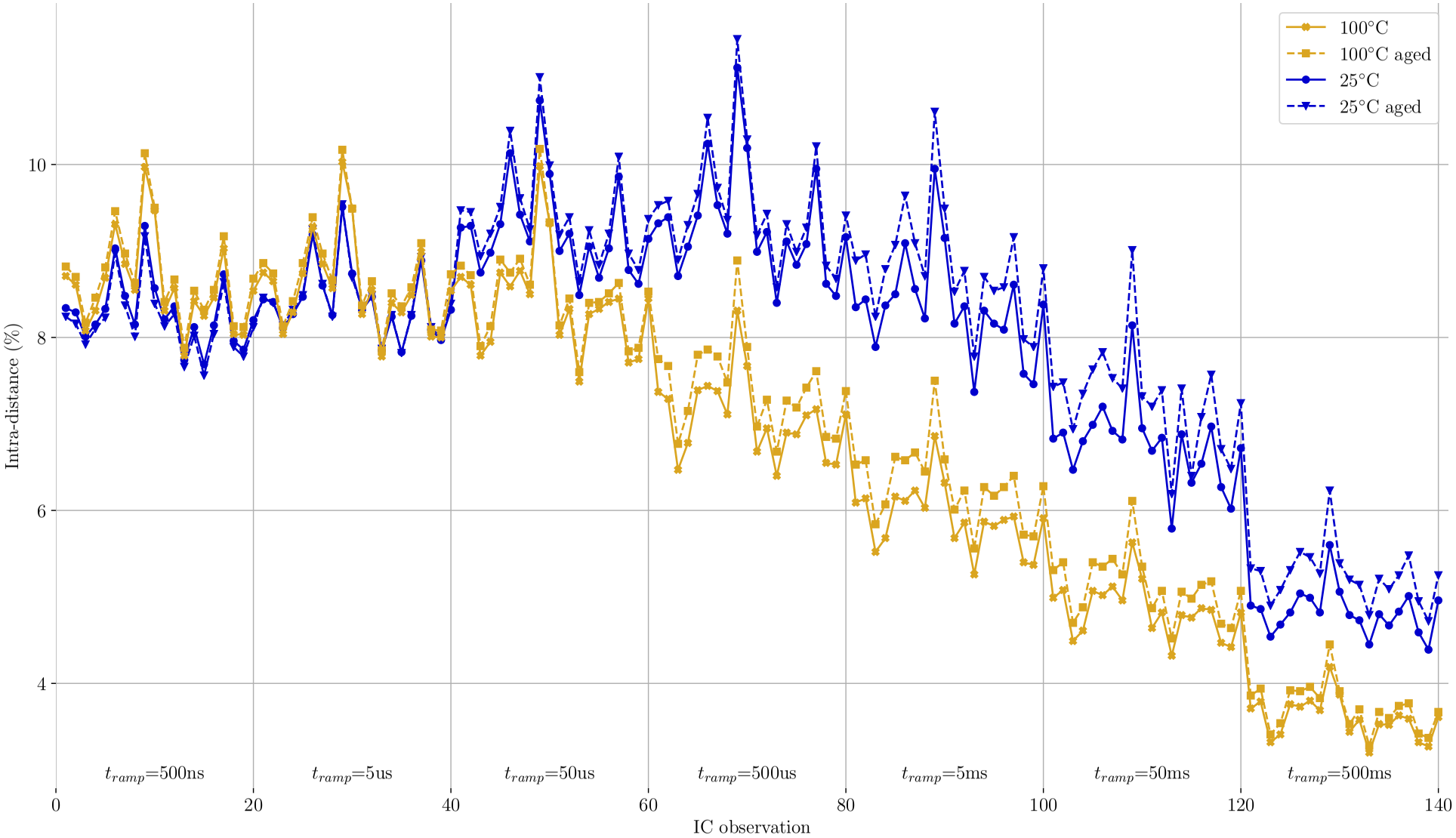


Figure 8.9: Mean intra-distance by IC and operating corner

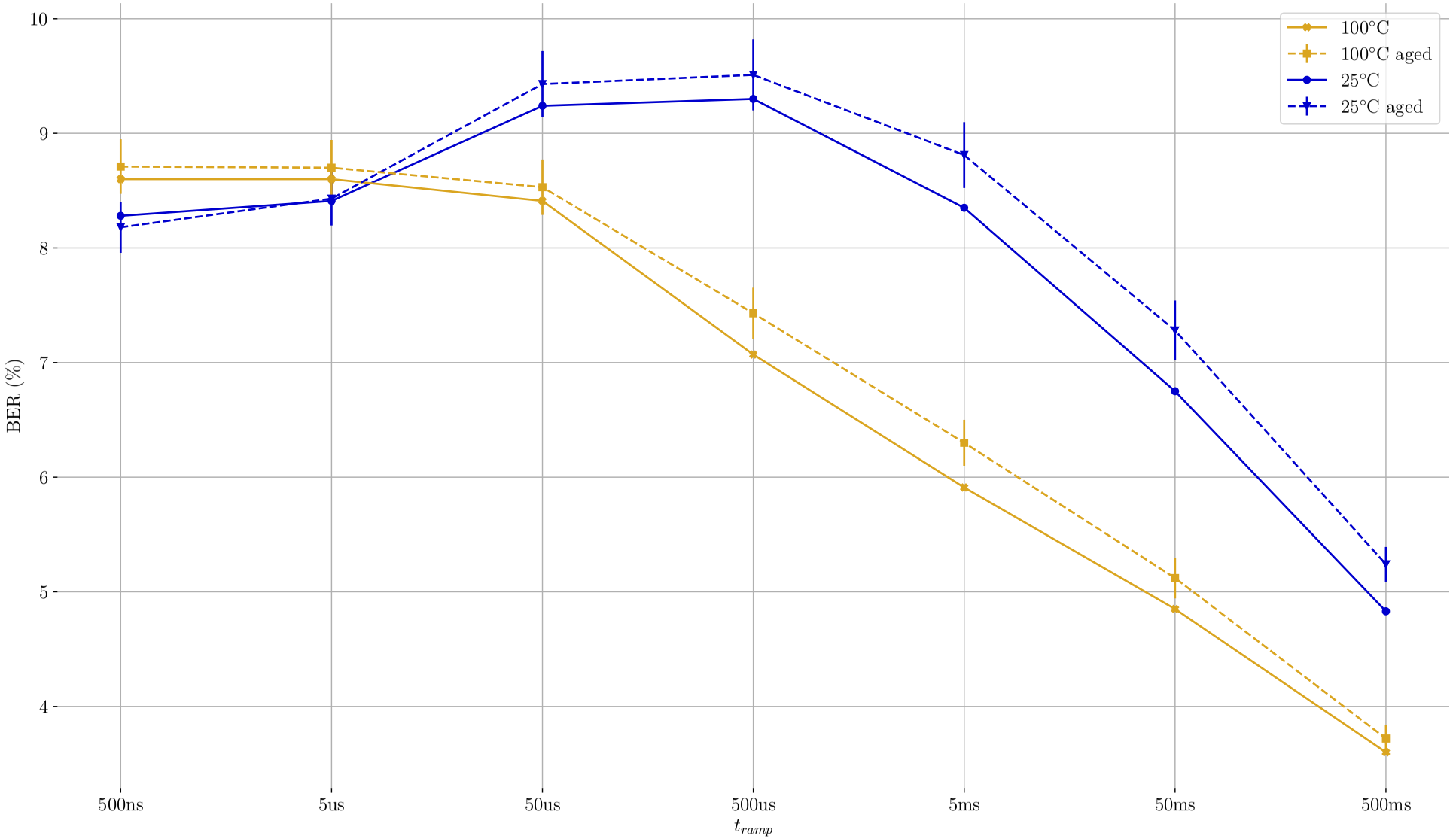


Figure 8.10: BER in all operating corners

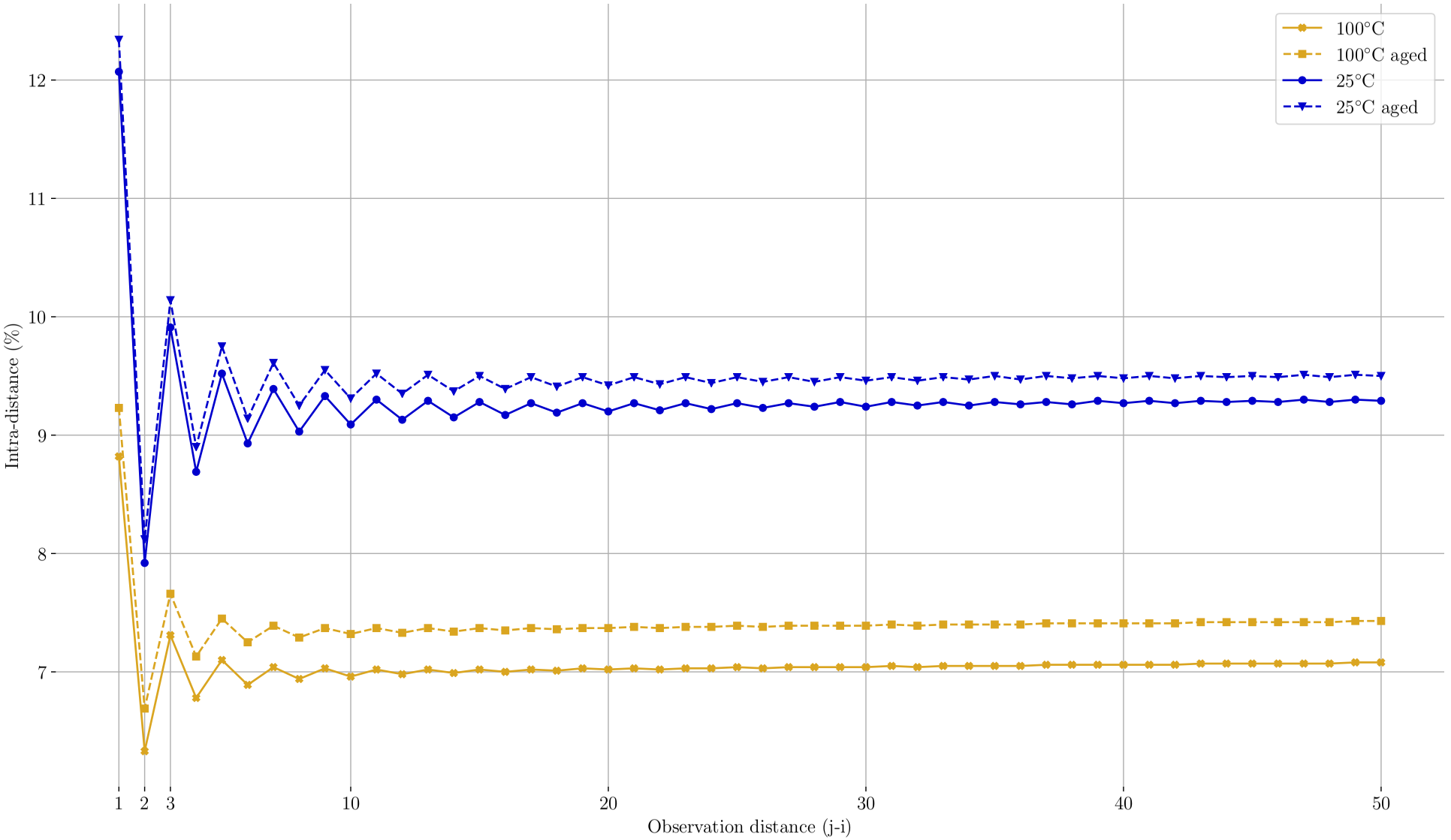


Figure 8.11: Sequential intra-distance in nominal ramp-up time ($t_{ramp} = 500us$)

8.5.3 Entropy

Intuitively, entropy seems to be in direct opposition to the common goal of PUFs: identification. However, the entropy or *unpredictability* of a portion of the SRAM cells is what gives SRAMs their unclonable nature. This is because the entropy is rooted in the very property of unpredictable physical variations that cannot be meaningfully controlled. In essence, a digital quantity (i.e. the value of an SRAM cell) is said to have *full entropy* when, given all the previous observations of the quantity, it is not possible to guess its value in future observations with a probability higher than 50%. In practice, entropy is often described as 'high' or 'low' entropy, although the definition of those terms is dependent on the context.

From a PUF perspective, entropy is a double-edged sword. Despite the aforementioned advantages, it also creates the requirement for costly error correction methods. On the other hand, harnessing this entropy allows for the generation of high-quality randomness, which is a fundamental part of most cryptographic schemes. As we discussed in Chapter 7, the conflicting design goals of stability and randomness necessitate the careful selection of PUF blocks to fulfil these goals. Thus, the entropy of a given PUF needs to be quantified and analysed.

In our analysis we use two metrics, defined in Section 8.3. The *min-entropy* is a pessimistic measure of the randomness provided by a PUF. Due to this very characteristic, it is often used in RNG designs, where conservative choices are required. Secondly, we evaluate the dispersion of unstable cells throughout the SRAM ICs, using a metric which we call *relative distance deviation (RDD)*. This metric allows us to assess the spread of entropy in the IC, and identify potential clustering of unstable cells, a factor that should be taken into consideration for entropy extraction methods.

Min-entropy

The min-entropy of a single cell can be calculated with Eq. (8.8) via calculating the bias of the cell over multiple observations of its power-up value. Under the assumption that cell power-up values are independent, the total min-entropy of an SRAM PUF response is the sum of the min-entropy of the underlying cells. Van der Leest et al. showed in [57] that 100 observations are sufficient for an accurate min-entropy estimation, and thus we were able to derive the metric for all our selected operating corners.

In Table 8.6 and Fig. 8.12, we present the min-entropy measurements in every operating corner. Since we are interested in the worst case, the min-entropy per bit is individually calculated for each IC, and the minimum value among the 20 ICs is selected. Thus, the actual entropy of the SRAM power-up values is guaranteed to exceed the reported values. Evidently, the shape of the min-entropy curves (Fig. 8.12) is very similar to that of the BER curves (Fig. 8.10), a fact that verifies the intuitive notion that entropy and noise are closely related for SRAM PUFs. The min-entropy decreases as the ambient temperature or the ramp-up time rise, for the reasons discussed in Section 8.5.2.

As discussed in Chapter 7, entropy extraction and accumulation methods are used in TRNG designs to harness PUF randomness, and transform it into full-entropy seeds, commonly used as inputs for PRNGs. The min-entropy measurements allow us to establish several facts about SRAM behaviour in the context of entropy extraction. Firstly, the inherent randomness of SRAM PUFs is greatly affected by the external conditions studied, but nevertheless remains above an acceptable threshold. Thus, designing a TRNG based on the worst case entropy value (around 4% from our measurements) will guarantee the correct operation of the generator regardless of external influences.

At the same time, using an appropriately selected (usually shorter) ramp-up time can increase the entropy content to more than double its magnitude. This observation is important, since in practical applications controlling the ambient temperature is rarely possible. However, the supply voltage curve can be controlled deterministically with dedicated circuitry. Despite its overhead, such a method allows the generation of full-entropy seeds from a limited number of SRAM cells, thus resulting in the reduction of the silicon area required for a TRNG implementation. Assuming a linear relationship between the silicon area and the number of SRAM cells, an increase of the min-entropy from 4% to 11% would reduce the required silicon area by over 60%, while retaining full-entropy.

Finally, the min-entropy showed an increase with ageing, effectively improving the performance of an SRAM-based TRNG over time. While the entropy gain is not significant enough to warrant a forced ageing process before deployment, it does provide an additional guarantee for the correct operation of the TRNG over time. The slightly lower entropy values for aged ICs in the lowest ramp-up times are not considered statistically significant since the difference is well within the standard measurement error.

	500ns	**5us**	**50us**	**500us**	**5ms**	**50ms**	**500ms**
25°C	0.102	0.104	0.112	0.111	0.101	0.083	0.058
25°C aged	0.101	0.104	0.114	0.114	0.105	0.089	0.062
100°C	0.100	0.100	0.097	0.081	0.070	0.058	0.042
100°C aged	0.102	0.101	0.099	0.085	0.074	0.060	0.043

Table 8.6: Min-entropy per bit in all operating corners (minimum over 20 ICs)

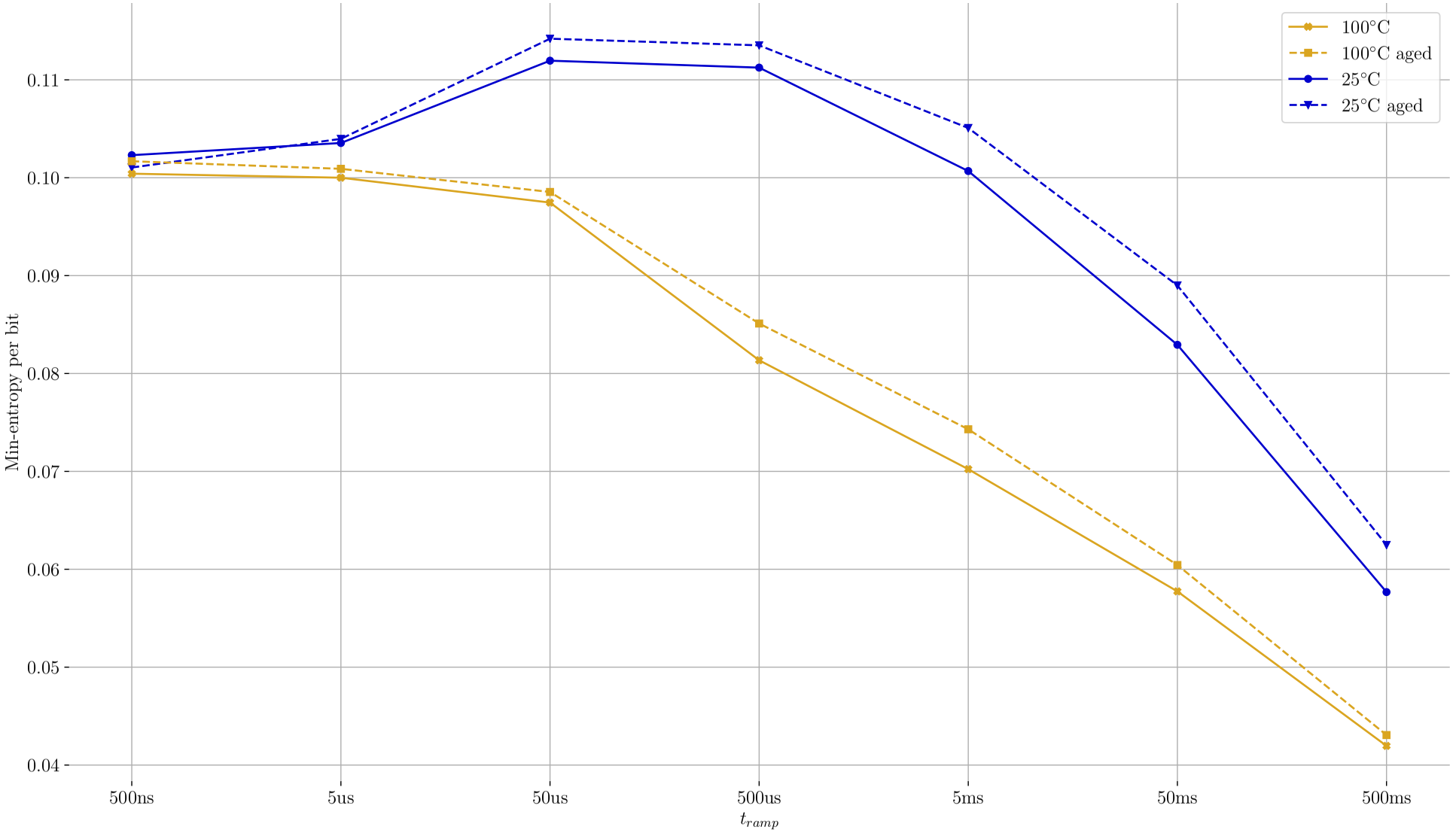


Figure 8.12: Min-entropy per bit in all operating corners (minimum over 20 ICs)

Dispersion of Unstable Cells

The randomness guarantees provided in the previous section are based on an estimation of the mean entropy for entire ICs. However, a different aspect of cell instability, their physical location on the IC, is also highly relevant in TRNG designs. To the best of our knowledge, previous work examining SRAM PUFs in the context of randomness generation assumes that the unstable cells providing this randomness are uniformly distributed. Based on this assumption, it is presumed that any random group of cells exhibits the same entropy content.

Referring to Fig. 8.14, it is clear that the proportion of highly unbiased (and thus unstable) cells is normally limited to less than 1% (detailed bias data is available in Table A.7). Therefore, our intuition is that clustering of unstable cells is possible. In order to quantify such behaviour, we use the RDD metric defined in Section 8.3. The advantage of the RDD is that it does not depend on the total number of unstable cells, which varies based on the conditions and the IC. Using the cell bias measurements, we select cells where $1\% \leq bias \leq 99\%$ and calculate the deviation of their distance from the uniform distance[5].

For uniformly distributed unstable cells, the distance between any pair of cells is equal, thus given by Eq. (8.11) as the size of the IC divided by the total number of unstable cells. Thus, the RDD of any pair of unstable cells, based on Eq. (8.12) can range from -1 for consecutive cells, to (in theory) the total number of cells in an IC. Negative RDD values signify distances that are smaller than the uniform, which in turn shows higher clustering, while positive values indicate reduced clustering which decreases as RDD increases.

Statistical evaluation of RDD values for the entire IC allows us to paint a picture of the unstable cell dispersion. In the 'ideal' case of fully uniformly distributed unstable cells, the RDD values would be overwhelmingly concentrated around zero. On the contrary, the measurements of Table 8.7 show that 50% of the unstable cells have an RDD of around 0.3 which means that the distance between those cells is 30% smaller than the uniform distance[6]. This indicates that the unstable cells are clustered somewhat closer together than what is normally expected. Indeed, Fig. 8.13 shows that over two thirds of the distance observations are in the negative domain, and only a negligible portion (0.3%) is equal to the uniform distance.

Further analysis of the RDD data provides insights into its detailed distribution pattern. We examine the negative and positive values separately, for convenience. In the negative domain, shown in Fig. 8.15, the RDD observations are fairly uniformly distributed with the only larger spikes very close to -1, indicating heavily concentrated clustering for approximately 1.6% of the observations. On the other hand, the vast majority of positive observations is concentrated towards the zero point, a behaviour that is also illustrated

[5]The distance between two cells is defined as the number of cells separating them. Consecutive cells have a distance of 0.

[6]On the significance of RDD decimal digits: the uniform distance between unstable cells ranges from 100 to 400. Thus, three decimal digits are sufficient to represent RDD values, since a cell distance value which is less than 1 has little meaning.

by the 99%-percentile data seen in Table 8.8.

Figs. 8.13, 8.15 and 8.16 refer to the nominal conditions. Nevertheless, it is clear from Tables 8.7 and 8.8 that the clustering behaviour of cells is very similar in every operating corner, and we can conclude that the operating conditions do not significantly affect the dispersion of unstable cells.

In summary, the evaluated SRAM ICs show significant clustering of their unstable cells, which is not however deviating strongly from the uniform distribution. Additionally, no discernible pattern was found in the distribution of unstable cells. Both conclusions are further indications of the robustness of SRAM PUFs, since a fault in a limited area of the IC, whether it is transient or permanent, is unlikely to significantly affect the unpredictability of the PUF. This robustness is improved even further by methods such as the ones proposed in the previous chapters, which utilise different parts of the PUF over time.

	500ns	**5us**	**50us**	**500us**	**5ms**	**50ms**	**500ms**
25°C	-0.314	-0.313	-0.315	-0.312	-0.316	-0.312	-0.311
25°C aged	-0.314	-0.316	-0.316	-0.316	-0.314	-0.313	-0.312
100°C	-0.313	-0.313	-0.314	-0.311	-0.311	-0.313	-0.309
100°C aged	-0.315	-0.314	-0.312	-0.314	-0.311	-0.312	-0.313

Table 8.7: Median relative distance deviation for cells with $1\% \leq bias \leq 99\%$

	500ns	**5us**	**50us**	**500us**	**5ms**	**50ms**	**500ms**
25°C	3.580	3.588	3.589	3.580	3.602	3.580	3.594
25°C aged	3.593	3.576	3.593	3.578	3.585	3.584	3.600
100°C	3.558	3.579	3.586	3.584	3.597	3.596	3.579
100°C aged	3.592	3.571	3.572	3.578	3.593	3.603	3.595

Table 8.8: 99%-percentile of the RDD for cells with $1\% \leq bias \leq 99\%$

	500ns	**5us**	**50us**	**500us**	**5ms**	**50ms**	**500ms**
25°C	12.242	13.092	11.089	11.393	14.105	13.331	11.980
25°C aged	13.582	14.528	16.424	13.322	13.510	11.888	12.655
100°C	12.162	12.848	15.744	11.415	12.663	12.687	11.377
100°C aged	11.849	12.474	12.026	12.656	14.915	11.000	11.460

Table 8.9: Maximum relative distance deviation for cells with $1\% \leq bias \leq 99\%$

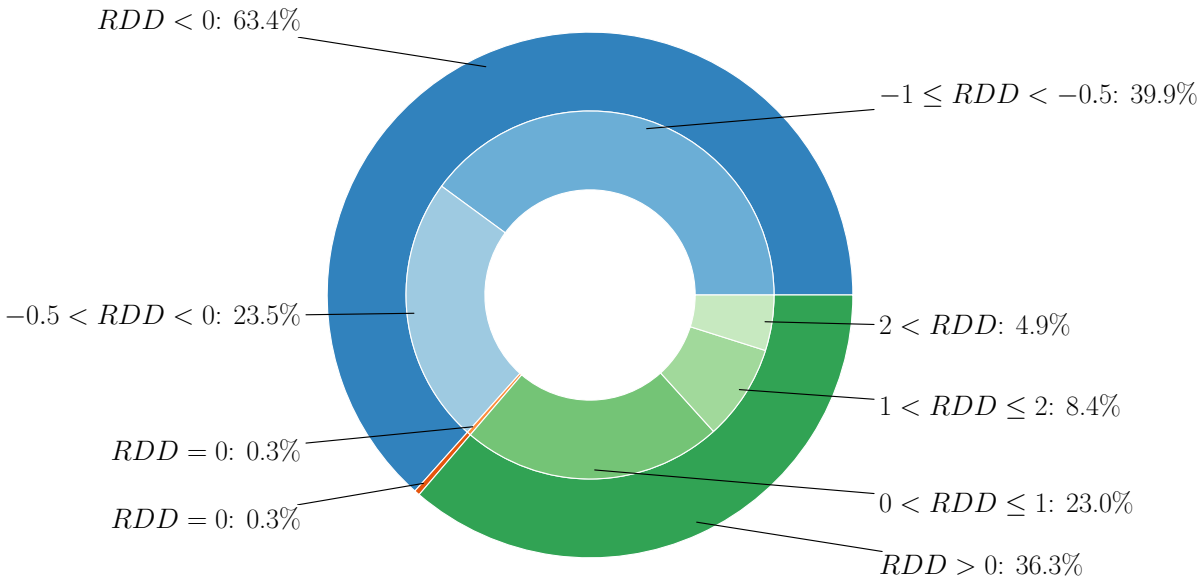


Figure 8.13: Distribution of RDD for cells with $1\% \leq bias \leq 99\%$ (25°C, $t_{ramp} = 500us$)

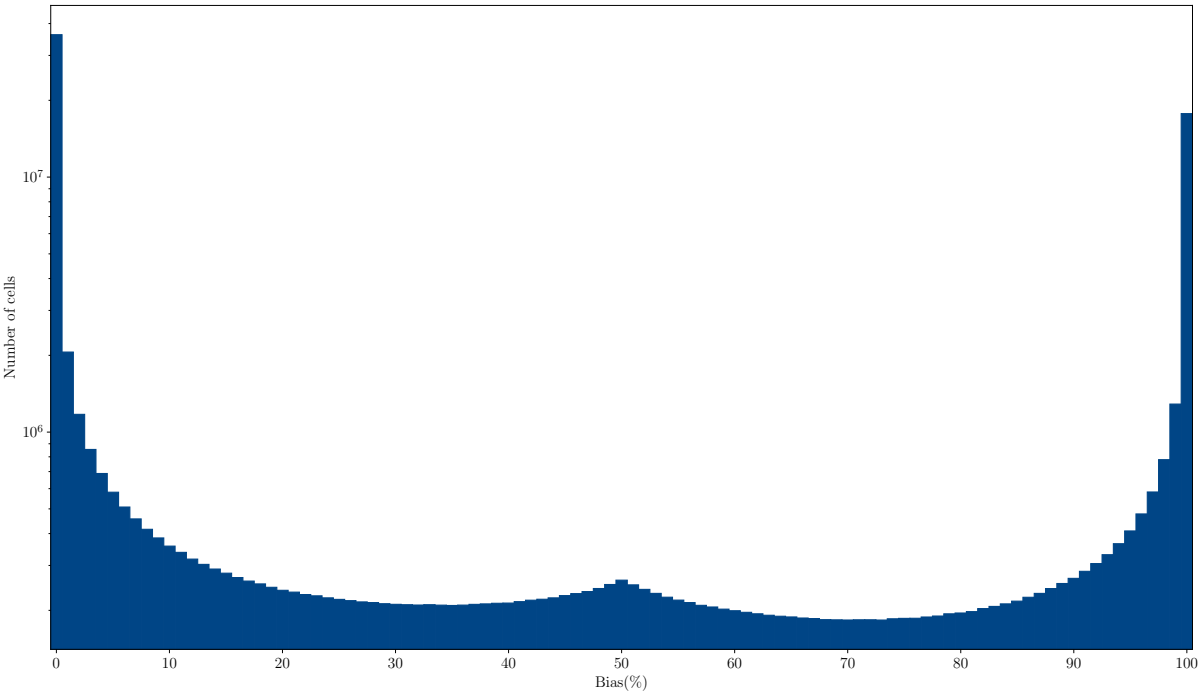


Figure 8.14: Bias distribution (20ICs, 25°C, $t_{ramp} = 500us$)

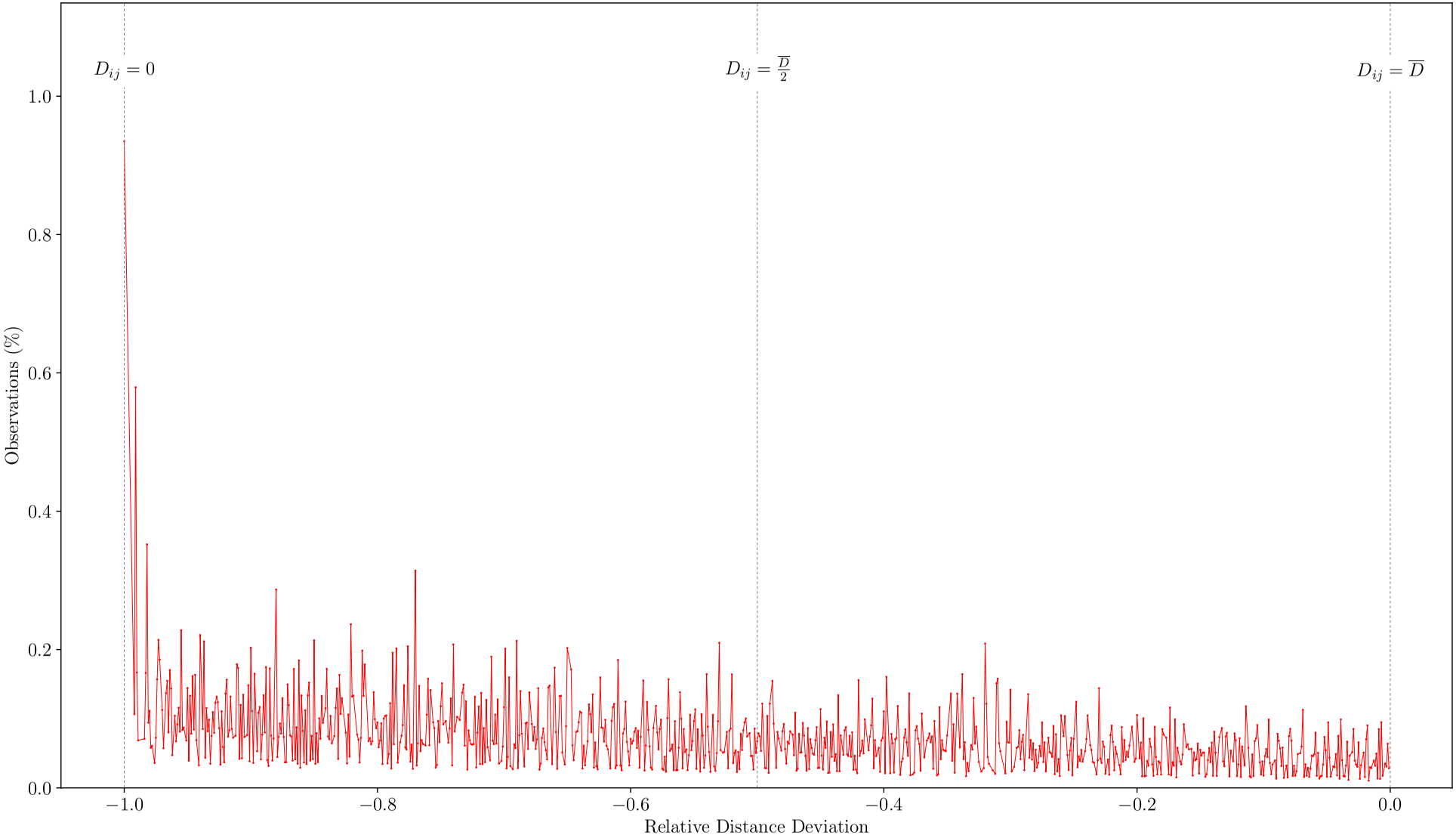


Figure 8.15: Distribution of negative RDD values for cells with $1\% \leq bias \leq 99\%$ (25°C, $t_{ramp} = 500us$)

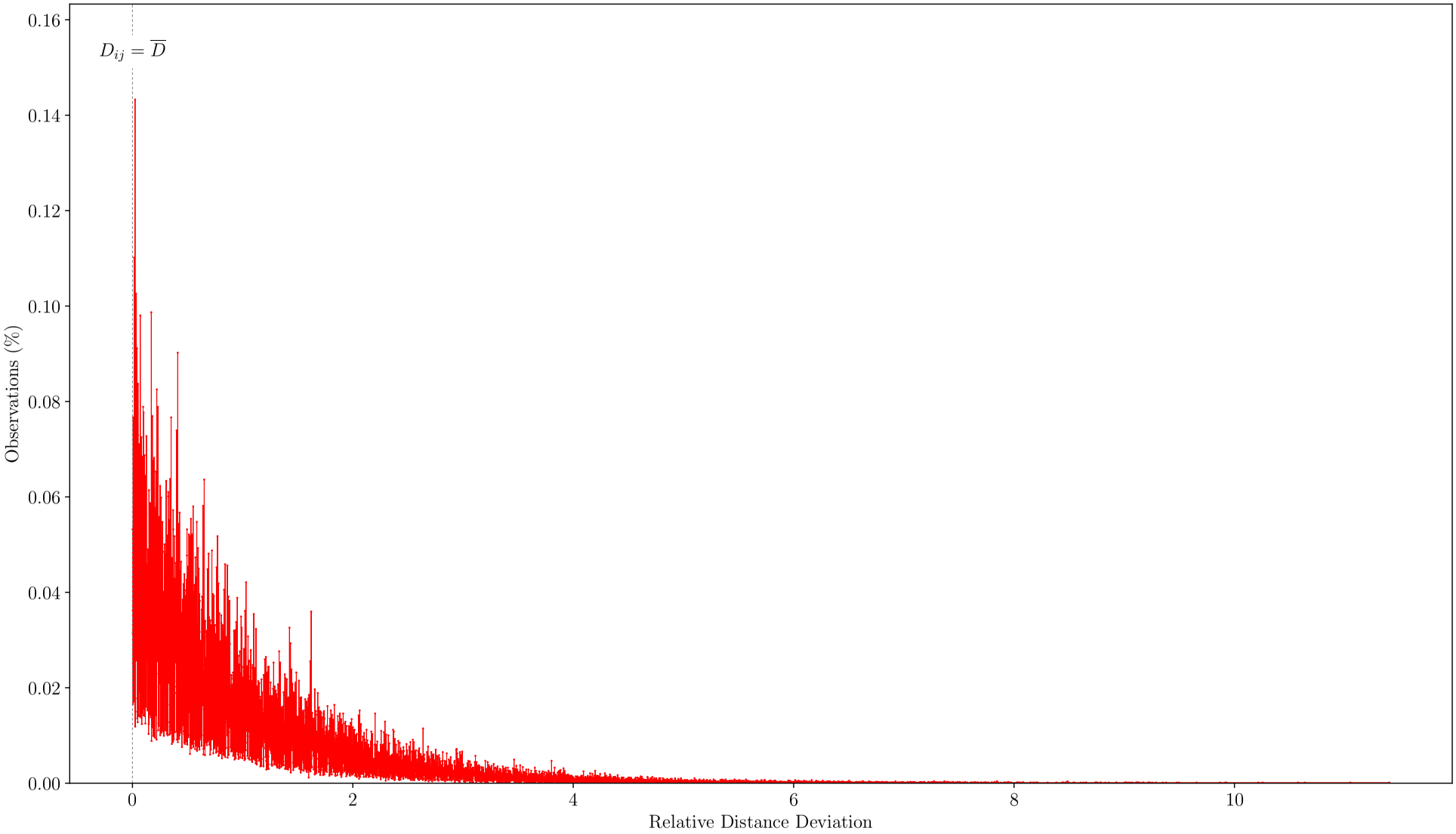


Figure 8.16: Distribution of positive RDD values for cells with $1\% \leq bias \leq 99\%$ (25°C, $t_{ramp} = 500us$)

8.6 Conclusion

In this chapter, we have presented a detailed discussion of SRAMs in the context of their use as PUFs, using experimental data in 28 operating corners, examining various ambient temperatures and supply voltage ramp-up speeds. Our analysis of almost 2 billion data points showed that SRAM PUFs conform to the PUF model that we defined in Section 4.3.1, with high inter-distance and low intra-distance. Additionally, we quantified and evaluated the distribution and behaviour of unstable SRAM cells, and their effect on the entropy content of the PUF. We also reached the conclusion that the examined SRAM ICs retained their high-quality PUF properties regardless of external influences. This is particularly useful in practical applications, making SRAM PUFs a robust option as both identity providers and randomness generators.

An interesting direction for further work is the investigation of the effect of IC geometry on the metrics discussed in this chapter. While the SRAM cells are often assumed to be arranged in a linear manner, this is not the case for COTS components. Thus, correlations between neighbouring cells, or other clustering effects can potentially be discovered by such an investigation.

9. Proof-of-Concept Implementation

9.1 Introduction

We modelled the protocols, methods, and architectures discussed in the previous chapters with a software implementation. For a framework with as many aspects as the ones described in this thesis, software prototyping is imperative for the evaluation and refinement of protocol logic and implementation details. In comparison to a hardware prototype, the development was faster and cheaper, allowing for easy iterative corrections as our research and understanding of the intricacies of such implementations progressed.

Our intuition as we began constructing this model was that it will uncover shortcomings that would be overlooked in the protocol-level analysis of Chapters 5 and 6. Indeed, the insights stemming from the model had broad implications on our protocols and therefore both the protocols and the model ended up being developed in parallel.

In the following sections we discuss the architecture of the software model as well as specific implementation choices for the many parameters of the system. As a result, this chapter, along with Chapter 7 can be used as blueprints to directly transfer our work to the hardware domain, with minimal additional effort. To that end, the implementation choices seen in this chapter follow current best practices and security guidelines. We took great care to reduce the overhead and related costs of the implementation but, unsurprisingly, increased security comes at a cost.

A hardware prototype is the natural next step, thus being a priority for future work. However, we were able to take advantage of the SRAM measurements and the resulting insights of Chapter 8 to effectively emulate the PUF block.

Notation

We use common notation to describe binary quantities. A string of bits with length m is denoted as $\{0,1\}^m$. Similarly, m-bit strings where all the bits have the value 0 or 1 are denoted as $\{0\}^m$ or $\{1\}^m$ respectively.

9.2 Software Model

9.2.1 Overview

The complete functionality of the cryptocore as well as the protocols of Chapters 5 and 6 were implemented in software[1], using Python 3[187]. Python made a good candidate for rapid prototyping due to its extensive library support. We were able to use the

[1]The full source code is available upon request and may be made publicly available in the future.

PyCryptodome library[188] to provide the basic cryptographic building blocks: encryption, signature and hashing. Nevertheless, the library does not support zero knowledge proofs, lightweight hash functions, or ECC encryption, and thus we developed our own implementations.

Fig. 9.1 gives an overview of the building blocks of the system, modelled with Python objects. In an architecture closely resembling the one of Section 2.4, nodes and authority devices were both derived from a common 'Device' class, due to their identical hardware structure. Importantly, every device contains an embodiment of the cryptocore discussed in Chapter 7.

A virtual system of nodes was built on the Linux Kernel Virtual Machine (KVM)[189] virtualisation platform. KVM was selected for the ability to be extensively controlled through a command line interface, allowing for the automated creation of various topologies. Pure TCP was used for communication infrastructure between all devices, using Python's built-in TCP library.

Protocol information was encoded with a custom TCP packet structure, which we call a *message*. Each message carried a header which included the current 'phase' and 'command' of the protocol being executed, as well as auxiliary flags and length information, detailed in Fig. 9.2. The auxiliary flags comprise an initialisation (INIT / I) and final (FIN / F) flag representing the first and last steps of the current protocol respectively. Accordingly, the communicating parties make use of the header information to advance their state and detect potential deviations of the protocol. A summary of the header values is given in Table C.1.

The exchanged messages carried varying information, from device identifiers and public keys, to PUF CRPs. In addition to the protocol interactions shown in their respective definitions in previous chapters, a few supplementary messages were necessitated by the practicalities of the implementation. These messages however constituted a negligible portion of the total network traffic. Table 9.1 provides an overview of the various data objects used throughout our implementation, with more details and the supporting rationale given in the following sections.

9.2.2 Physical Unclonable Function

The PUF component was emulated using the data acquired from the SRAM ICs (see Chapter 8). The entire contents of the 20 ICs were read and stored in a binary format. This process was repeated 100 times in each operating corner, resulting in 2800 samples, each containing $20 \times 512Kb = 10Mb$ of PUF power-up state. In order to model a realistic scenario we only made use of the data of one IC at a time, since multiple SRAMs are rarely found in COTS devices. Nevertheless, multiple ICs are also supported by the software model.

Every time the software is initialised, one of the ICs is selected at random to serve as the PUF. Upon query of the PUF, the relevant data was read from one of 100 data

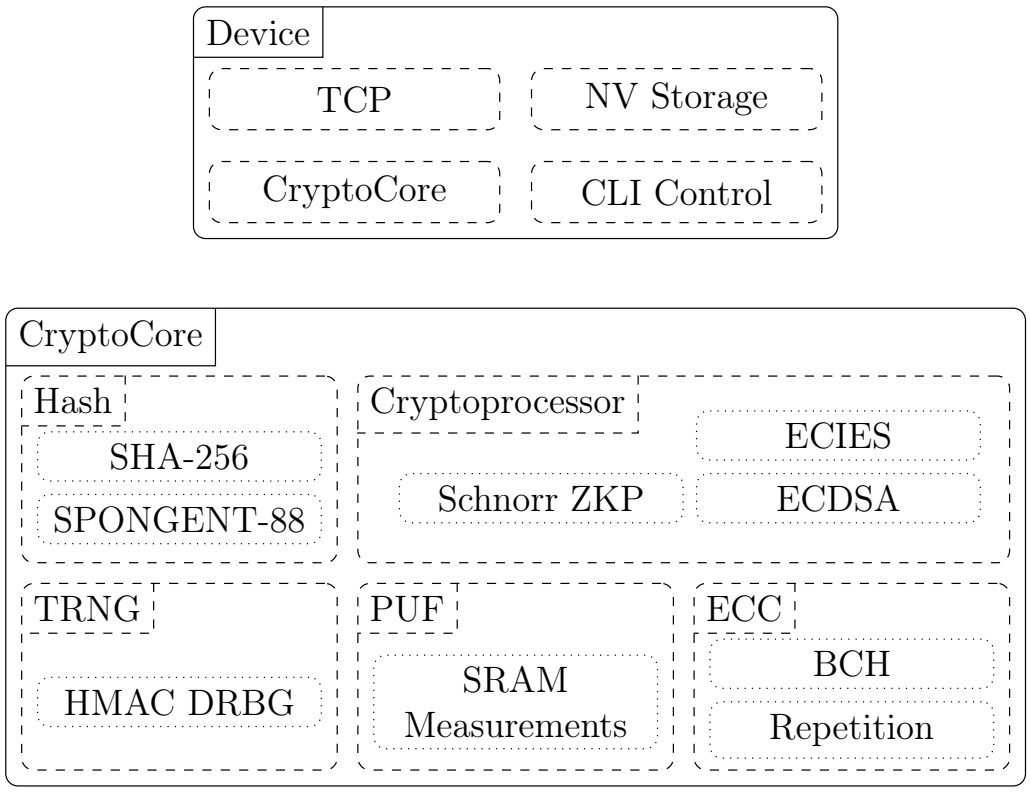


Figure 9.1: Model object architecture

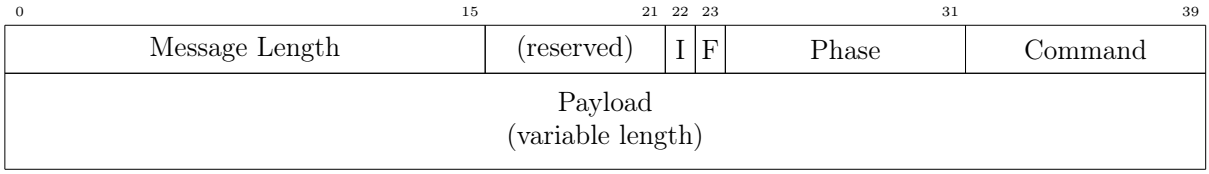


Figure 9.2: Message structure

files and returned as a response. The sample files were used in the order which they were initially created, thus emulating the statistical properties of the PUF responses over time.

The SRAM ICs have 18 address pins, accessing a total of 256K 16-bit memory locations. In order to interface with this configuration, challenges are divided into addresses of 18 bits addressing a single memory location each. After the optional error correction step, the concatenation of the memory contents is hashed and returned as the final response. This versatile generation method allows the PUF block to accept challenges of arbitrary length supporting different applications. For example, in the case of random seed generation, the error correction step is skipped and a large number of 16-bit long memory addresses is concatenated and compressed by the hash function as described in the next sections.

For the remainder of this chapter, it is assumed that the length of challenges and responses is constant over the lifetime of the system. The length of the PUF responses is determined by the digest length of the selected hash function, which in our implementation was SHA-256. The CRP Ratchet protocol (see Chapter 6), requires challenges and responses to have the same length, thus both quantities were set at 256 bits. Con-

Data Object	Size in bits
IDs	8
Random Nonces	64
PUF challenges	256
PUF responses	256
BCH helper data per response	60
ECDSA public keys	472
ECDSA signatures	512
Hash digests	256
HMAC tags	256
ZK challenges	64
ZK commitments	256
ZK proofs	264

Table 9.1: Sizes of common data objects

sequently, each PUF response is generated by a concatenation of 15 memory locations, before hashing.

The entropy requirements, which directly determine the required PUF bits, for the methods proposed in previous chapters are as follows:

RNG seed generation This case bears the strict requirement of responses with full entropy. Based on the process described in Section 7.3.6, at least 8534 PUF bits are required to provide that entropy level. Thus, the power-up state of 534 memory cells ($\lceil \frac{8534}{16} \rceil$) is concatenated and hashed to produce the required seed. Error correction is not needed and thus no entropy is leaked through helper data.

Key seed generation The key seeds used by the Authority Device Scheme (see Algorithm 5.1) have the same entropy requirements as above. However, in this case error correction is required, resulting in an entropy loss through helper data, as discussed in the next section. Therefore, the content of $\lceil \frac{8534}{16} \times \frac{240}{116} \rceil = 1104$ memory cells are concatenated to produce a full entropy 256-bit output.

Entity identification In the context of the ADS, PUF CRPs are used for simple entity identification. The scheme design dictates that the identification process is executed only once during enrolment. For slightly modified applications where multiple identifications are needed, the required entropy can be generated as described above.

CRP ratchet The entropy requirements are far more limited in the case of CRP ratchets, since the responses are XORed with full entropy random challenges before being transmission, hindering modelling attempts.

ZK ratchet The nature of zero knowledge proofs does not require the committed quantities, which are the PUF responses, to be random.

9.2.3 Error Correction

The error correction block was based on the concatenated EC code method introduced in Chapter 7. Raw PUF responses were first processed with a repetition code (also referred to as *Temporal Majority Voting*[190]), essentially collecting multiple responses to any given challenge and using majority decision to generate a single intermediate response with a lower BER. The process, described in Algorithm 9.1, has a significant effect on the BER, as seen in Fig. 9.3. Subsequently, a BCH code was used to correct the remaining errors.

For the proof-of-concept implementation, we used a $[n_{bch} = 7, k_{bch} = 11, t_{bch} = 3]$ binary repetition code, followed by a BCH code with $[n_{bch} = 240, k_{bch} = 124, t_{bch} = 15]$[2]. For the purposes of the demonstration, helper data was not deleted. However, for certain classes of protocols such as the ones presented in Chapter 6 it is possible to reduce the lifetime of the helper data leading to considerable storage savings.

Algorithm 9.1 (PUF Repetition Error Correction)**.**

```
1:  procedure GenerateResponse(challenge, n_rep)
2:      responses = [ ]
3:      for all i ∈ [1, n_rep] do
4:          responses[i] = PUF(challenge)
5:      return Majority(responses, n_rep)
6:
7:  procedure Majority(responses, n_response)
8:      output = {0}^m
9:      for all i ∈ [1, n_response] do
10:         for all j ∈ [i+1, n_response] do
11:             output = BitwiseOR(output, BitwiseAND(responses[i], responses[j]))
12:     return output
```

9.2.4 Hash Function

There exist numerous practical implementations of hash functions as they were described in Section 3.3.2. As with most applied cryptography methods, highly regarded hash functions quickly fall from grace when vulnerabilities in their algorithm are discovered or technological advances render them obsolete. In practice, the safest choice is to use a well-established and thoroughly audited construction. Arguably, the most widely used choice is the SHA-256 variant of the SHA2 family[118]. The SHA3 family followed the example of its predecessor but is based on a radically different foundation, disallowing, among others, length extension attacks[120]. BLAKE2[119] and Whirlpool[191] have also been used in practice.

[2]Codeword shortening allows the construction of codes with $[n_{bch} = 2^u - 1 - s, k_{bch} - s, t_{bch}]$.

The aforementioned solutions are not optimised for lightweight operation which would be desirable in embedded systems applications. Therefore, several works have recently targeted the fairly unexplored field of lightweight cryptographic functions. The most notable efforts include SPONGENT[192], PRESENT[193], PHOTON[194], and QUARK[195]. All of them are based on the so-called 'sponge' construction which is also used in the SHA3 family[196]. SPONGENT has been the preferred hash function for several PUF applications, mainly due to its simplicity and support for a wide range of configuration parameters.

For our implementation we chose to evaluate SHA-256 and SPONGENT-88/176/88. We implemented SPONGENT-88/176/88, hereon referred to as 'SPONGENT-88', based on its original specification in [192].

9.2.5 Message Authentication Code

For the message authentication code we used HMAC-SHA256, taking advantage of the existing hash function and XOR logic. Such is the simplicity and robustness of HMACs in general that not much can be said about any particular implementation. The main advantage of HMAC compared to other algorithms is its immunity against attacks that affect the underlying hash function, including length extension attacks[146]. HMAC can process inputs of arbitrary size, while its output has a fixed size equal to the digest length of the underlying hash function, which was 256 bits for SHA-256.

9.2.6 Random Number Generator

As discussed in Chapter 7, for our random number generator we used the HMAC DRBG primitive proposed by NIST[51], seeded by the PUF block. NIST is widely accepted as one of the leading authorities on cryptographic standards, and our choice of the HMAC DRBG over the other algorithms proposed in [51] was motivated by the fact that an HMAC block was already present in our implementation. The two parts of the RNG effectively share the same SHA-256 foundation. The hash function is used as an entropy accumulator during seed generation as well as in the context of the HMAC algorithm. For the random number generation process we refer the reader to Chapter 7, and Algorithm 7.1 in particular.

9.2.7 Asymmetric Cryptography

The main contenders for the encryption and signature implementations were the RSA cryptosystem and cryptosystems based on ECC, as discussed in Section 3.3. In the context of IoT, an Elliptic Curve algorithm is preferable since it is generally faster and requires a much shorter keys to achieve the same security level. As an example, for our chosen parameters the RSA key is 30 times longer than the ECC key for the same level of security.

Having chosen the algebraic basis, the selection of the particular algorithms was straightforward, since our main goal was to reduce the implementation overhead by re-

using existing logic. The only realistic ECC options for encryption and signature were ECIES[197] and ECDSA[115], respectively. For the zero knowledge proofs, Schnorr ZKP was selected, in its ECC configuration[198]. All three primitives were implemented with the same basic ECC operations over the NIST P-256 curve. The choice of curve was motivated both by its prevalence (allowing for easy prototyping) and its security guarantees. According to the relevant NIST guidelines, private keys of 512-bit length for ECC provide the equivalent of 256 bits of security and would be sufficient against the vast majority of adversaries, for at least the next two decades[121].

The ECDSA implementation was provided by the PyCryptodome library, and our ECIES implementation was based on the library's AES implementation. We also implemented the Schnorr ZKP protocol in the ECC setting, based on its specification in RFC8235[198].

Public keys were represented in the binary DER format specified in [199]. This format includes additional information about the algorithm, leading to a slightly larger total size for key representation: 472 bits in compressed form. Evidently, the compressed form has a lower storage and bandwidth overhead but also involves the cost of decompression operation every time the key is used. Since ADS public key operations are used fairly infrequently, we chose to use the compressed form which allowed for reduced storage requirements. Of course, caching of the uncompressed key in volatile memory can also be used when it is frequently needed, for example in the ZK CRP ratchet. Due to the fact that the ratchets are only used with immediate neighbours, the number of cached public keys remains relatively low.

9.3 Discussion

9.3.1 Error Correction

We evaluated a range of repetition cycles and compared the results to the raw PUF responses. Odd repetition values were selected to avoid a potential tie in the majority decision. In addition, binary repetition codes with odd length are also *perfect codes*[200].

As indicated in Fig. 9.3, in all operating corners, even a small number of repetitions dramatically lowers the intra-distance between post-processed responses, achieving a reduction of up to 80%. However, this method quickly reaches its saturation point at around $n_{rep} = 7$ repetitions, offering only incremental improvements past that point. Thus, taking into account the added overhead of each additional SRAM access, the optimal repetition count is in the range of 7 to 11. The above results were similar for all operating corners, as demonstrated in Tables C.3 to C.6 and Figs. C.1 to C.3.

Eq. (9.1)[200] gives the failure probability of an [n, k, t] binary block code, with P_e being the error probability of the input. When multiple EC codes are used, the failure probability of each level is used as the input error probability of the next. We can thus derive the final failure probability of our implementation which is 3.1×10^{-6}, in the worst case. The failure probability for all operating corners can be found in Table C.2. In the

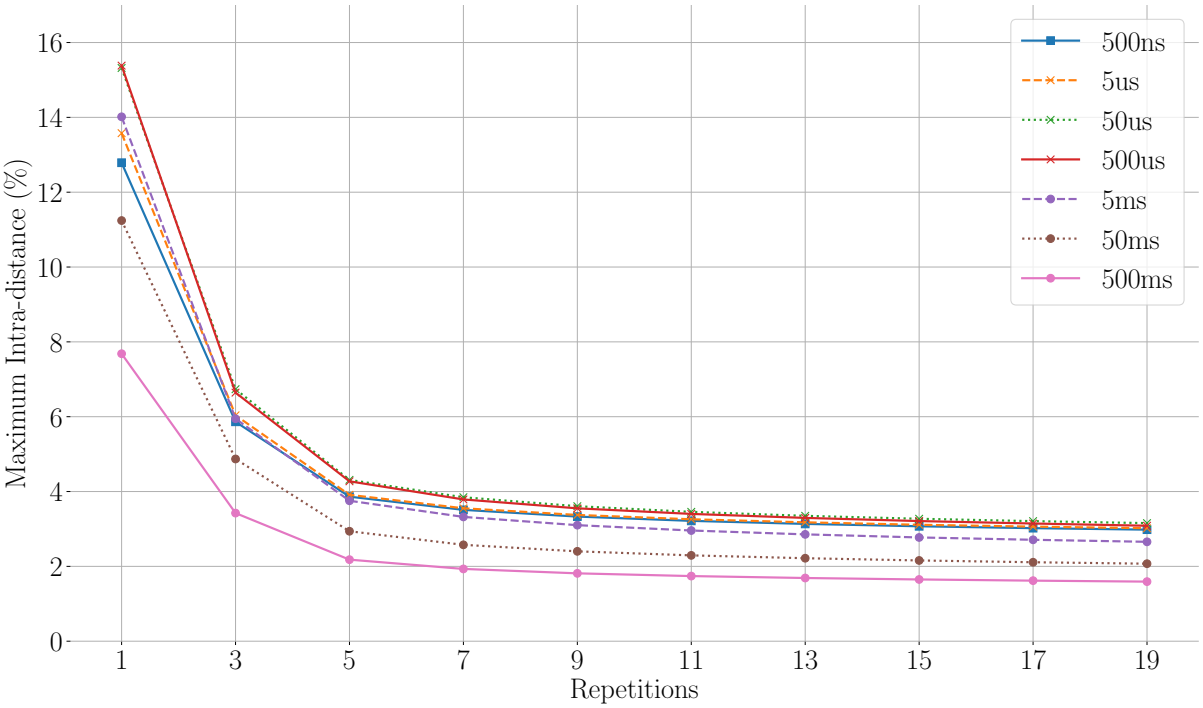


Figure 9.3: Maximum intra-distance at 25°C

table, it is also possible to see that the results of Eq. (9.1) for the repetition code largely match the experimental results discussed above.

$$P_{fail} = 1 - \sum_{i=0}^{t} \binom{n}{i} P_e^i (1 - P_e)^{n-i} \tag{9.1}$$

For the generation of a 256-bit PUF response used in the CRP Ratchet, a total of $n_{bch} - k_{bch} = 116$ bits of helper data are produced. A reduction in the total storage required for helper data is possible due to the response generation method. Since segments are individually corrected, certain segments are included in multiple CRPs. Therefore, the same helper data can be used multiple times, decreasing the error correction overhead.

9.3.2 Hash Function

Having been standardised by NIST, SHA-256 has the advantage of inherent compatibility with other NIST standards, including the HMAC DRBG construction used in our RNG. In addition, due its popularity, SHA-256 implementations have been extensively studied and optimised. In fact, when the parameters of SPONGENT-88 are selected to match the security level of the commonly used SHA-256, SPONGENT-88 has only about 1% the throughput of SHA-256. To achieve a comparable throughput SPONGENT-88 requires a configuration which eventually performs worse than SHA-256 both in silicon area and power consumption[192].

The output length of the chosen hash function has a direct effect on the communication overhead of our protocols, since the majority of messages are in some way derived from hash digests. As a result, the use of SPONGENT-88 in the CRP Ratchet resulted in

a 24% shorter mean message size, as seen in Table 9.2. However, collision, pre-image, and second pre-image resistance are proportional to the number of output bits, making algorithms with shorter outputs less secure. For example, a birthday attack requires 2^{44} evaluations for SPONGENT-88 and 2^{128} evaluations for SHA-256.

	Size in bits	
	SHA-256	**SPONGENT-88**
Both ratchets		
Ratchet Authorisation #1	1312	1312
Ratchet Authorisation #2	1312	1312
CRP Ratchet		
Initialisation #1	768	600
Initialisation #2	1024	688
Initialisation #3	768	600
Step #1	768	264
Step #2	1024	352
Step #3	512	176
ZK CRP Ratchet		
Initialisation #1	768	600
Initialisation #2	1280	1112
Initialisation #3	1024	1024
Step #1	576	240
Step #2	840	504
Step #3	328	328
Step #4	328	328
Step #5	1096	928
Step #6	1032	864
Mean	868	660.8

Table 9.2: Message size in the ratchet interactions

9.3.3 Random Number Generation

Evaluating the quality of random sequences is not an easy task. Different definitions of entropy exist, and the derived entropy metrics are not guaranteed to encapsulate the full behaviour of the examined sequence. This is due to the inherent complexity of randomness, which leads to evaluation methods that merely aim to detect if a given sequence is not random. Matters are complicated even further by the sensitivity of randomness testing methods which leads to radically different results for minor configuration changes.

The operation of our RNG implementation was validated by testing samples of both the input seeds and the RNG outputs. For our evaluation, we used SRAM experimental data and focused on the operating corner with the worst min-entropy, as it was derived in Section 8.5.3, namely at 100°C with $t_{ramp} = 500ms$. By evaluating the RNG in the corner with the worst randomness behaviour, we can be certain that the output quality will be at least equivalent in the rest of the corners.

Firstly, we estimated the entropy of a given set of random sequences with mean μ and standard deviation σ, using Eq. (9.2)[201].

$$entropy = \frac{\mu \times (1 - \mu)}{\sigma^2} \tag{9.2}$$

Each 512Kb SRAM IC is capable of producing $\lfloor \frac{512Kb \times 8}{8534} \rfloor = 491$ full-entropy seeds for each of the 100 measurements acquired during the experiment. After entropy accumulation, seeds have a length of 256 bits, thus resulting in 49100 256-bit sequences per chip. The entropy of these sequences derived with Eq. (9.2) is shown in Table 9.3. In the table we can see that the entropy between ICs presents only minor differences, by virtue of the conservative design of the entropy accumulation process. We can thus conclude that similar entropy any SRAM IC is capable of generating similar entropy.

The seed quality was also verified using the NIST randomness test suite[49] which is considered the authoritative method for statistical randomness testing. Seed data from the 20 ICs was concatenated and divided in sequences of different sizes, in a compromise between stream size and stream quantity. For a detailed description of the individual tests we refer to the original NIST publication. The tests examine each stream and produce:

- Ten 'bin' values C1 through C10, representing the uniformity of the p-value observations. A random stream is expected to exhibit high uniformity.
- An overall P-value, also signifying the aforementioned uniformity. A random stream is expected to have P-value ≥ 0.0001.
- A proportion of successful tests over the total number of tests. The exact proportion expected from a random stream varies depending on the sample size.

Similarly, the output streams of the RNG present a high entropy content. We generated 1000 random streams of 40000 bits in each operating corner and derived their entropy with Eq. (9.2), summarising the results in Table 9.4. The influence of the reseeding interval on the output entropy was minimal, verifying the high quality of the HMAC DRBG construction. The entropy metric does not encapsulate potential compromise of the DRBG internal state. However, replenishing the entropy of the RNG after adversary access has been removed, ensures the continued security of the generator.

IC	Entropy bits	Entropy density
1	256.00	1.0000
2	255.97	0.9999
3	255.99	1.0000
4	255.89	0.9996
5	255.91	0.9996
6	256.02	1.0001
7	256.00	1.0000
8	256.12	1.0005
9	256.00	1.0000
10	255.96	0.9998
11	256.06	1.0002
12	256.06	1.0002
13	256.06	1.0002
14	255.98	0.9999
15	256.02	1.0001
16	256.00	1.0000
17	256.03	1.0001
18	256.00	1.0000
19	256.02	1.0001
20	256.02	1.0001
Mean	256.01	1.0000

Table 9.3: Seed entropy for 100°C, $t_{ramp} = 500ms$. Entropy bit values > 256 are due to the estimation being made on a population sample.

Reseed interval (bits)	Reseeds per stream	Entropy bits	Entropy density
2^8	156	256.00	1.0000
2^9	78	256.01	1.0000
2^{10}	39	256.00	1.0000
2^{12}	4	255.99	0.9999
2^{66}	0	255.98	0.9999

Table 9.4: RNG output entropy for 100°C, $t_{ramp} = 500ms$. Entropy bit values > 256 are due to the estimation being made on a population sample.

Test	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-Value	Proportion	Decision
Stream length: 25600 bits, stream count: 9820, expected proportion: 0.986965													
Block Frequency	974	1006	973	1006	937	955	996	963	1014	996	0.746423	0.990326	PASS
Frequency	1028	939	1019	975	1053	976	938	951	941	1000	0.074708	0.991446	PASS
Longest Run	934	966	1031	977	1023	929	1039	988	987	946	0.110730	0.989206	PASS
Runs	971	976	1009	956	1004	945	970	990	992	1007	0.881079	0.991344	PASS
Stream length: 1001561 bits, stream count: 251, expected proportion: 0.964072 (0.957983 for the random excursion variant)													
Approximate Entropy	31	45	17	21	22	28	24	23	26	14	0.001730	0.988048	PASS
Block Frequency	27	25	21	32	15	22	33	31	20	25	0.206629	0.988048	PASS
Frequency	23	33	29	18	25	22	25	27	23	26	0.735908	0.996016	PASS
Linear Complexity	23	21	18	31	32	21	31	17	27	30	0.215574	0.988048	PASS
Longest Run	28	37	21	23	20	26	28	27	21	20	0.340858	0.980080	PASS
Random Excursions†	20	13	14	10	27	14	12	14	11	12	0.047609	0.986395	PASS
Random Excursions Variant†	19	11	19	12	13	13	13	18	14	15	0.747045	0.986395	PASS
Runs	28	25	21	23	29	30	19	26	25	25	0.892036	0.984064	PASS
Serial†	29	21	32	22	27	17	28	23	22	30	0.514124	0.980080	PASS
Universal	31	30	25	14	25	27	28	18	26	27	0.353733	0.988048	PASS
Stream length: 1505341 bits, stream count: 167, expected proportion: 0.968127 (0.959184 for the random excursion variant)													
Approximate Entropy	24	31	24	16	16	8	11	9	16	12	0.000225	0.982036	PASS
Block Frequency	15	27	16	15	13	18	16	18	11	18	0.306892	0.994012	PASS
Frequency	16	12	23	13	14	15	15	18	24	17	0.408942	0.994012	PASS
Linear Complexity	19	13	15	16	24	14	19	11	16	20	0.478598	0.994012	PASS
Longest Run	19	26	24	14	14	16	9	15	15	15	0.103676	1.000000	PASS
Random Excursions†	10	9	15	12	8	13	6	18	21	7	0.015305	0.983193	PASS
Random Excursions Variant†	12	12	11	18	14	10	12	10	11	9	0.730786	0.991597	PASS
Runs	18	11	15	18	17	21	14	21	19	13	0.669618	0.988024	PASS
Serial†	20	17	16	17	15	17	13	18	11	23	0.669618	0.982036	PASS
Universal	17	20	16	11	18	16	19	14	15	21	0.806261	0.988024	PASS

†This test produced a large number of results. A small representative sample has been included in the table.

Table 9.5: NIST test suite result for seeds at 100°C, $t_{ramp} = 500ms$.

9.3.4 Implementation Cost

It is evident from our discussion of the system architecture that all the components used in the cryptocore, including the PUF, can be constructed from existing primitives that are highly mature and have been used in practice for a long time. As a result, the cost of fabricating the devices discussed in this chapter is relatively low.

While all devices include the same cryptocore, authority devices bear a higher cost due to their interaction with a large number of nodes. This cost is mainly reflected in non-volatile storage space, which has an associated silicon area cost. However, authority devices are also used in limited numbers and are not destined for consumer use. Thus, the low impact that our methods have on nodes is of considerably higher importance than the overhead added by the use of authority devices.

A different cost consideration is that of the total number of CRPs required by the PUF. Different PUF classes can provide widely varying number of CRPs, and the relationship between this number and the PUF silicon area also varies. For memory-based PUFs including SRAMs, the silicon area scales almost linearly with the number of available CRPs. Nevertheless, an estimation of the required number of CRPs over the lifetime of a device is not trivial. For example, in the case of CRP Ratchets, a relatively large number of CRPs is used, with a fairly high speed. However, due to the nature of the protocols, CRPs can be reused.

Application Scenario

We considered a number of example scenarios to verify the features of the implementation. All scenarios were executed on a network of virtual machines, representing each one of the devices.

Simple: The basic protocols of setup, verification, enrolment, and decommission are performed between a single node and a single authority device.

Multiple nodes: Takes place in two phases, with two nodes and a single AD. In the first phase, the AD performs the setup, verification, and enrolment operations with both nodes, the equivalent of adding the nodes to a neighbourhood. Subsequently, the nodes take part in a key exchange and take turns authenticating each other.

Multiple authority devices: With two ADs and a single node, this scenario demonstrates the interactions involved in managing multiple ownership. The first authority device initially executes the setup and the enrolment with the node and afterwards the second AD proceeds to with the enrolment of the node, after it has received authorisation from the first AD. After the completion of this process, the node reaches a state where it simultaneously belongs to two neighbourhoods.

For the rest of this section we use the following use case as a basis for our discussion: a utility company installs 5000 smart meters and enrols them with a single authority device. The topology of the system is such that meters are organised in neighbourhoods

of approximately 50 meters, with each meter having 10 immediate neighbours. Upon installation, the engineer initialises a CRP ratchet between the meter and its immediate neighbours, resulting in 10 ratchets per meter. The purpose of the application is to detect breaches of the smart meters, taking advantage of the redundancy provided by the multiplicity of ratchets.

Network Traffic

All protocol exchanges were captured and analysed with TShark[202] and Wireshark[203], with the help of a custom plug-in developed for the purposes of our work. The high level functionality of the protocols was confirmed by the observed traffic. Nevertheless, examining the traffic while developing our proof-of-concept was invaluable for the detection of implementation issues and shortcomings in the protocol specifications.

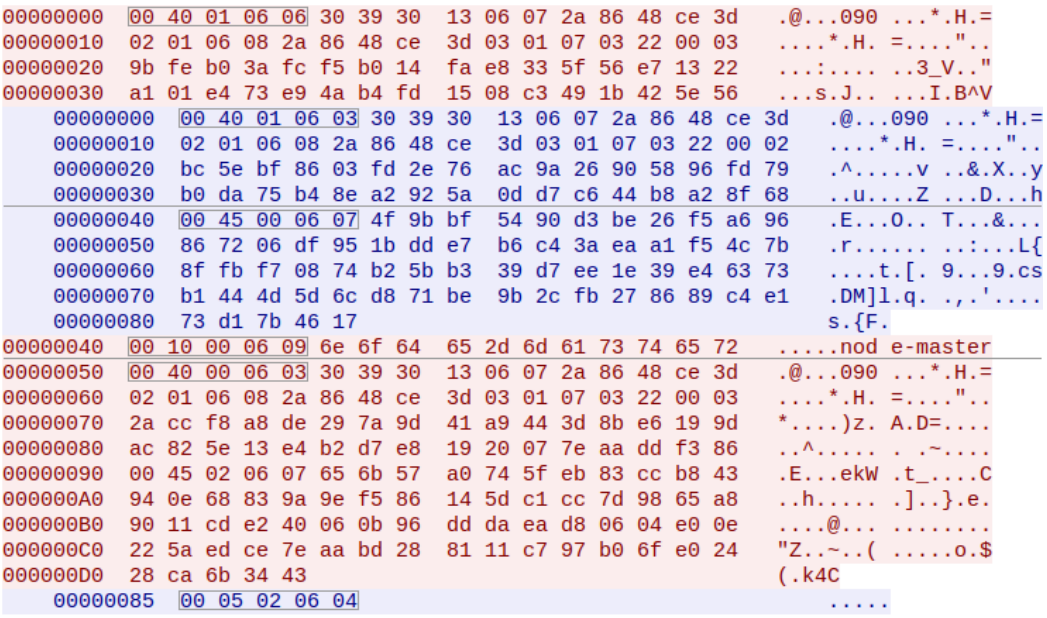
Fig. 9.4 shows an example of the ADS key exchange protocol. In this exchange the nodes have been given the identifiers 'node-master' (in red) and 'node-slave' (in blue) to signify the entity which initiates the protocol. In the last three table columns, the values are shown in hexadecimal and their representation is given in parentheses. The message headers, marked with a rectangle comprise: the total length, flags, phase, command, and payload. As per Table C.1, the value of the Phase header is 0x06 for all Key Exchange messages.

Table 9.2 offers a summary of the communication overhead for both ratchet protocols. Due to their infrequent use, the protocols used by the ADS for neighbourhood management (enrolment, decommission, key exchange, and authentication) do not have a significant effect on the network traffic, especially when considered in the context of a network system that continuously exchanges application data.

Device Storage

An overview of the storage requirements for the two main protocol schemes of Chapters 5 and 6 is given in Table 9.6. As expected, the required storage for the authority devices scales linearly ($O(n)$) with the number of nodes and similarly, peer information stored by nodes is also proportional to the number of peers. There is also a constant cost for each CRP produced by the PUF, since the associated helper data is stored for future use. However, in typical ADS scenarios each node should only be required to produce only one CRP per AD, in addition to the key CRP. The situation is similar for the ratchets, with required node storage scaling linearly with the number of peers. The advantage of these protocols is that, due to their temporal nature, helper data can be safely discarded periodically, thus reducing the long term storage overhead.

Using the example use case introduced above we can calculate the total storage needed on each device, shown in Tables 9.7 and 9.8. The feasibility of such an application from the perspective of storage is clear, since only 650Kb are required on the AD and a mere 4.6Kb are needed on the meters. Therefore, even in applications with higher scaling requirements (e.g. higher number of ratchet pairs) the storage overhead would be minimal, compared



Packet No.	Source	Destination	Length	Flags	Command
1	Master	Slave	0040 (64)	01 (INIT)	06 (Initiate)
2	Slave	Master	0040 (64)	01 (INIT)	03 (Public Key)
3	Slave	Master	0045 (69)	00	07 (Signature)
4	Master	Slave	0010 (16)	00	09 (ID)
5	Master	Slave	0040 (64)	00	03 (Public Key)
6	Slave	Master	0005 (5)	02 (FIN)	04 (Acknowledgement)

Figure 9.4: TCP packet flow for the Key Exchange protocol

to the capabilities of contemporary devices.

Power Consumption

In this section we provide a rough estimate of the energy costs associated with the proposed solutions. Evidently, deriving the power consumption of even the most fundamental operations is not an easy task. A great number of variables affect the actual consumption ranging from transistor-level choices, to implementation details, and high-level algorithm optimisations. Thus, the metrics presented are intended as a guide, and would normally reflect the worst-case performance, for the reasons discussed below.

Table 9.9 summarises the total power consumption for a single round of the protocols of Chapters 5 and 6. We obtained these results using power consumption estimates for basic operations found in literature (see Table 9.10), multiplied by number of times each operation is used. A detailed count of each operation for the various protocols is given in Tables C.7 and C.8.

As seen in Table 9.10, a number of simplifications are made in estimating the power consumption. Firstly, operations like exclusive OR, concatenation, and random number generation are assumed to have negligible contribution (in the order of fW) to the total

Device Type	Object	Size in bits
All	Helper data per PUF response	116
	Various counters	8
AD	Public key per enrolled node	472
	CRP per node that has been set up	512
Node	Public key per authorised/enrolling AD	472
	Signature for own public key per enrolling AD	512
	Public key per peer	472
	ID per peer	8
	Ratchet challenge per peer	256
	Ratchet key per peer	256
	Ratchet ZK commitment per peer	512

Table 9.6: Storage requirements of different objects

Object	Size	Count	Total size (bits)	Total size (Kb)
Key generation helper data	8584	1	8584	1.05
Node public key	472	5000	2360000	288.09
Node CRP	512	5000	2560000	312.50
Node ID	8	5000	40000	4.88
Ratchet counter per node pair	8	45000	360000	43.95
Total			5328584	650.46

Table 9.7: Example of AD storage requirements in smart metering application

consumption[208]. The RNG operation only bears a significant cost when reseeding is needed, an event which is very infrequent, as discussed in Chapter 7. Similarly, the energy cost of each access to the SRAM PUF cells is typically measured in pJ, and thus also assumed negligible. Finally, elliptic curve cryptography operations are approximated with scalar multiplications, as is common practice in literature [205], due to the relatively high cost of this multiplication.

In addition, the energy cost of the radio transmission between nodes cannot be accurately estimated without considering the conditions of the individual application. Factors such as distance, environmental noise, transmission power, and even the underlying communication protocols have a vast effect on the power consumption. However, in typical smart metering applications customer data is transmitted as often as once every five minutes, a frequency which is largely adequate for the proposed protocols, allowing the addition of protocol data to the existing traffic.

Nevertheless, the above results are likely to be overestimating the actual power con-

Object	Size	Count	Total size (bits)	Total size (Kb)
Key generation helper data	8584	1	8584	1.05
AD public key	472	1	472	0.06
Peer public key	472	50	23600	2.88
Peer ID	8	50	400	0.05
Peer ratchet challenge	256	9	2304	0.28
Peer ratchet key	256	9	2304	0.28
Ratchet failure counter	8	9	72	0.01
Ratchet authentication counter	8	9	72	0.01
Total			37808	4.62

Table 9.8: Example of node storage requirements in smart metering application

sumption, since they are derived from several separate works which have somewhat differing aims. In reality, a hardware architecture such as the one described in Chapter 7 would be treated as a whole, enabling optimisations that are not possible for individual hardware blocks.

The data presented in Table 9.9 verifies our intuition regarding the power consumption overhead of various cryptographic primitives. Protocols involving only the use of PUFs and lightweight cryptography such as hash functions (for example the ADS Setup and the CRP Ratchet Step protocols) have very limited energy requirements. One particularly demonstrative example can be seen in the comparison of the Step protocols for the two CRP Ratchet variants. While both protocols serve a similar purpose, the energy overhead of the ZK variant is an order of magnitude higher than that of the 'plain' CRP Ratchet.

Considering the typical load for smart metering applications even in the smallest households, we can draw some conclusions about the impact of our solutions in smart meters. A typical refrigerator consumes around 200 per hour[209]. For the equivalent power, a CRP Ratchet Step can be executed every 15ms. Such a high execution frequency is highly unlikely to be useful in practice, where one execution every few minutes would be adequate to achieve the security goals of the system.

9.4 Conclusion

In this chapter we discussed our software proof-of-concept implementation for the protocols and architectures presented in previous chapters. Through the description of the implementation architecture and the subsequent discussion, we were able to ascertain the practical feasibility of our ideas, especially showing that many of the primitives already exist in current consumer devices.

The value of this implementation was threefold. Firstly, it allowed the detection of errors in protocol specifications, followed by the necessary corrections. Secondly, it provided

Protocol	Power Consumption (mW)			
	AD 1	**AD 2**	**Node 1**	**Node 2**
ADS Setup	21.7	-	21.7	-
ADS Verification	416.1	-	416.1	-
ADS Enrolment	810.5	-	810.5	-
ADS Enrolment (multiple ownership)	394.4	1599.3	1599.3	-
ADS Decommission	394.4	-	394.4	-
ADS Key Exchange	-	-	394.4	394.4
ADS Mutual Authentication	-	-	788.8	788.8
CRP Ratchet Authorisation	878.6	-	439.3	439.3
CRP Ratchet Initialisation	-	-	2083.5	2083.5
CRP Ratchet Step	-	-	43.4	43.4
ZK CRP Ratchet Initialisation	-	-	2038.6	2038.6
ZK CRP Ratchet Step	-	-	761.8	761.8

Table 9.9: Power consumption of a single protocol run (with $\lambda = 5$)

us with a view into the realities of integrating multiple cryptographic and networking components into a single system. And finally, it provided a useful platform for discussion of the different intricacies and implementation choices that are often not immediately clearly while discussing protocols in theory.

As with the cryptocore, the natural next step for the work of this chapter would be the creation of hardware prototype systems and their evaluation in varying conditions.

Operation	Power Consumption (µW)	Reference	Comments
HASH SPONGENT 88	1.57	[117]	-
HASH SPONGENT 256	6.62	[117]	-
HASH SHA-256	11.2	[204]	-
Scalar multiplication	44900	[205]	-
SIG (ECDSA)	394400	[206]	-
VER (ECDSA)	394400	[206]	-
ENC	44900	-	One scalar multiplication.
DEC	44900	-	One scalar multiplication.
ZKC	44900	-	One scalar multiplication.
ZKP	44900	-	One scalar multiplication.
ZKV	89800	-	Two scalar multiplications.
PUF	21680	[207]	Error correction cost. SRAM access cost assumed negligible.
HMG	22.4	-	Two XOR, two HASH, two CONCAT
HMV	22.4	-	Two XOR, two HASH, two CONCAT
RNG	0	-	Assumed negligible, besides reseeding.
XOR	0	[208]	Assumed negligible.
CONCAT	0	-	Assumed negligible.
TX	0	-	Assumed negligible in comparison with existing traffic
RX	0	-	Assumed negligible in comparison with existing traffic

Table 9.10: Power consumption of basic operations

Part IV

Conclusions

10. Conclusions

10.1 Conclusion

The Internet of Things (IoT) has created a new world of connected devices which are constantly monitoring multiple aspects of our society, collecting and transmitting sensitive data. At the same time, these devices are placed in unmonitored environments, or come under the control of individuals who lack the expertise or the interest to manage them securely. Unfortunately, the easiest way to manage the security of such scenarios is with increased centralisation which places a great amount of trust in the central authority.

In the present thesis we have studied the primitive of *unclonability* with the aim of integrating it in IoT scenarios, to establish ownership relationships and ensure the secure operation of the networked devices. Throughout our work, unclonability is supported by Physical Unclonable Functions (PUFs), which server as a building block for a wide variety of protocols and methods.

In Part I, we introduced the notion of unclonability in the context of networks, and described an *unclonability stack* which facilitates the propagation of unclonability from the physical domain of PUFs, to the logical domain of network topologies. This is achieved through the incremental creation of trust relationships between components. Devices source their unclonable secrets from the corresponding PUF which is integrated in their hardware. Subsequently, these secrets are used as the basis of pairwise relationships that are eventually transformed into group relationships. The multiple relationship levels allow the detection of topology distortions in the network, including events such as node removal, unknown node introduction, change of node location etc.

Part II follows the structure of the unclonability stack. In Chapter 5, we described a collection of cryptographic protocols designed to enable the creation of *neighbourhoods* of network nodes. The protocols are based on a combination of public key cryptography and PUFs, and provide key generation, key exchange, mutual authentication as well as group membership management. The most noteworthy achievement of this chapter was the introduction of *authority devices (ADs)*, which are unclonable tokens that are used for the initialisation and management of the neighbourhoods. Only a single AD can have control over a neighbourhood and thus, due its unclonability properties, its holder can be certain that no other entity has the same access. Therefore, authority devices provide technical solutions that support strong ownership relationships in the digital domain.

In Chapter 6, we combined the unclonable secrets provided by the PUFs, with the primitive of key refreshment or 'ratcheting'. We proposed and evaluated two variants of a PUF-based *continuous authentication protocol*. The first variant uses simple hash

and XOR operations to encrypt and transmit PUF responses, with minimal computation overhead. On the other hand, the second variant provides higher security guarantees by using zero knowledge proofs to provide the same features. Both variants are designed to run periodically, generating a temporal authentication sequence.

Thirdly, in Part III we turned our attention to issues affecting the implementation of our methods in practice. In Chapter 7, we presented the blueprint for a cryptographic core (cryptocore) with a dual purpose: to efficiently implement the necessary cryptographic operations, while encompassing the PUF with a secure interface. A separate cryptocore component creates a physical boundary between the PUF and the rest of the system, and gathers the sensitive logic circuits in a well-defined area which can then be physically protected as needed.

A particularly prominent PUF construction, the SRAM PUF is examined in Chapter 8. Based on experimental data recorded in two ambient temperatures and seven supply voltage ramp-up times, we evaluated the behaviour of SRAM ICs as PUFs. We looked at the inter-distance and intra-distance metrics which confirmed that SRAMs are capable of providing responses which are both highly unique and considerably stable. At the same time, we were able to produce full entropy random seeds from approximately 2.2Kb of SRAM data. This result clearly demonstrates the suitability of SRAMs for entropy generation, even under adverse environmental conditions.

Finally, the work of Chapters 7 and 8 was combined into a software proof-of-concept (PoC), which allowed us to gain insight into the detailed operation of our methods in practice. We evaluated a number of cryptographic primitives, and developed the PoC based on state-of-the-art building blocks. All the cryptographic operations were performed with elliptic curve cryptography variants, enabling their use in resource constrained applications, which are common in the IoT. Additionally, the PUF component was emulated with experimental data acquired from hardware SRAM ICs, thus achieving a highly accurate representation of a practical application.

10.2 Future Work

Throughout this thesis, we chose to focus on the whole unclonability stack rather than attempt to fully investigate every possible aspect of the proposed methods and protocols. We strongly believe in the potential of unclonability as a primitive for novel security applications. Integrating unclonability in consumer devices would have a profound impact on the societal concepts of ownership, authority, and eventually trust. The main driver behind adoption of unclonability in practice are PUFs. A number of interesting research paths arise from our work, that would allow the further development of the PUF field, and aid the integration of unclonability in practical applications.

Firstly, we concentrated our efforts in setting the foundation for the unclonability stack and its individual layers. Thus, formal verification of the proposed protocols would be highly beneficial for the establishment of this foundational understanding. Especially

in the case of the continuous authentication protocols, to the best of our knowledge the combination of PUFs and ratchets, and the resulting security properties have not been studied before.

Additionally, we can discern a clear potential for integration of the proposed methods in existing IoT platforms. Our solutions focus on supporting *unclonable topologies*, rather than *unclonable data*. Nevertheless, in the authentication protocols of Chapter 6 the mechanisms for establishing shared secrets are already present and can be extended to provide additional features including data encryption and signature. One can envision a form of Application Programming Interface (API) allowing the use of unclonable authentication and other unclonability principles from high level applications including operating systems.

Finally, we took great care to ensure the practicality of our solutions, as seen through this thesis, and in Chapter 9 in particular. However, as discussed in the same chapter, several practical details and intricacies are often obscured during high level protocol analysis. In the context of our work, this was often a desirable effect, allowing us to focus on the big picture. Nonetheless, we are certain that a reference implementation in hardware would generate supplementary insights.

We expect that, as PUFs and their unclonability offerings increasingly become the focus of research endeavours from both the industry and the academia, the rate of adoption in consumer and commercial applications in general will rise sharply. Our hope is that researchers and practitioners will continue focusing on decentralised solutions in addition to the highly centralised commercial offerings of our time. In the following sections, a few interesting additions to our work are sketched, with an eye towards further research.

10.2.1 Neighbourhood Chains

To begin with, we outline a protocol for detecting and tracing network topology distortions as they were defined in Section 2.4.5. The protocol relies on two pillars: the pairwise authentication protocols of Chapter 6, and the inherent redundant topology of common networks.

For our discussion we use the same system of nodes as in the previous chapters. The nodes are organised in a neighbourhood and have previously established a ratchet authentication protocol with each of their immediate neighbours. The type of authentication protocol does not affect the methods discussed below, provided that the necessary security guarantees are ensured. Different protocols can be used depending on the threat model of each node, for example the 'plain' CRP ratchets can be mixed with the ZK CRP ratchets in the same neighbourhood.

In summary, the protocol operates in rounds. During each round, one or more nodes authenticate their immediate neighbours via the appropriate authentication protocols. The results of the round are broadcast to the whole neighbourhood which collectively decides on a status for each node. Subsequently, the status information is added to an

attestation chain which we call *neighbourhood chain*, allowing for the establishment of temporal continuity of node status. The resulting chain can be used as an *attestation log* that can be examined when required by members of the neighbourhood or outsiders, proving the status of the neighbourhood as a whole. Therefore, the proposed protocol does not include provisions for distortion prevention or recovery and appropriate actions should be taken at the system level.

In a manner similar to blockchain applications, the joined consensus of all the nodes ensures that the influence of a minority of compromised nodes will not have an adverse effect on the security of the system. Additionally, the high level of redundant connections exhibited in many IoT scenarios, greatly increases the probability of distortion detection, especially in densely connected networks, as shown in Fig. 10.1.

One could imagine this redundancy and the protocol operation discussed below, in the context of a 'smart' home. For every *connected device* added to the network, the number of relationships in the neighbourhood grows exponentially, since the majority of devices will be in range of each other or can be bridged through a gateway.

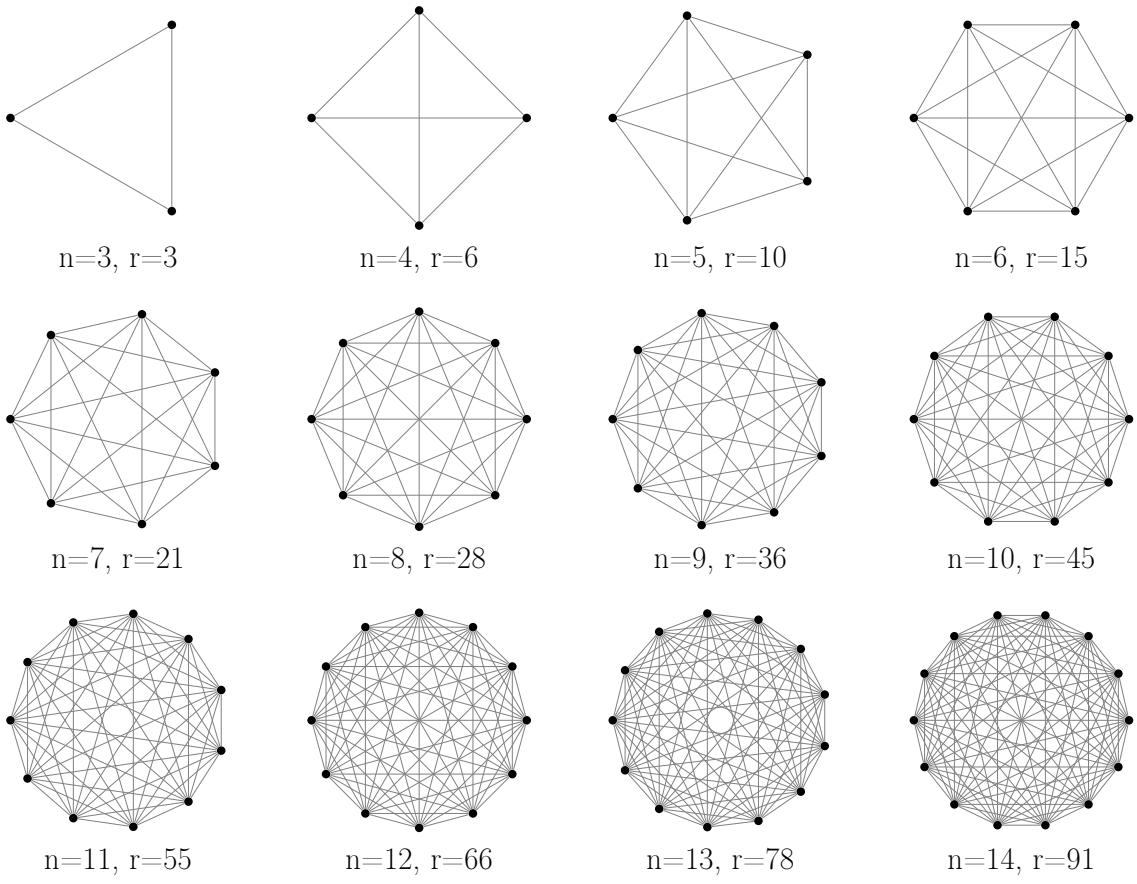


Figure 10.1: Number of relationships in mesh topologies (n:nodes, r:relationships)

There exist many aspects of the discussed protocols that can vary by application and security requirements. Thus, in our discussion we consider an example embodiment of the protocols, with a view to demonstrating the possibilities. In particular, we assume that potential adversaries are not capable of controlling more than 49% of the nodes, and we focus on neighbourhoods with a relatively small number of members.

Each round of the chain protocol comprises an *inspection phase* and a *consensus phase*, described below. The rounds are periodically executed with a frequency defined by the application requirements. As expected, a frequency trade-off is required between increased security and reduced overhead. Each of the nodes stores a status table regarding its immediate neighbours. This table is updated periodically with information stemming from pairwise authentication as well as notifications from neighbours.

Inspection Phase

During the inspection phase, one or more nodes are responsible for the verification of their immediate neighbours and the broadcasting of the results. The transfer of responsibility between nodes can be considered in terms of a virtual token passing protocol, with one or multiple tokens. The verification itself is based on the pre-established ratchets between the nodes, effectively leading to a new ratchet step every time a node pair is involved in the inspection phase.

One or more nodes can be responsible for carrying out the inspection phase in every round and the number of nodes can be predefined or it can vary between rounds, based on the selection algorithm. However, for the remainder of this section, we assume that a single node is selected per round. We discern the following selection algorithm classes:

Static: Based on static analysis, the order of nodes can be preselected and embedded in the protocol, through examination of the network graph. For example, nodes with a higher number of immediate neighbours (graph vertices with a higher degree) are able to authenticate a larger portion of the network.

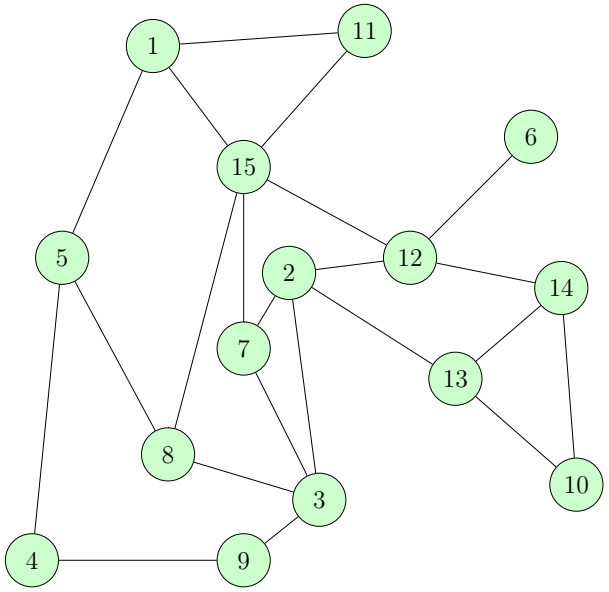
Dynamic: Avoiding the need for extensive analysis of the neighbourhood topology, dynamic selection can be based on multiple criteria, both regarding security and efficiency. For example, nodes with a lower battery level can be selected less often, to avoid further depletion, or neighbours of nodes that are deemed questionable can be selected with a higher frequency.

Randomised: A hybrid between the above, a randomised selection algorithm is independent of the neighbourhood topology. *Gossip* or *epidemic* broadcast protocols are a typical example of this category[210]. Randomisation ensures an unpredictable pattern that cannot be exploited by adversaries, and reduces the chances of error due to node failure.

Evidently, the choice of selection method is heavily dependent on the application and the nature of the network, with static algorithms involving higher setup costs but also providing increased robustness. Dynamic algorithms on the other hand, are more suited to actively changing topologies and might be simpler to implement. In either case, two metrics are of interest: (a) the mean time between consecutive authentications of a node, and (b) the mean number of authenticated nodes per round. The options can be evaluated

in a simulation environment and the aforementioned metrics can be easily derived from the resulting observations.

At the start of the protocol, every node establishes a table of its immediate neighbours and attaches an authentication weight $w_a = 1$ to all of them. The table also contains a timestamp for the last successful authentication attempt. During an inspection phase, the selected node performs a ratchet step with m randomly selected neighbours. The random selection is guided by the weights w_a: the weight of a node is decreased after a successful authentication, and increased upon failure, thus performing more frequent checks on nodes with questionable activity. Once all the ratchet steps are finalised, the initiating node populates its status table with the latest information for each of the m neighbours. Upon completion of the above process, the inspection phase is finished, and the next inspection takes place after t_i time units, in a infinite loop. Fig. 10.2 provides a snapshot of the status tables for a neighbourhood topology example.



(a) Example neighbourhood topology

Node	Weight	Last Success Seq. Number	Last Failure Seq. Number	Verified
1	0.1	6	-1†	True
4	0.1	6	-1†	True
8	1.5	2	3	True
15	1.0	4	5	False

†Signifies no known failures.

(b) Status table snapshot for Node 5

Figure 10.2: Neighbourhood topology and status table snapshot

Consensus Phase

The nodes consolidate their status tables by performing a consensus protocol with a period of t_g time units. The protocol uses a 'gossip' method, to avoid flooding the neighbourhood

with broadcast packets. We outline the process below:

1. The last node that was responsible for performing the Inspection Phase also initiates the Consensus Phase by transmitting a status packet to one of its neighbours, selected at random. As seen in Fig. 10.3, the packet contains the status table of the source node (omitting the weights), a sequence number, and a signature. If a public key signature algorithm is undesirable, the packet can be forwarded in a hop-by-hop manner, with a new signature generated for every transmission, based on the ratchet key of the corresponding node pair.

2. Upon receipt of the status, the destination node:

   (a) Verifies the packet signature.

   (b) Updates its own status table with the received information by joining the two tables. For nodes that are found as expected, the timestamps are updated to the latest value. If a new node or an authentication failure is included in the received information, then it is added to the table as 'unverified'.

   (c) The destination node validates the unverified entries of its status table in two ways. For immediate neighbours, a ratchet step is performed and unverified information is accepted or discarded accordingly. For other neighbourhood members, the node waits for corroborating information originating from additional nodes. At the end of the consensus phase, a majority vote is used to decide on information that is still unverified.

   (d) When all decisions are finalised, the node weights are updated accordingly, to reflect the last status of each node.

3. The two nodes taking part in the protocol forward their status tables to two other neighbours, selected at random. The above process is repeated until all n nodes of the neighbourhood have been updated. Due to this forwarding technique, $log_2 n$ gossip rounds are required to update the whole neighbourhood, making it an efficient process even for very large neighbourhoods[210].

4. When a consensus is reached, all nodes add the final status table to their corresponding chain, by appending a data block containing the table and hash of the previous block (see Fig. 10.4). Similarly to the blockchain primitive, this method creates a tamper-evident log of the neighbourhood state.

The first block of the neighbourhood chain, or 'genesis block', has a special status since its validity has to be proven using a different method than the rest of the blocks. In our case, the first genesis block is a random nonce signed by the authority device which controls the neighbourhood, and can thus be verified by every node (and other entities with knowledge of the AD's public key).

Sequence Number		
Signature		
Node	Success SeqNo	Failure SeqNo
.	.	.
.	.	.
.	.	.

Figure 10.3: Gossip Packet Structure

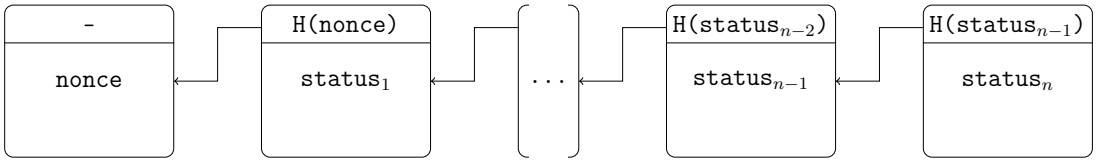


Figure 10.4: Neighbourhood chain structure

A common issue with this approach is that the size of the chain grows monotonically with time, leading to growing storage demands. Storage space is especially scarce in IoT devices, and thus the neighbourhood chains should be kept relatively small. The advantage of such systems compared to cryptocurrency applications is that the number of nodes is usually many orders of magnitude smaller in IoT. Therefore, a possible method is to periodically 'trim' the chain and replace the genesis block with a later block. The latter can be signed by all the nodes through a group signature protocol[211].

10.2.2 Node Context

In the majority of this work, we only considered authentication methods based on secrets stemming from a single source. However, the nodes possess a multiplicity of attributes that describe their state, both individually and within the neighbourhood. As a result, these attributes can be used to construct the *node context* which can be used as a weak identity provider, in aid of the PUF-based secrets. The context of a certain node can comprise:

- Device state such as battery level, physical location, remaining storage space, and hardware architecture.
- Logical location which is the position of the node within the neighbourhood graph. This component is part of the neighbourhood chains.
- The state of the neighbourhood from the point of view of the node. This is the 'status table' kept by nodes in the neighbourhood chains.

Unlike the PUF secrets, the above quantities can be largely predicted or modelled. Thus, an unexpected diversion from the expected values (for example, a sudden decrease in the battery level) can signify external interference or, at best, malfunction.

Similar ideas have been proposed to authenticate nodes and detect clones in different applications[212], [213]. Signorini in [213] used the term 'proof-of-knowledge' to describe the continuity and attestation of node context. In essence, when a malicious node infiltrates the neighbourhood, it can be differentiated from existing, legitimate nodes due to the fact that it has not been present throughout the history of the neighbourhood. As such, the malicious node would not have access to the same amount of information about previous states of the neighbourhood and it would not be able to prove its knowledge.

The context derived from the information discussed above is neither unclonable nor secret. Therefore, the value of context in authentication applications relates to the increased barrier against attackers that do not have the resources to track the state of the neighbourhood over time. Secure methods of proving certain node properties are required, especially regarding the physical domain (i.e. battery level).

10.2.3 System Level Interactions

We can also envision higher level interactions between neighbourhoods. For example, additional layers of chains can be constructed in a tree organisation, with the system level at the root, and the individual nodes as leaves. As the depth of the tree increases, blocks are added with a higher frequency. Based on this structure, a third party would be able to recursively verify the current and past states of the system without relying on the honesty of specific nodes.

Communication and trust between neighbourhoods is also enabled by the neighbourhood chains. Hand-off methods can be used to allow for nodes to operate in or seamlessly traverse multiple neighbourhoods. The destination neighbourhood can verify the incoming node's history through examining the appropriate chain, and can thus make an informed decision about whether to accept the outsider or not.

Part V

Appendices

A. SRAM Data Analysis

A.1 Inter-distance Results

	25°C				**100°C**			
t_{ramp}	**Min.**	**Mean**	**Max.**	**Std.Dev.**	**Min.**	**Mean**	**Max.**	**Std.Dev.**
500ns	34.27 %	37.55 %	39.38 %	0.93 %	41.78 %	43.77 %	44.97 %	0.61 %
5us	34.80 %	38.15 %	41.12 %	0.91 %	43.01 %	44.75 %	46.21 %	0.54 %
50us	40.23 %	42.34 %	44.58 %	0.69 %	47.80 %	48.47 %	49.43 %	0.18 %
500us	45.64 %	46.72 %	47.71 %	0.35 %	49.37 %	49.63 %	49.75 %	0.03 %
5ms	49.11 %	49.50 %	49.72 %	0.10 %	49.14 %	49.61 %	49.73 %	0.03 %
50ms	49.50 %	49.62 %	49.73 %	0.03 %	49.46 %	49.59 %	49.72 %	0.03 %
500ms	49.47 %	49.57 %	49.70 %	0.03 %	49.46 %	49.59 %	49.72 %	0.03 %
	25°C aged				**100°C aged**			
t_{ramp}	**Min.**	**Mean**	**Max.**	**Std.Dev.**	**Min.**	**Mean**	**Max.**	**Std.Dev.**
500ns	29.08 %	36.43 %	38.48 %	1.02 %	41.20 %	43.35 %	44.67 %	0.65 %
5us	34.59 %	37.20 %	39.69 %	0.95 %	42.71 %	44.48 %	46.09 %	0.56 %
50us	39.64 %	41.64 %	43.97 %	0.73 %	47.58 %	48.28 %	48.85 %	0.19 %
500us	44.57 %	46.23 %	48.21 %	0.42 %	49.46 %	49.62 %	49.73 %	0.03 %
5ms	48.94 %	49.44 %	49.92 %	0.13 %	49.51 %	49.62 %	49.73 %	0.03 %
50ms	49.05 %	49.63 %	49.74 %	0.03 %	49.47 %	49.60 %	49.73 %	0.03 %
500ms	49.46 %	49.58 %	49.71 %	0.03 %	49.23 %	49.59 %	49.71 %	0.03 %

Table A.1: Inter-distance by operating corner

	25°C			100°C			25°C aged			100°C aged		
IC	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
0	46.06 %	46.71 %	47.40 %	49.43 %	49.63 %	49.73 %	45.14 %	46.19 %	47.91 %	49.46 %	49.62 %	49.72 %
1	45.83 %	46.49 %	47.22 %	49.40 %	49.62 %	49.72 %	44.83 %	45.95 %	47.80 %	49.49 %	49.61 %	49.69 %
2	46.30 %	46.84 %	47.50 %	49.44 %	49.62 %	49.72 %	45.46 %	46.39 %	48.01 %	49.52 %	49.61 %	49.69 %
3	46.46 %	47.00 %	47.67 %	49.43 %	49.64 %	49.75 %	45.63 %	46.54 %	48.16 %	49.54 %	49.63 %	49.72 %
4	46.02 %	46.65 %	47.39 %	49.42 %	49.62 %	49.72 %	45.07 %	46.12 %	47.88 %	49.49 %	49.61 %	49.73 %
5	46.35 %	46.93 %	47.64 %	49.42 %	49.63 %	49.73 %	45.45 %	46.47 %	48.21 %	49.52 %	49.62 %	49.70 %
6	45.79 %	46.47 %	47.23 %	49.45 %	49.63 %	49.74 %	44.83 %	45.93 %	47.72 %	49.52 %	49.62 %	49.70 %
7	45.75 %	46.42 %	47.19 %	49.37 %	49.63 %	49.74 %	44.74 %	45.86 %	47.70 %	49.52 %	49.62 %	49.70 %
8	45.64 %	46.37 %	47.23 %	49.39 %	49.63 %	49.70 %	44.57 %	45.80 %	47.87 %	49.47 %	49.61 %	49.70 %
9	45.84 %	46.53 %	47.18 %	49.41 %	49.64 %	49.73 %	44.94 %	46.04 %	47.84 %	49.55 %	49.63 %	49.72 %
10	46.30 %	46.89 %	47.55 %	49.46 %	49.63 %	49.72 %	45.49 %	46.43 %	48.09 %	49.53 %	49.62 %	49.71 %
11	46.26 %	46.84 %	47.54 %	49.48 %	49.64 %	49.74 %	45.42 %	46.37 %	48.07 %	49.53 %	49.63 %	49.72 %
12	46.49 %	46.99 %	47.67 %	49.45 %	49.64 %	49.75 %	45.66 %	46.55 %	48.17 %	49.50 %	49.63 %	49.72 %
13	46.05 %	46.63 %	47.34 %	49.45 %	49.62 %	49.73 %	45.15 %	46.13 %	47.91 %	49.52 %	49.61 %	49.68 %
14	45.64 %	46.24 %	47.01 %	49.50 %	49.61 %	49.73 %	44.83 %	45.71 %	47.57 %	49.49 %	49.60 %	49.70 %
15	46.12 %	46.73 %	47.43 %	49.46 %	49.63 %	49.74 %	45.38 %	46.23 %	47.94 %	49.52 %	49.62 %	49.70 %
16	46.10 %	46.74 %	47.43 %	49.44 %	49.63 %	49.74 %	45.41 %	46.26 %	48.00 %	49.54 %	49.62 %	49.71 %
17	46.55 %	47.08 %	47.71 %	49.50 %	49.63 %	49.72 %	45.87 %	46.64 %	48.20 %	49.52 %	49.63 %	49.71 %
18	46.58 %	47.09 %	47.66 %	49.48 %	49.65 %	49.75 %	45.92 %	46.64 %	47.98 %	49.56 %	49.64 %	49.72 %
19	46.16 %	46.76 %	47.34 %	49.54 %	49.63 %	49.74 %	45.54 %	46.28 %	47.55 %	49.51 %	49.62 %	49.69 %
Mean		46.72 %			49.63 %			46.23 %			49.62 %	

Table A.2: Inter-distance for the nominal ramp-up time ($t_{ramp} = 500us$)

	$t_{ramp} = 500ns$			$t_{ramp} = 500us$			$t_{ramp} = 500ms$		
IC	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
0	35.41 %	37.53 %	38.66 %	46.06 %	46.71 %	47.40 %	49.47 %	49.57 %	49.66 %
1	34.78 %	37.01 %	38.15 %	45.83 %	46.49 %	47.22 %	49.47 %	49.57 %	49.65 %
2	36.33 %	38.24 %	39.34 %	46.30 %	46.84 %	47.50 %	49.49 %	49.56 %	49.68 %
3	36.53 %	38.43 %	39.38 %	46.46 %	47.00 %	47.67 %	49.50 %	49.58 %	49.67 %
4	35.24 %	37.39 %	38.50 %	46.02 %	46.65 %	47.39 %	49.47 %	49.55 %	49.66 %
5	35.52 %	37.62 %	38.74 %	46.35 %	46.93 %	47.64 %	49.49 %	49.57 %	49.66 %
6	35.19 %	37.35 %	38.43 %	45.79 %	46.47 %	47.23 %	49.49 %	49.57 %	49.66 %
7	34.66 %	36.89 %	37.99 %	45.75 %	46.42 %	47.19 %	49.48 %	49.57 %	49.67 %
8	34.27 %	36.23 %	37.37 %	45.64 %	46.37 %	47.23 %	49.47 %	49.57 %	49.66 %
9	34.41 %	36.58 %	37.63 %	45.84 %	46.53 %	47.18 %	49.49 %	49.55 %	49.65 %
10	36.38 %	38.25 %	39.25 %	46.30 %	46.89 %	47.55 %	49.49 %	49.58 %	49.67 %
11	36.29 %	38.13 %	39.20 %	46.26 %	46.84 %	47.54 %	49.50 %	49.58 %	49.65 %
12	36.44 %	38.24 %	39.32 %	46.49 %	46.99 %	47.67 %	49.48 %	49.58 %	49.69 %
13	35.46 %	37.45 %	38.51 %	46.05 %	46.63 %	47.34 %	49.50 %	49.57 %	49.68 %
14	34.47 %	36.63 %	37.69 %	45.64 %	46.24 %	47.01 %	49.51 %	49.61 %	49.70 %
15	35.68 %	37.67 %	38.73 %	46.12 %	46.73 %	47.43 %	49.49 %	49.58 %	49.66 %
16	35.15 %	37.20 %	38.31 %	46.10 %	46.74 %	47.43 %	49.48 %	49.56 %	49.64 %
17	36.44 %	38.23 %	39.29 %	46.55 %	47.08 %	47.71 %	49.49 %	49.58 %	49.66 %
18	36.48 %	38.30 %	39.32 %	46.58 %	47.09 %	47.66 %	49.51 %	49.59 %	49.70 %
19	35.70 %	37.63 %	38.68 %	46.16 %	46.76 %	47.34 %	49.48 %	49.57 %	49.66 %
Mean		37.55 %			46.72 %			49.57 %	

Table A.3: Inter-distance at 25°C

A.2 Intra-distance Results

	25°C				100°C			
t_{ramp}	**Min.**	**Mean**	**Max.**	**Std.Dev.**	**Min.**	**Mean**	**Max.**	**Std.Dev.**
500ns	6.38 %	8.28 %	12.79 %	0.59 %	7.06 %	8.60 %	13.21 %	0.61 %
5us	6.46 %	8.41 %	13.58 %	0.62 %	7.07 %	8.60 %	13.76 %	0.63 %
50us	7.10 %	9.24 %	15.32 %	0.73 %	6.78 %	8.41 %	13.26 %	0.64 %
500us	7.12 %	9.30 %	15.38 %	0.79 %	5.70 %	7.07 %	10.53 %	0.53 %
5ms	6.19 %	8.35 %	14.01 %	0.71 %	4.24 %	5.91 %	9.00 %	0.47 %
50ms	4.68 %	6.75 %	11.24 %	0.59 %	3.37 %	4.85 %	8.47 %	0.43 %
500ms	3.68 %	4.83 %	7.68 %	0.34 %	2.50 %	3.60 %	5.87 %	0.31 %
	25°C aged				**100°C aged**			
t_{ramp}	**Min.**	**Mean**	**Max.**	**Std.Dev.**	**Min.**	**Mean**	**Max.**	**Std.Dev.**
500ns	6.18 %	8.18 %	12.18 %	0.59 %	7.15 %	8.71 %	14.43 %	0.63 %
5us	6.54 %	8.43 %	14.24 %	0.62 %	7.13 %	8.70 %	14.52 %	0.64 %
50us	7.20 %	9.43 %	16.34 %	0.76 %	6.92 %	8.53 %	13.64 %	0.64 %
500us	7.07 %	9.51 %	15.05 %	0.82 %	5.97 %	7.43 %	12.64 %	0.59 %
5ms	6.55 %	8.81 %	14.02 %	0.76 %	4.67 %	6.30 %	10.80 %	0.53 %
50ms	4.99 %	7.28 %	11.78 %	0.69 %	3.56 %	5.12 %	8.95 %	0.47 %
500ms	3.97 %	5.24 %	8.64 %	0.40 %	2.54 %	3.72 %	5.58 %	0.32 %

Table A.4: Intra-distance by operating corner

A.3 Unstable Cells

	25°C			100°C			25°C aged			100°C aged		
IC	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
1	7.93 %	9.32 %	12.75 %	6.60 %	7.37 %	9.29 %	7.88 %	9.53 %	12.46 %	6.97 %	7.75 %	10.92 %
2	7.93 %	9.39 %	12.89 %	6.48 %	7.29 %	9.24 %	7.85 %	9.58 %	12.57 %	6.84 %	7.67 %	10.84 %
3	7.41 %	8.71 %	11.83 %	5.75 %	6.47 %	8.20 %	7.40 %	8.90 %	11.57 %	6.03 %	6.77 %	9.51 %
4	7.65 %	9.05 %	12.40 %	5.99 %	6.78 %	8.62 %	7.70 %	9.30 %	12.12 %	6.35 %	7.15 %	10.11 %
5	8.00 %	9.41 %	12.82 %	6.58 %	7.39 %	9.35 %	8.01 %	9.66 %	12.62 %	6.98 %	7.80 %	11.02 %
6	8.58 %	10.24 %	14.06 %	6.55 %	7.44 %	9.50 %	8.66 %	10.54 %	13.85 %	6.98 %	7.86 %	11.18 %
7	8.06 %	9.53 %	13.08 %	6.56 %	7.38 %	9.35 %	8.05 %	9.73 %	12.74 %	6.95 %	7.78 %	11.00 %
8	7.83 %	9.20 %	12.59 %	6.34 %	7.11 %	8.98 %	7.73 %	9.36 %	12.24 %	6.69 %	7.48 %	10.60 %
9	9.34 %	11.12 %	15.38 %	7.38 %	8.31 %	10.53 %	9.48 %	11.45 %	15.05 %	7.95 %	8.89 %	12.64 %
10	8.67 %	10.19 %	13.84 %	6.80 %	7.67 %	9.62 %	8.52 %	10.29 %	13.39 %	7.06 %	7.89 %	11.12 %
11	7.63 %	8.99 %	12.30 %	5.91 %	6.68 %	8.48 %	7.61 %	9.18 %	11.99 %	6.20 %	6.97 %	9.85 %
12	7.80 %	9.22 %	12.63 %	6.15 %	6.95 %	8.85 %	7.80 %	9.43 %	12.34 %	6.47 %	7.28 %	10.31 %
13	7.12 %	8.40 %	11.46 %	5.70 %	6.40 %	8.12 %	7.07 %	8.58 %	11.21 %	5.97 %	6.68 %	9.41 %
14	7.78 %	9.11 %	12.41 %	6.14 %	6.90 %	8.70 %	7.74 %	9.31 %	12.10 %	6.51 %	7.27 %	10.26 %
15	7.56 %	8.84 %	11.98 %	6.11 %	6.88 %	8.69 %	7.46 %	8.99 %	11.69 %	6.41 %	7.19 %	10.19 %
16	7.71 %	9.08 %	12.42 %	6.31 %	7.10 %	8.98 %	7.65 %	9.27 %	12.13 %	6.64 %	7.42 %	10.49 %
17	8.37 %	9.95 %	13.65 %	6.36 %	7.17 %	9.08 %	8.40 %	10.21 %	13.36 %	6.77 %	7.61 %	10.73 %
18	7.29 %	8.62 %	11.79 %	5.79 %	6.55 %	8.33 %	7.25 %	8.83 %	11.55 %	6.09 %	6.85 %	9.74 %
19	7.24 %	8.48 %	11.50 %	5.81 %	6.53 %	8.22 %	7.19 %	8.68 %	11.25 %	6.11 %	6.83 %	9.64 %
20	7.86 %	9.16 %	12.42 %	6.32 %	7.11 %	8.99 %	7.84 %	9.41 %	12.19 %	6.62 %	7.38 %	10.35 %
Mean		9.30 %			7.07 %			9.51 %			7.43 %	

Table A.5: Intra-distance for the nominal ramp-up time ($t_{ramp} = 500us$)

	$t_{ramp} = 500ns$			$t_{ramp} = 500us$			$t_{ramp} = 500ms$		
IC	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
1	4.15 %	4.90 %	6.73 %	6.88 %	8.34 %	11.46 %	7.93 %	9.32 %	12.75 %
2	4.01 %	4.86 %	6.72 %	6.78 %	8.29 %	11.40 %	7.93 %	9.39 %	12.89 %
3	3.85 %	4.54 %	6.16 %	6.64 %	8.01 %	10.99 %	7.41 %	8.71 %	11.83 %
4	3.90 %	4.68 %	6.41 %	6.69 %	8.15 %	11.14 %	7.65 %	9.05 %	12.40 %
5	4.02 %	4.82 %	6.61 %	6.87 %	8.33 %	11.38 %	8.00 %	9.41 %	12.82 %
6	4.11 %	5.04 %	7.02 %	7.34 %	9.03 %	12.42 %	8.58 %	10.24 %	14.06 %
7	4.20 %	4.99 %	6.83 %	6.96 %	8.48 %	11.65 %	8.06 %	9.53 %	13.08 %
8	4.08 %	4.82 %	6.58 %	6.72 %	8.15 %	11.17 %	7.83 %	9.20 %	12.59 %
9	4.66 %	5.60 %	7.68 %	7.54 %	9.29 %	12.79 %	9.34 %	11.12 %	15.38 %
10	4.20 %	5.06 %	6.97 %	7.06 %	8.57 %	11.66 %	8.67 %	10.19 %	13.84 %
11	4.08 %	4.79 %	6.44 %	6.76 %	8.21 %	11.25 %	7.63 %	8.99 %	12.30 %
12	3.94 %	4.73 %	6.47 %	6.85 %	8.36 %	11.44 %	7.80 %	9.22 %	12.63 %
13	3.74 %	4.45 %	6.08 %	6.39 %	7.74 %	10.62 %	7.12 %	8.40 %	11.46 %
14	4.07 %	4.80 %	6.48 %	6.72 %	8.12 %	11.06 %	7.78 %	9.11 %	12.41 %
15	3.98 %	4.67 %	6.36 %	6.38 %	7.69 %	10.49 %	7.56 %	8.84 %	11.98 %
16	4.09 %	4.83 %	6.59 %	6.69 %	8.14 %	11.16 %	7.71 %	9.08 %	12.42 %
17	4.16 %	5.01 %	6.90 %	7.15 %	8.73 %	11.98 %	8.37 %	9.95 %	13.65 %
18	3.86 %	4.59 %	6.26 %	6.53 %	7.96 %	10.92 %	7.29 %	8.62 %	11.79 %
19	3.68 %	4.39 %	6.01 %	6.51 %	7.86 %	10.71 %	7.24 %	8.48 %	11.50 %
20	4.24 %	4.96 %	6.62 %	6.81 %	8.20 %	11.18 %	7.86 %	9.16 %	12.42 %
Mean		4.83 %			8.28 %			9.30 %	

Table A.6: Intra-distance at 25°C

	$bias = 50\%$			$49\% \leq bias \leq 51\%$			$1\% \leq bias \leq 99\%$		
IC	**500ns**	**500us**	**500ms**	**500ns**	**500us**	**500ms**	**500ns**	**500us**	**500ms**
1	0.29 %	0.31 %	0.15 %	0.84 %	0.89 %	0.44 %	32.44 %	35.82 %	19.65 %
2	0.30 %	0.32 %	0.16 %	0.87 %	0.93 %	0.46 %	32.24 %	35.96 %	19.33 %
3	0.28 %	0.29 %	0.14 %	0.82 %	0.85 %	0.41 %	31.35 %	33.67 %	18.24 %
4	0.29 %	0.31 %	0.15 %	0.85 %	0.91 %	0.44 %	31.80 %	34.75 %	18.69 %
5	0.29 %	0.32 %	0.15 %	0.84 %	0.92 %	0.44 %	32.53 %	36.12 %	19.21 %
6	0.33 %	0.36 %	0.17 %	0.96 %	1.05 %	0.49 %	34.82 %	39.01 %	20.11 %
7	0.30 %	0.32 %	0.16 %	0.87 %	0.94 %	0.46 %	32.99 %	36.54 %	19.88 %
8	0.29 %	0.31 %	0.15 %	0.83 %	0.90 %	0.44 %	31.84 %	35.43 %	19.31 %
9	0.34 %	0.38 %	0.18 %	0.98 %	1.12 %	0.53 %	35.79 %	42.25 %	22.34 %
10	0.30 %	0.34 %	0.16 %	0.87 %	1.00 %	0.47 %	33.44 %	39.03 %	20.31 %
11	0.29 %	0.30 %	0.14 %	0.85 %	0.88 %	0.43 %	32.04 %	34.65 %	19.26 %
12	0.30 %	0.31 %	0.15 %	0.87 %	0.92 %	0.45 %	32.58 %	35.41 %	18.91 %
13	0.28 %	0.28 %	0.14 %	0.79 %	0.82 %	0.41 %	30.25 %	32.45 %	17.85 %
14	0.28 %	0.30 %	0.14 %	0.82 %	0.88 %	0.43 %	31.74 %	35.14 %	19.26 %
15	0.27 %	0.28 %	0.14 %	0.78 %	0.84 %	0.42 %	30.20 %	34.19 %	18.76 %
16	0.29 %	0.31 %	0.15 %	0.83 %	0.89 %	0.44 %	31.75 %	34.90 %	19.34 %
17	0.32 %	0.34 %	0.16 %	0.92 %	1.00 %	0.47 %	33.82 %	38.07 %	20.08 %
18	0.29 %	0.30 %	0.14 %	0.82 %	0.86 %	0.42 %	31.02 %	33.19 %	18.40 %
19	0.27 %	0.28 %	0.14 %	0.78 %	0.81 %	0.40 %	30.74 %	32.84 %	17.64 %
20	0.28 %	0.30 %	0.14 %	0.81 %	0.86 %	0.43 %	32.17 %	35.36 %	20.04 %
Mean	0.29 %	0.31 %	0.15 %	0.85 %	0.91 %	0.44 %	32.28 %	35.74 %	19.33 %

Table A.7: Percentage of unstable cells at 25°C

B. Formal Verification

B.1 ProVerif Protocol Encodings

B.1.1 Common Definitions

```
(* RNG *)
(* Modelled by random nonces *)

(* PUF *)
type puf_key.
fun PUF(bitstring, puf_key): bitstring.

(* CONCAT *)
fun CONCAT2(bitstring, bitstring): bitstring.
fun CONCAT3(bitstring, bitstring, bitstring): bitstring.
fun CONCAT4(bitstring, bitstring, bitstring, bitstring):
    bitstring.

(* HASH *)
fun HASH(bitstring): bitstring.

(* XOR *)
const zeros: bitstring. (* 00..0 *)
fun XOR(bitstring,bitstring):bitstring.
  equation forall x:bitstring, y:bitstring; XOR(XOR(x,y),y) = x.
  equation forall x:bitstring; XOR(x,x) = zeros.  (* XOR(x,x) =
     00..0 *)
  equation forall x:bitstring; XOR(zeros,x) = x.  (* XOR(00..0,x
     ) = x *)
  equation forall x:bitstring; XOR(x,zeros) = x.  (* XOR(x
     ,00..0) = x *)

(* HMAC (Gen. and V) *)
const hmac_opad: bitstring [data].
const hmac_ipad: bitstring [data].
fun HMAC(bitstring, bitstring): bitstring.
equation forall k: bitstring, m: bitstring; HMAC(k, m) = HASH(
```

```
    CONCAT2(XOR(k, hmac_opad), HASH(CONCAT2(XOR(k, hmac_ipad), m)
    ))).

(* Asymmetric cryptography key types *)
type secret_key.
type public_key.

(* ENC and DEC *)

fun PKGEN(secret_key): public_key.
fun ENC(bitstring, public_key): bitstring.

reduc forall m: bitstring, sk: secret_key; DEC(ENC(m,PKGEN(sk)),
    sk) = m.

(* PK SIG and PK VER *)

fun SIG(bitstring, secret_key): bitstring.
reduc forall m: bitstring, sk: secret_key; MSG(SIG(m,sk)) = m.
reduc forall m: bitstring, sk: secret_key; VER(SIG(m,sk), PKGEN(
    sk), m) = true.

(* Communication channel (public) *)
free c:channel.

(* Communication channel (private, modelling a secure
    environment) *)
free sc:channel [private].

(* ACK *)
type acknowledgment.
fun ACK(bitstring): acknowledgment.

(* Converters *)
fun key_to_bitstring(public_key): bitstring [typeConverter].
fun bitstring_to_key(bitstring): public_key [typeConverter].
```

B.1.2 ADS Setup and Verification

```
event verifiedX(public_key, public_key).
event verifiedN(public_key, public_key).

event startX(public_key, public_key).
event startN(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(verifiedN(k1, k2)).
query k1:public_key, k2:public_key; event(verifiedX(k1, k2)).

(* Secrecy queries *)
query attacker(new skX).
query attacker(new skN).
query attacker(new pufKeyX).
query attacker(new pufKeyN).

(* Authentication queries *)
query k1:public_key, k2:public_key; event(verifiedN(k1, k2)) ==>
    event(startX(k1, k2)).
query k1:public_key, k2:public_key; event(verifiedX(k1, k2)) ==>
    event(startN(k1, k2)).

let processX(skX: secret_key, pkX: public_key, pufKeyX: puf_key)
    =
    (* Setup *)
    new cn: bitstring;
    new tN: bitstring;
    out(sc, (tN, cn, pkX));
    in(sc, (kn:bitstring, pkN:public_key));
    event startX(pkN, pkX);
    let rn = HASH(PUF(kn, pufKeyX)) in

    (* Verification *)
    let qxn = SIG(CONCAT2(tN, cn), skX) in
    out(c, (cn, qxn));

    in(c, kn':bitstring);
    let rn' = HASH(PUF(kn', pufKeyX)) in
    if rn' = rn then
```

```
        out(c, ACK(kn'));
        event verifiedX(pkN, pkX).

let processN(skN: secret_key, pkN: public_key, pufKeyN: puf_key)
    =
    (* Setup *)
    in(sc, (tN:bitstring, cn:bitstring, pkX:public_key));
    event startN(pkN, pkX);
    let kn = HASH(PUF(cn, pufKeyN)) in
    out(sc, (kn, pkN));

    (* Verification *)
    in(c, (cn':bitstring, qxn:bitstring));
    if VER(qxn, pkX, CONCAT2(tN, cn)) then
        let kn' = HASH(PUF(cn', pufKeyN)) in
        out(c, kn');
        in(c, ack1:acknowledgment);
        if ack1 = ACK(kn') then
            event verifiedN(pkN, pkX).

process
    new pufKeyX: puf_key;
    new pufKeyN: puf_key;

    new skX: secret_key; let pkX = PKGEN(skX) in out(c, pkX);
 new skN: secret_key; let pkN = PKGEN(skN) in out(c, pkN);

 (!processX(skX, pkX, pufKeyX)) | (!processN(skN, pkN, pufKeyN))
```

B.1.3 ADS Enrolment

```
event verifiedX(public_key, public_key).
event verifiedN(public_key, public_key).

event startX(public_key, public_key).
event startN(public_key, public_key).

event enrolledX(public_key, public_key).
event enrolledN(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(verifiedN(k1, k2)).
query k1:public_key, k2:public_key; event(verifiedX(k1, k2)).
query k1:public_key, k2:public_key; event(enrolledX(k1, k2)).
query k1:public_key, k2:public_key; event(enrolledN(k1, k2)).

(* Secrecy queries *)
query attacker(new skX).
query attacker(new skN).
query attacker(new pufKeyX).
query attacker(new pufKeyN).

(* Authentication queries *)
query k1:public_key, k2:public_key; event(verifiedN(k1, k2)) ==>
    event(startX(k1, k2)).
query k1:public_key, k2:public_key; event(verifiedX(k1, k2)) ==>
    event(startN(k1, k2)).

query k1:public_key, k2:public_key; event(enrolledN(k1, k2)) ==>
    event(verifiedN(k1, k2)).
query k1:public_key, k2:public_key; event(enrolledX(k1, k2)) ==>
    event(verifiedX(k1, k2)).

let processX(skX: secret_key, pkX: public_key, pufKeyX: puf_key)
    =
    (* Setup *)
    new cn: bitstring;
    new tN: bitstring;
    out(sc, (tN, cn, pkX));
    in(sc, (kn:bitstring, pkN:public_key));
```

```
    event startX(pkN, pkX);
    let rn = HASH(PUF(kn, pufKeyX)) in

    (* Verification *)
    let qxn = SIG(CONCAT2(tN, cn), skX) in
    out(c, (cn, qxn));

    in(c, kn':bitstring);
    let rn' = HASH(PUF(kn', pufKeyX)) in
    if rn' = rn then
        out(c, ACK(kn'));
        event verifiedX(pkN, pkX);

    (* Enrolment *)
    let qxn = SIG(key_to_bitstring(pkN), skX) in
    out(c, qxn);
    in(c, ack2: acknowledgment);
    if ack2 = ACK(qxn) then
        event enrolledX(pkN, pkX).

let processN(skN: secret_key, pkN: public_key, pufKeyN: puf_key)
    =
    (* Setup *)
    in(sc, (tN:bitstring, cn:bitstring, pkX:public_key));
    event startN(pkN, pkX);
    let kn = HASH(PUF(cn, pufKeyN)) in
    out(sc, (kn, pkN));

    (* Verification *)
    in(c, (cn':bitstring, qxn:bitstring));
    if VER(qxn, pkX, CONCAT2(tN, cn)) then
        let kn' = HASH(PUF(cn', pufKeyN)) in
        out(c, kn');
        in(c, ack1:acknowledgment);
        if ack1 = ACK(kn') then
            event verifiedN(pkN, pkX);

    (* Enrolment *)
    in(c, qxn: bitstring);
    if VER(qxn, pkX, key_to_bitstring(pkN)) then
```

```
        event enrolledN(pkN, pkX);
        out(c, ACK(qxn)).

process
    new pufKeyX: puf_key;
    new pufKeyN: puf_key;

    new skX: secret_key; let pkX = PKGEN(skX) in out(c, pkX);
 new skN: secret_key; let pkN = PKGEN(skN) in out(c, pkN);

 (!processX(skX, pkX, pufKeyX)) | (!processN(skN, pkN, pufKeyN))
```

B.1.4 ADS Key Exchange

```
event startA(public_key, public_key).
event startB(public_key, public_key).
event exchangedA(public_key, public_key).
event exchangedB(public_key, public_key).

(* Secrecy queries *)
query attacker(new skA).
query attacker(new skB).
query attacker(new pufKeyA).
query attacker(new pufKeyB).

let processA(skA: secret_key, pkA: public_key, pufKeyA: puf_key,
    pkX:public_key, qXA: bitstring) =
    out(c, (pkX, pkA, qXA));
    in(c, (pkB:public_key, qXB: bitstring));
    event startA(pkA, pkB);
    if VER(qXB, pkX, key_to_bitstring(pkB)) then
        out(c, ACK(qXB));
        event exchangedA(pkA, pkB).

let processB(skB: secret_key, pkB: public_key, pufKeyB: puf_key,
    pkX:public_key, qXB: bitstring) =
    in(c, (pkX':public_key, pkA:public_key, qXA:bitstring));
    if pkX' = pkX then
        event startB(pkA, pkB);
        if VER(qXA, pkX, key_to_bitstring(pkA)) then
            out(c, (pkB, qXB));
            in(c, ack:acknowledgment);
            if ack = ACK(qXB) then
                event exchangedB(pkA, pkB).

process
    new pufKeyA: puf_key;
    new pufKeyB: puf_key;

    new skA: secret_key; let pkA = PKGEN(skA) in out(c, pkA);
 new skB: secret_key; let pkB = PKGEN(skB) in out(c, pkB);

    (* The following statements model the storage of the public
```

```
    key of the AD and the signed public keys of the nodes. *)
  new skX: secret_key; let pkX = PKGEN(skX) in out(c, pkX);
  let qXA = SIG(key_to_bitstring(pkA), skX) in
  let qXB = SIG(key_to_bitstring(pkB), skX) in

(!processA(skA, pkA, pufKeyA, pkX, qXA)) | (!processB(skB, pkB,
    pufKeyB, pkX, qXB))
```

B.1.5 ADS Mutual Authentication

```
event startA(public_key, public_key).
event startB(public_key, public_key).

event authenticatedA(public_key, public_key).
event authenticatedB(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(authenticatedA(k1, k2)
    ).
query k1:public_key, k2:public_key; event(authenticatedB(k1, k2)
    ).

(* Secrecy queries *)
query attacker(new skA).
query attacker(new skB).
query attacker(new pufKeyA).
query attacker(new pufKeyB).

(* Authentication queries *)
query k1:public_key, k2:public_key; event(authenticatedA(k1, k2)
    ) ==> event(startB(k1, k2)).
query k1:public_key, k2:public_key; event(authenticatedB(k1, k2)
    ) ==> event(startA(k1, k2)).

let processA(skA: secret_key, pkA: public_key, pufKeyA: puf_key,
    pkX:public_key, qXA: bitstring, pkB: public_key, qXB:
    bitstring) =
    new tA: bitstring;
    event startA(pkA, pkB);
    let qA = SIG(CONCAT2(tA, qXA), skA) in
    out(c, (pkX, tA, qA));
    in(c, (tB:bitstring, qB: bitstring));
    if VER(qB, pkB, CONCAT3(tB, tB, qXB)) then
        event authenticatedA(pkA, pkB);
        out(c, ACK(qB)).

let processB(skB: secret_key, pkB: public_key, pufKeyB: puf_key,
    pkX:public_key, qXB: bitstring, pkA: public_key, qXA:
    bitstring) =
```

```
    in(c, (pkX':public_key, tA:bitstring, qA: bitstring));
    event startB(pkA,pkB);
    if pkX' = pkX then
        if VER(qA, pkA, CONCAT2(tA, qXA)) then
            new tB: bitstring;
            let qB = SIG(CONCAT3(tB, tB, qXB), skB) in
            out(c, (tB, qB));
            in(c, ack:acknowledgment);
            if ack = ACK(qB) then
                event authenticatedB(pkA, pkB).

process
    new pufKeyA: puf_key;
    new pufKeyB: puf_key;

    new skA: secret_key; let pkA = PKGEN(skA) in out(c, pkA);
 new skB: secret_key; let pkB = PKGEN(skB) in out(c, pkB);

    (* The following statements model the storage of the public
       key of the AD and the signed public keys of the nodes. *)
    new skX: secret_key; let pkX = PKGEN(skX) in out(c, pkX);
    let qXA = SIG(key_to_bitstring(pkA), skX) in
    let qXB = SIG(key_to_bitstring(pkB), skX) in

 (!processA(skA, pkA, pufKeyA, pkX, qXA, pkB, qXB)) | (!processB
    (skB, pkB, pufKeyB, pkX, qXB, pkA, qXA))
```

B.1.6 ADS Decommission

```
event startX(public_key, public_key).
event startN(public_key, public_key).

event decommissionedX(public_key, public_key).
event decommissionedN(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(decommissionedN(k1, k2
    )).
query k1:public_key, k2:public_key; event(decommissionedX(k1, k2
    )).

(* Secrecy queries *)
query attacker(new skX).
query attacker(new skN).
query attacker(new pufKeyX).
query attacker(new pufKeyN).

(* Authentication queries *)
query k1:public_key, k2:public_key; event(decommissionedN(k1, k2
    )) ==> event(startX(k1, k2)).

let processX(skX: secret_key, pkX: public_key, pufKeyX: puf_key,
    pkN:public_key) =
    out(c, pkX);
    event startX(pkN, pkX);
    in(c, t:bitstring);
    let q = SIG(t, skX) in
        out(c, q);
        in(c, ack:acknowledgment);
        if ack = ACK(q) then
            event decommissionedX(pkN, pkX).

let processN(skN: secret_key, pkN: public_key, pufKeyN: puf_key,
    pkX:public_key) =
    in(c, pkX':public_key);
    if pkX' = pkX then
        event startN(pkN, pkX);
        new t: bitstring;
```

```
        out(c, t);
        in(c, q:bitstring);
        if VER(q, pkX, t) then
            event decommissionedN(pkN, pkX);
            out(c, ACK(q)).

process
    new pufKeyX: puf_key;
    new pufKeyN: puf_key;

    new skX: secret_key; let pkX = PKGEN(skX) in out(c, pkX);
 new skN: secret_key; let pkN = PKGEN(skN) in out(c, pkN);

 (!processX(skX, pkX, pufKeyX, pkN)) | (!processN(skN, pkN,
    pufKeyN, pkX))
```

B.1.7 Ratchet Authorisation

```
event startX(public_key, public_key).
event startA(public_key, public_key).
event startB(public_key, public_key).

event authorisedX(public_key, public_key).
event authorisedA(public_key, public_key).
event authorisedB(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(authorisedX(k1, k2)).
query k1:public_key, k2:public_key; event(authorisedA(k1, k2)).
query k1:public_key, k2:public_key; event(authorisedB(k1, k2)).

(* Secrecy queries *)
query attacker(new skX).
query attacker(new skA).
query attacker(new skB).
query attacker(new pufKeyX).
query attacker(new pufKeyA).
query attacker(new pufKeyB).

query attacker(new tAB).

(* Authentication queries *)
query k1:public_key, k2:public_key; event(authorisedA(k1, k2))
  ==> event(startX(k1, k2)).
query k1:public_key, k2:public_key; event(authorisedB(k1, k2))
  ==> event(startX(k1, k2)).

let processX(skX: secret_key, pkX: public_key, pufKeyX: puf_key,
    pkA:public_key, pkB:public_key) =
    new tAB: bitstring;
    out(c, pkB);
    in(c, tA:bitstring);
    event startX(pkX, pkA);
    let qXA = SIG(CONCAT4(tAB, tA, key_to_bitstring(pkA),
       key_to_bitstring(pkB)), skX) in
    let etA = ENC(tAB, pkA) in
    out(c, (qXA, etA));
```

```
    in(c, ackA:acknowledgment);
    if ackA = ACK(etA) then
        event authorisedX(pkX, pkA).

let processA(skA: secret_key, pkA: public_key, pufKeyA: puf_key,
    pkX:public_key, pkB:public_key) =
    in(c, pkB':public_key);
    if pkB' = pkB then
        new tA: bitstring;
        out(c, tA);
        in(c, (qXA:bitstring, etA:bitstring));
        event startA(pkX, pkA);
        let tAB = DEC(etA, skA) in
        if VER(qXA, pkX, CONCAT4(tAB, tA, key_to_bitstring(pkA),
            key_to_bitstring(pkB))) then
            event authorisedA(pkX, pkA);
            out(c, ACK(etA)).

process
    new pufKeyX: puf_key;
    new pufKeyA: puf_key;
    new pufKeyB: puf_key;

    new skX: secret_key; let pkX = PKGEN(skX) in out(c, pkX);
 new skA: secret_key; let pkA = PKGEN(skA) in out(c, pkA);
 new skB: secret_key; let pkB = PKGEN(skB) in out(c, pkB);

    (* pkX, pkA, pkB are stored in the nodes and the AD during
       their introduction via the ADS protocols *)

 (!processX(skX, pkX, pufKeyX, pkA, pkB)) | (!processA(skA, pkA,
     pufKeyA, pkX, pkB))
```

B.1.8 CRP Ratchet Initialisation

```
event startA(public_key, public_key).
event startB(public_key, public_key).

event initialisedA(public_key, public_key).
event initialisedB(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(initialisedA(k1, k2)).
query k1:public_key, k2:public_key; event(initialisedB(k1, k2)).

(* Secrecy queries *)
query attacker(new skA).
query attacker(new skB).
query attacker(new pufKeyA).
query attacker(new pufKeyB).
query attacker(new tAB).

let processA(skA: secret_key, pkA: public_key, pufKeyA: puf_key,
    pkB:public_key, tAB: bitstring) =
    new cB0: bitstring;
    let qAC = SIG(CONCAT2(cB0, tAB), skA) in
    out(c, (cB0, qAC));
    event startB(pkB, pkA);
    in(c, (erB0:bitstring, cA0:bitstring, qB:bitstring));
    let rB0 = DEC(erB0, skA) in
    if VER(qB, pkB, CONCAT3(cA0, rB0, tAB)) then
        let kA0 = XOR(rB0, cA0) in
        let rA0 = PUF(cA0, pufKeyA) in
        let erA0 = ENC(rA0, pkB) in
        let qAR = SIG(CONCAT2(rA0,tAB), skA) in
        out(c, (erA0, qAR));
        in(c, ack:acknowledgment);
        if ack = ACK(erA0) then
            event initialisedA(pkA, pkB).

let processB(skB: secret_key, pkB: public_key, pufKeyB: puf_key,
    pkA:public_key, tAB: bitstring) =
    in(c, (cB0:bitstring, qAC:bitstring));
    event startB(pkA, pkB);
```

```
    if VER(qAC, pkA, CONCAT2(cB0, tAB)) then
        let rB0 = PUF(cB0, pufKeyB) in
        let erB0 = ENC(rB0, pkA) in
        new cA0: bitstring;
        let qB = SIG(CONCAT3(cA0, rB0, tAB), skB) in
        out(c, (erB0, cA0, qB));
        in(c, (erA0: bitstring, qAR:bitstring));
        let rA0 = DEC(erA0, skB) in
        if VER(qAR, pkA, CONCAT2(rA0, tAB)) then
        let kB0 = XOR(rA0, cB0) in
        event initialisedB(pkB, pkA);
        out(c, ACK(erA0)).

process
    new pufKeyA: puf_key;
    new pufKeyB: puf_key;

 new skA: secret_key; let pkA = PKGEN(skA) in out(c, pkA);
 new skB: secret_key; let pkB = PKGEN(skB) in out(c, pkB);

    (* pkA, pkB are stored in the nodes and the AD during their
       introduction via the ADS protocols *)

    (* tAB is stored in the nodes during the ratchet
       authorisation *)
    new tAB: bitstring;

 (!processA(skA, pkA, pufKeyA, pkB, tAB)) | (!processB(skB, pkB,
     pufKeyB, pkA, tAB))
```

B.1.9 CRP Ratchet Step

```
event startA(public_key, public_key).
event startB(public_key, public_key).

event authenticatedA(public_key, public_key).
event authenticatedB(public_key, public_key).

(* Reachability queries (the desired result is FALSE) *)
query k1:public_key, k2:public_key; event(authenticatedA(k1, k2)
    ).
query k1:public_key, k2:public_key; event(authenticatedB(k1, k2)
    ).

(* Secrecy queries *)
query attacker(new skA).
query attacker(new skB).
query attacker(new pufKeyA).
query attacker(new pufKeyB).

(* Authentication queries *)
query k1:public_key, k2:public_key; event(authenticatedA(k1, k2)
    ) ==> event(startB(k1, k2)).
query k1:public_key, k2:public_key; event(authenticatedB(k1, k2)
    ) ==> event(startA(k1, k2)).

let processA(skA: secret_key, pkA: public_key, pufKeyA: puf_key,
    pkB:public_key, kAprev: bitstring, cBprev:bitstring) =
    new cB: bitstring;
    let xcA = HMAC(kAprev, CONCAT3(cBprev, cB, key_to_bitstring(
        pkA))) in
    out(c, (cBprev, cB, xcA));
    event startA(pkB, pkA);

    in(c, (erB:bitstring, cAprev:bitstring, cA:bitstring, xcB:
        bitstring));
    let kBprev = XOR(PUF(cAprev, pufKeyA), cBprev) in
    let k = XOR(kAprev, kBprev) in
    if HMAC(k, CONCAT4(cAprev, cA, erB, cB)) = xcB then
        let rA = PUF(cA, pufKeyA) in
        let erA = XOR(rA, k) in
```

```
        let xrA = HMAC(k, CONCAT3(erA, cA, cB)) in
        out(c, (erA, xrA));
        in(c, ack:acknowledgment);
        if ack = ACK(xrA) then
            let rB = XOR(erB, k) in
            let kA = XOR(rB, cA) in
                event authenticatedA(pkA, pkB).

let processB(skB: secret_key, pkB: public_key, pufKeyB: puf_key,
    pkA:public_key, kBprev: bitstring, cAprev:bitstring) =
    in(c, (cBprev:bitstring, cB:bitstring, xcA:bitstring));
    event startB(pkA, pkB);

    let kAprev = XOR(PUF(cBprev, pufKeyB), cAprev) in
    let k = XOR(kAprev, kBprev) in
    if HMAC(kAprev, CONCAT3(cBprev, cB, key_to_bitstring(pkA)))
      = xcA then
        let rB = PUF(cB, pufKeyB) in
        new cA: bitstring;
        let erB = XOR(rB, k) in
        let xcB = HMAC(k, CONCAT4(cAprev, cA, erB, cB)) in
        out(c, (erB, cAprev, cA, xcB));

        in(c, (erA:bitstring, xrA:bitstring));
        if HMAC(k, CONCAT3(erA, cA, cB)) = xrA then
            event authenticatedB(pkB, pkA);
            out(c, ACK(xrA));
            let rA = XOR(erA, k) in
            let kB = XOR(erA, cB) in
                0.

process
    new pufKeyA: puf_key;
    new pufKeyB: puf_key;

 new skA: secret_key; let pkA = PKGEN(skA) in out(c, pkA);
 new skB: secret_key; let pkB = PKGEN(skB) in out(c, pkB);

    (* pkA, pkB are stored in the nodes and the AD during their
       introduction via the ADS protocols *)
```

```
    (*  kAprev, kBprev, cAprev, cBprev stored in the nodes
       during the previous ratchet step or the ratchet
       initialisation *)
    new cAprev: bitstring;
    new cBprev: bitstring;

    let kAprev = XOR(PUF(cBprev, pufKeyB), cAprev) in
    let kBprev = XOR(PUF(cAprev, pufKeyA), cBprev) in

(!processA(skA, pkA, pufKeyA, pkB, kAprev, cBprev)) | (!
    processB(skB, pkB, pufKeyB, pkA, kBprev, cAprev))
```

Query	Result
All protocols	
query attacker(new skX)	True
query attacker(new skN)	True
query attacker(new pufKeyX)	True
query attacker(new pufKeyN)	True
Setup and Verification	
query k1:public_key, k2:public_key; event(verifiedN(k1, k2)) ==> event(startX(k1, k2)).	True
query k1:public_key, k2:public_key; event(verifiedX(k1, k2)) ==> event(startN(k1, k2)).	True
Enrolment	
query k1:public_key, k2:public_key; event(verifiedN(k1, k2)) ==> event(startX(k1, k2)).	True
query k1:public_key, k2:public_key; event(verifiedX(k1, k2)) ==> event(startN(k1, k2)).	True
query k1:public_key, k2:public_key; event(enrolledN(k1, k2)) ==> event(verifiedN(k1, k2)).	True
query k1:public_key, k2:public_key; event(enrolledX(k1, k2)) ==> event(verifiedX(k1, k2)).	True
Decommission	
query k1:public_key, k2:public_key; event(decommissionedN(k1, k2)) ==> event(startX(k1, k2)).	True
Mutual Authentication	
query k1:public_key, k2:public_key; event(authenticatedA(k1, k2)) ==> event(startB(k1, k2)).	True
query k1:public_key, k2:public_key; event(authenticatedB(k1, k2)) ==> event(startA(k1, k2)).	True

Table B.1: ProVerif queries and results for the Authority Device Scheme

Query	Result
CRP Ratchet Authorisation	
query attacker(new skX).	True
query attacker(new skA).	True
query attacker(new skB).	True
query attacker(new pufKeyX).	True
query attacker(new pufKeyA).	True
query attacker(new pufKeyB).	True
query attacker(new tAB).	True
query k1:public_key, k2:public_key; event(authorisedA(k1, k2)) ==> event(startX(k1, k2)).	True
query k1:public_key, k2:public_key; event(authorisedB(k1, k2)) ==> event(startX(k1, k2)).	True
CRP Ratchet Initialisation	
query attacker(new skA).	True
query attacker(new skB).	True
query attacker(new pufKeyA).	True
query attacker(new pufKeyB).	True
query attacker(new tAB).	True
CRP Ratchet Step	
query attacker(new skA).	True
query attacker(new skB).	True
query attacker(new pufKeyA).	True
query attacker(new pufKeyB).	True
query k1:public_key, k2:public_key; event(authenticatedA(k1, k2)) ==> event(startB(k1, k2)).	True
query k1:public_key, k2:public_key; event(authenticatedB(k1, k2)) ==> event(startA(k1, k2)).	True

Table B.2: ProVerif queries and results for the Continuous Pairwise Authentication protocols

C. Proof-of-Concept Implementation

Header	Value	Hexadecimal Value
Phase	Null	00
	Setup	01
	Verification	02
	Enrolment	03
	Decommission	04
	Authentication	05
	Key Exchange	06
	Authority Device Authorisation	07
	Ratchet Setup	08
	Ratchet Step	09
	Ratchet ZK Setup	0A
	Ratchet ZK Step	0B
Command	Null	00
	PUF Challenge	01
	PUF Response	02
	Public Key	03
	Acknowledgement	04
	Failure	05
	Initiate	06
	Signature	07
	Nonce	08
	Id	09
	Commitment	0A
	Proof	0B

Table C.1: Message header values

	25°C		100°C	
	Repetition P_{fail}	Final P_{fail}	Repetition P_{fail}	Final P_{fail}
500ns	0.006781	1.66×10^{-11}	0.007641	9.36×10^{-11}
5us	0.008440	3.88×10^{-10}	0.008856	7.68×10^{-10}
50us	0.013056	1.57×10^{-7}	0.007734	1.11×10^{-10}
500us	0.013250	1.91×10^{-7}	0.003307	3.33×10^{-16}
5ms	0.009470	1.97×10^{-9}	0.001840	$<10^{-16}$
50ms	0.004219	1.43×10^{-14}	0.001460	1.11×10^{-16}
500ms	0.001009	$<10^{-16}$	0.000360	1.11×10^{-16}
	25°C aged		**100°C aged**	
	Repetition P_{fail}	Final P_{fail}	Repetition P_{fail}	Final P_{fail}
500ns	0.005673	1.21×10^{-12}	0.010522	8.52×10^{-9}
5us	0.010034	4.41×10^{-9}	0.010772	1.18×10^{-8}
50us	0.016448	3.10×10^{-6}	0.008578	4.88×10^{-10}
500us	0.012258	6.80×10^{-8}	0.006500	8.95×10^{-12}
5ms	0.009472	1.98×10^{-9}	0.003636	1.55×10^{-15}
50ms	0.005009	1.90×10^{-13}	0.001797	$<10^{-16}$
500ms	0.001575	2.22×10^{-16}	0.000296	$<10^{-16}$

Table C.2: Error correction failure probability

Repetitions	500ns	5us	50us	500us	5ms	50ms	500ms
1	12.79 %	13.58 %	15.32 %	15.38 %	14.01 %	11.24 %	7.68 %
3	5.86 %	6.04 %	6.73 %	6.64 %	5.95 %	4.87 %	3.42 %
5	3.86 %	3.91 %	4.31 %	4.27 %	3.75 %	2.94 %	2.18 %
7	3.51 %	3.55 %	3.85 %	3.79 %	3.32 %	2.58 %	1.93 %
9	3.33 %	3.37 %	3.61 %	3.55 %	3.10 %	2.40 %	1.81 %
11	3.22 %	3.26 %	3.46 %	3.40 %	2.96 %	2.29 %	1.74 %
13	3.13 %	3.18 %	3.35 %	3.30 %	2.86 %	2.22 %	1.69 %
15	3.07 %	3.11 %	3.27 %	3.21 %	2.77 %	2.16 %	1.65 %
17	3.02 %	3.06 %	3.21 %	3.15 %	2.71 %	2.11 %	1.62 %
19	2.98 %	3.02 %	3.16 %	3.09 %	2.66 %	2.07 %	1.59 %

Table C.3: Maximum intra-distance at 25°C

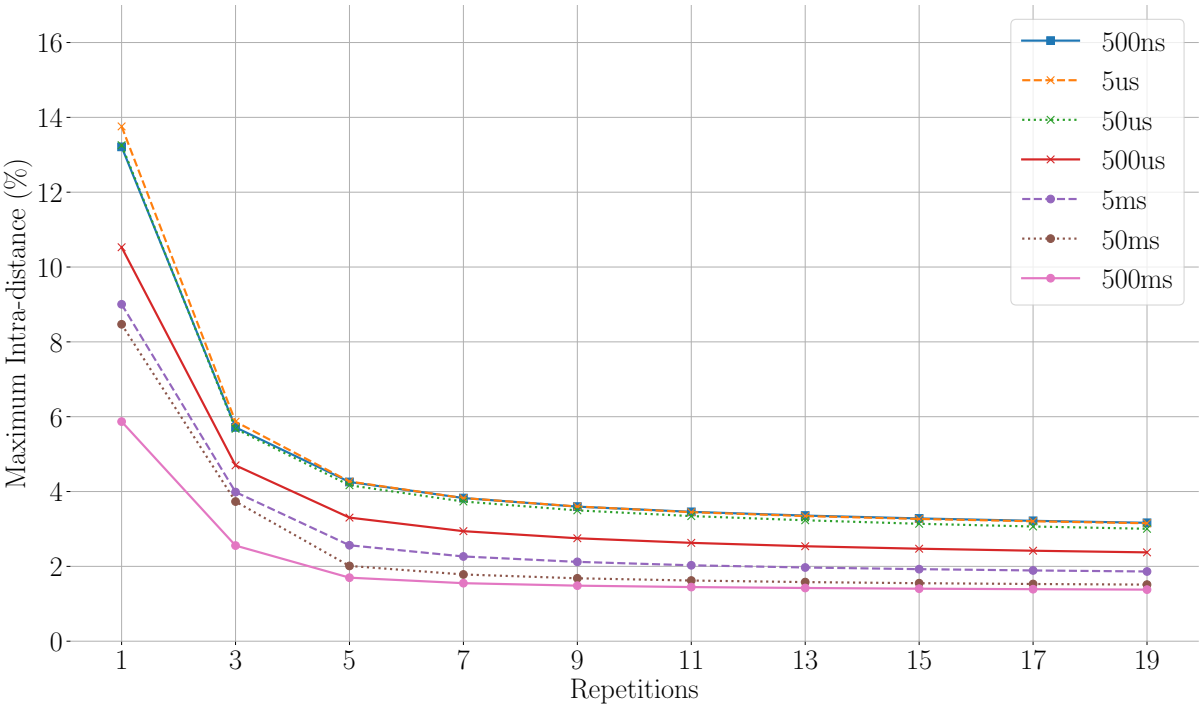


Figure C.1: Maximum intra-distance at 100°C

Repetitions	500ns	5us	50us	500us	5ms	50ms	500ms
1	13.21 %	13.76 %	13.26 %	10.53 %	9.00 %	8.47 %	5.87 %
3	5.71 %	5.87 %	5.68 %	4.70 %	3.99 %	3.73 %	2.56 %
5	4.26 %	4.27 %	4.17 %	3.30 %	2.56 %	2.01 %	1.70 %
7	3.83 %	3.83 %	3.73 %	2.94 %	2.26 %	1.78 %	1.55 %
9	3.60 %	3.60 %	3.50 %	2.75 %	2.12 %	1.68 %	1.48 %
11	3.46 %	3.45 %	3.34 %	2.63 %	2.03 %	1.62 %	1.45 %
13	3.36 %	3.34 %	3.23 %	2.54 %	1.97 %	1.58 %	1.42 %
15	3.28 %	3.27 %	3.14 %	2.47 %	1.93 %	1.55 %	1.40 %
17	3.22 %	3.21 %	3.07 %	2.42 %	1.89 %	1.53 %	1.39 %
19	3.16 %	3.15 %	3.00 %	2.37 %	1.86 %	1.51 %	1.38 %

Table C.4: Maximum intra-distance at 100°C

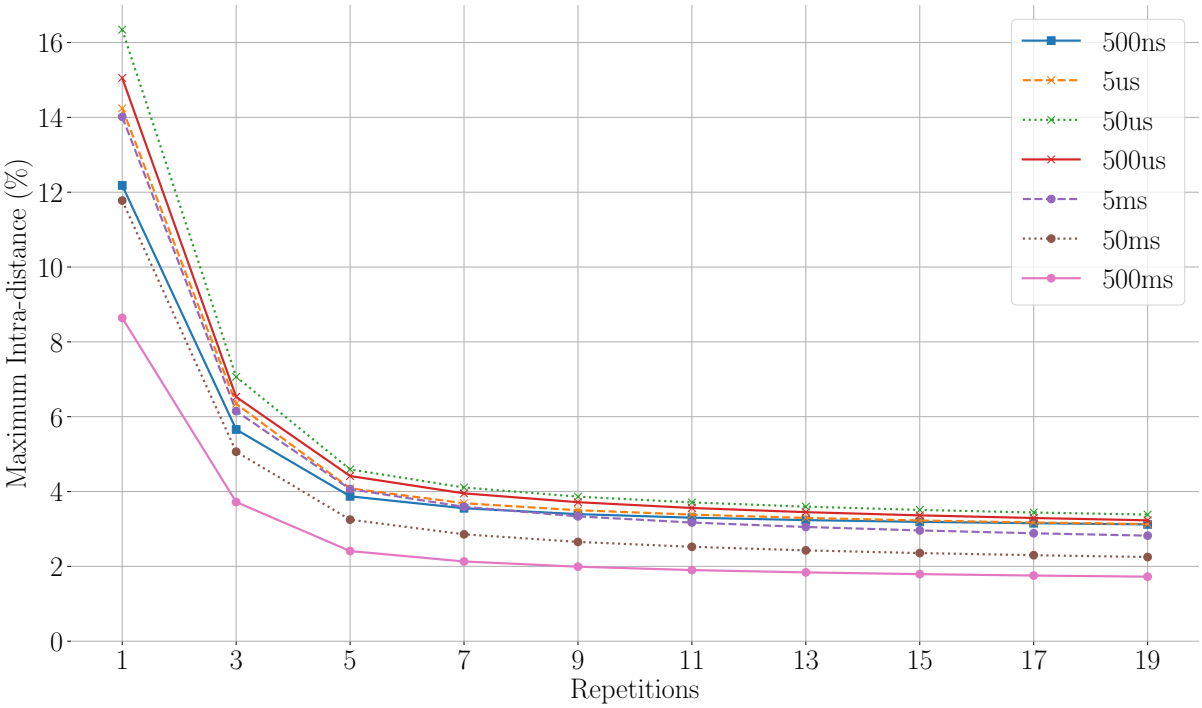


Figure C.2: Maximum intra-distance at 25°C (aged)

Repetitions	500ns	5us	50us	500us	5ms	50ms	500ms
1	12.18 %	14.24 %	16.34 %	15.05 %	14.02 %	11.78 %	8.64 %
3	5.66 %	6.35 %	7.07 %	6.53 %	6.15 %	5.06 %	3.72 %
5	3.87 %	4.08 %	4.59 %	4.41 %	4.06 %	3.25 %	2.41 %
7	3.55 %	3.69 %	4.11 %	3.95 %	3.59 %	2.86 %	2.13 %
9	3.40 %	3.50 %	3.86 %	3.71 %	3.34 %	2.65 %	1.99 %
11	3.30 %	3.38 %	3.71 %	3.56 %	3.17 %	2.52 %	1.90 %
13	3.24 %	3.29 %	3.60 %	3.45 %	3.05 %	2.43 %	1.84 %
15	3.19 %	3.23 %	3.51 %	3.36 %	2.96 %	2.36 %	1.79 %
17	3.15 %	3.17 %	3.44 %	3.29 %	2.88 %	2.30 %	1.75 %
19	3.12 %	3.13 %	3.38 %	3.23 %	2.82 %	2.25 %	1.72 %

Table C.5: Maximum intra-distance at 25°C (aged)

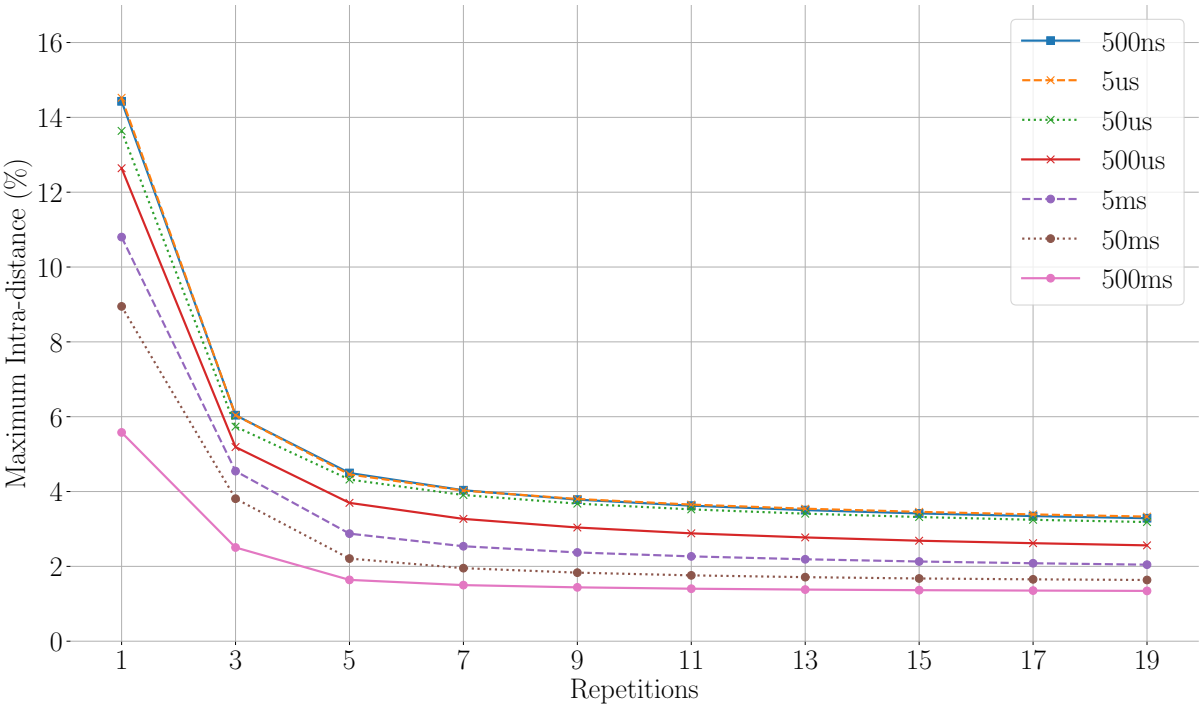


Figure C.3: Maximum intra-distance at 100°C (aged)

Repetitions	500ns	5us	50us	500us	5ms	50ms	500ms
1	14.43 %	14.52 %	13.64 %	12.64 %	10.80 %	8.95 %	5.58 %
3	6.04 %	6.04 %	5.73 %	5.19 %	4.55 %	3.81 %	2.50 %
5	4.50 %	4.46 %	4.32 %	3.70 %	2.87 %	2.21 %	1.64 %
7	4.03 %	4.02 %	3.91 %	3.27 %	2.54 %	1.95 %	1.50 %
9	3.78 %	3.80 %	3.68 %	3.04 %	2.37 %	1.83 %	1.44 %
11	3.62 %	3.65 %	3.52 %	2.88 %	2.27 %	1.76 %	1.40 %
13	3.50 %	3.54 %	3.41 %	2.77 %	2.19 %	1.71 %	1.38 %
15	3.41 %	3.46 %	3.32 %	2.69 %	2.13 %	1.68 %	1.36 %
17	3.34 %	3.39 %	3.24 %	2.62 %	2.08 %	1.65 %	1.35 %
19	3.29 %	3.33 %	3.18 %	2.56 %	2.05 %	1.64 %	1.34 %

Table C.6: Maximum intra-distance at 100°C (aged)

	Setup		Verification		Enrolment		Enrolment (multiple ownership)			Decommission		Key Exchange		Mutual Authentication	
	X	N	X	N	X	N	X	Y	N	X	N	A	B	A	B
SIG	0	0	1	0	1	0	1	1	0	1	0	0	0	1	1
VER	0	0	0	1	0	1	0	1	2	0	1	1	1	1	1
PUF	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
RNG	2	0	0	0	0	0	0	0	0	0	1	0	0	1	1
HASH	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
CONCAT	0	0	1	1	0	0	1	1	1	0	0	0	0	3	3
TX	2	1	2	1	1	1	1	3	1	2	2	2	1	2	1
RX	1	2	1	2	1	1	2	2	1	2	2	1	2	1	2

Table C.7: Execution count of basic operations in ADS protocols

	Ratchet Authorisation			CRP Ratchet Initialisation		CRP Ratchet Step		ZK CRP Ratchet Initialisation		ZK CRP Ratchet Step	
	X	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
ENC	2	0	0	1	1	0	0	0	0	0	0
DEC	0	1	1	1	1	0	0	0	0	0	0
SIG	2	0	0	2	1	0	0	2	1	0	0
VER	0	1	1	1	2	0	0	1	2	0	0
ZKC	0	0	0	0	0	0	0	1	1	1	1
ZKP	0	0	0	0	0	0	0	0	0	λ	λ
ZKV	0	0	0	0	0	0	0	0	0	λ	λ
PUF	0	0	0	1	1	2	2	1	1	2	2
RNG	1	1	1	1	1	1	1	1	0	$\lambda+1$	$\lambda+1$
XOR	0	0	0	1	1	5	5	0	0	0	0
HASH	0	0	0	0	0	0	0	0	0	2	2
CONCAT	6	3	3	4	4	7	7	7	7	2	2
HMG	0	0	0	0	0	2	1	0	0	0	0
HMV	0	0	0	0	0	1	2	0	0	0	0
TX	4	2	2	2	2	2	2	2	2	λ	$\lambda+1$
RX	4	2	2	2	2	2	2	2	2	$\lambda+1$	λ

Table C.8: Execution count of basic operations in Continuous Pairwise Authentication protocols

References

[1] K. Goutsos and A. Bystrov, "Lightweight PUF-based Continuous Authentication Protocol," in *2019 International Conference on Computing, Electronics Communications Engineering (iCCECE)*, 2019, pp. 229–234.

[2] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, vol. IX, pp. 5–38, 1883.

[3] WEForum. (2019). "The Next Economic Growth Engine: Scaling Fourth Industrial Revolution Technologies in Production," [Online]. Available: https://www.weforum.org/whitepapers/the-next-economic-growth-engine-scaling-fourth-industrial-revolution-technologies-in-production/ (visited on 22/10/2019).

[4] PricewaterhouseCoopers. (2019). "The Internet of Things: The next growth engine for the semiconductor industry," [Online]. Available: https://www.pwc.com/gx/en/industries/technology/publications/internet-of-things.html (visited on 22/10/2019).

[5] R. Pappu, "Physical one-way functions.," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2001.

[6] M. Marlinspike and T. Perrin, "The Double Ratchet Algorithm."

[7] D. Loshin, "Knowledge integrity: Data ownership," *Data Ownership, US Department of Health and Human Services*, 2002.

[8] Trusted Computing Group, "Trusted Platform Module Library Part 1: Architecture," 2014.

[9] D. Airehrour, J. Gutierrez and S. K. Ray, "Secure routing for internet of things: A survey," *Journal of Network and Computer Applications*, vol. 66, pp. 198–213, 2016.

[10] R. Maes, *Physically Unclonable Functions: Constructions, Properties and Applications*. Springer Berlin Heidelberg, 2013, 1-185, ISBN: 978-3-642-41395-7.

[11] N. Ferguson and B. Schneier, *Cryptography Engineering: Design Principles and Practical Applications*. Wiley, 2010, 353 pp., ISBN: 0-470-47424-6.

[12] U. Rührmair, "Towards Secret-Free Security," 2019.

[13] M. Bellare, A. C. Singh, J. Jaeger, M. Nyayapati and I. Stepanovs, "Ratcheted encryption and key exchange: The security of messaging," in *Advances in Cryptology – CRYPTO 2017*, 2017, pp. 619–650.

[14] C. Brzuska, M. Fischlin, H. Schröder and S. Katzenbeisser, "Physically uncloneable functions in the universal composition framework," in *Advances in Cryptology – CRYPTO 2011*, 2011, pp. 51–70.

[15] 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 Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 146–159, 2016.

[16] S. Kerr, M. S. Kirkpatrick and E. Bertino, "PEAR," in *3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, 2010, p. 18.

[17] R. Roman, P. Najera and J. Lopez, "Securing the Internet of Things," *Computer*, vol. 44, no. 9, pp. 51–58, 2011.

[18] O. Arias, J. Wurm, K. Hoang and Y. Jin, "Privacy and Security in Internet of Things and Wearable Devices," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 2, pp. 99–109, 2015.

[19] T. R. Peltier, *Information Security Fundamentals*. CRC press, 2013.

[20] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*. Springer Berlin Heidelberg, 2003, ISBN: 978-3-642-07716-6.

[21] K. Yoneyama, R. Yoshida, Y. Kawahara, T. Kobayashi, H. Fuji and T. Yamamoto, *Multi-Cast Key Distribution: Scalable, Dynamic and Provably Secure Construction*, 2016.

[22] A. Khalili, J. Katz and W. Arbaugh, "Toward secure key distribution in truly ad-hoc networks," in *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, 2003, pp. 342–346.

[23] S. Camtepe, B. Yener and M. Yung, "Expander Graph based Key Distribution Mechanisms in Wireless Sensor Networks," in *2006 IEEE International Conference on Communications*, vol. 5, 2006, pp. 2262–2267.

[24] J. Katz, A. J. Menezes, P. C. Van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC press, 1996.

[25] P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," in, 1999, pp. 388–397.

[26] R. Spreitzer, V. Moonsamy, T. Korak and S. Mangard, "Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 465–488, 2018.

[27] N. Sklavos, R. Chaves, G. Di Natale and R. Francesco, *Hardware Security and Trust*. 2017, ISBN: 978-3-319-44316-4.

[28] D. Karaklajić, J. M. Schmidt and I. Verbauwhede, "Hardware Designer's Guide to Fault Attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 12, pp. 2295–2306, 2013.

[29] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, 2014, pp. 361–372.

[30] M. Tehranipoor and C. Wang, Eds., *Introduction to Hardware Security and Trust*. Springer New York, 2012, ISBN: 978-1-4419-8079-3.

[31] R. Anderson, M. Bond, J. Clulow and S. Skorobogatov, "Cryptographic processors - A survey," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 357–369, 2006.

[32] N. Sklavos, "On the Hardware Implementation Cost of Crypto-Processors Architectures," *Information Security Journal: A Global Perspective*, vol. 19, no. 2, pp. 53–60, 2010.

[33] M. Sabt, M. Achemlal and A. Bouabdallah, "Trusted Execution Environment: What It is, and What It is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 57–64.

[34] J. Winter, "Trusted Computing Building Blocks for Embedded Linux-based ARM Trustzone Platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, (Alexandria, Virginia, USA), 2008, pp. 21–30.

[35] Thales Group. (2020). "Luna Network Hardware Security Modules (HSMs)," [Online]. Available: https://cpl.thalesgroup.com/encryption/hardware-security-modules/network-hsms (visited on 28/06/2020).

[36] Thales Group, "Risk management strategies for digital processes with HSMs." (visited on 28/06/2020).

[37] nCipher Security. (2020). "Cryptographic Solutions Delivering Cloud, IoT, Blockchain and Digital Payment Security," [Online]. Available: https://www.ncipher.com/ (visited on 28/06/2020).

[38] N. S. A. GmbH. (2020). "MIFARE ICs | MIFARE," [Online]. Available: https://www.mifare.net/en/products/chip-card-ics/ (visited on 28/06/2020).

[39] Secure Tech Alliance. (2020). "About Smart Cards : Applications," [Online]. Available: https://www.securetechalliance.org/smart-cards-applications/ (visited on 28/06/2020).

[40] M. Gulati, M. J. Smith and S. Y. Yu, "Security enclave processor for a system on a chip," U.S. Patent 8832465B2, 2014.

[41] Microsoft Corporation. (2020). "TPM recommendations in Windows 10," [Online]. Available: https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/tpm-recommendations (visited on 28/06/2020).

[42] Arch Linux. (2020). "Trusted Platform Module - Arch Linux," [Online]. Available: https://wiki.archlinux.org/index.php/Trusted_Platform_Module (visited on 28/06/2020).

[43] Trusted Computing Group. (2019). "Trusted Computing Group," [Online]. Available: https://trustedcomputinggroup.org/ (visited on 20/11/2019).

[44] U. Rührmair, S. Devadas and F. Koushanfar, "Security Based on Physical Unclonability and Disorder," in *Introduction to Hardware Security and Trust*, 2012, pp. 65–102.

[45] M. Yamakoshi, J. Tanaka, M. Furuie, M. Hirabayashi and T. Matsumoto, "Individuality evaluation for paper based artifact-metrics using transmitted light image," in *Security, Forensics, Steganography, and Watermarking of Multimedia Contents X*, vol. 6819, 2008, 68190H.

[46] D. R. Stinson and M. Paterson, *Cryptography: Theory and Practice*. CRC press, 2018.

[47] J. P. Aumasson, *Serious Cryptography: A Practical Introduction to Modern Encryption*. No Starch Press, 2017.

[48] M. E. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 16, no. 6, 1978.

[49] L. E. Bassham, A. L. Rukhin, J. Soto, J. R. Nechvatal, M. E. Smid, E. B. Barker, S. D. Leigh, M. Levenson, M. Vangel, D. L. Banks, N. A. Heckert, J. F. Dray and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," National Institute of Standards and Technology, 2010.

[50] D. Mukhopadhyay, "PUFs as Promising Tools for Security in Internet of Things," *IEEE Design & Test*, vol. 33, no. 3, pp. 103–115, 2016.

[51] E. B. Barker and J. M. Kelsey, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators," National Institute of Standards and Technology, 2015.

[52] R. Ostrovsky, A. Scafuro, I. Visconti and A. Wadia, "Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions," in *Lect. Notes Comput Sc.* Vol. 7881 LNCS, 2013, pp. 702–718.

[53] U. Rührmair, "Oblivious Transfer Based on Physical Unclonable Functions," in *Trust and Trustworthy Computing*, 2010, pp. 430–440.

[54] B. Gassend, D. Clarke, M. van Dijk and S. Devadas, "Silicon physical random functions," *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS2002*, p. 148, November 2002.

[55] P. Tuyls and B. Škorić, "Strong Authentication with Physical Unclonable Functions," in *Security, Privacy, and Trust in Modern Data Management*, 2007, pp. 133–148.

[56] M. Delavar, S. Mirzakuchaki and J. Mohajeri, "PUF-based solutions for secure communication in Advanced Metering Infrastructure (AMI)," 2016.

[57] V. van der Leest, E. van der Sluis, G. J. Schrijen, P. Tuyls and H. Handschuh, "Efficient Implementation of True Random Number Generator Based on SRAM PUFs," in *Cryptography and Security: From Theory to Applications: Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, 2012, pp. 300–318.

[58] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk and S. Devadas, "AEGIS," in *Proceedings of the 17th Annual International Conference on Supercomputing - ICS '03*, 2003, p. 160.

[59] C. Hoffman, M. Cortes, D. F. Aranha and G. Araujo, "Computer security by hardware-intrinsic authentication," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015, pp. 143–152.

[60] Intrinsic ID. (2019). "Intrinsic ID," [Online]. Available: https://www.intrinsic-id.com/ (visited on 25/11/2019).

[61] D. Clarke, B. Gassend, M. V. Dijk and S. Devadas, "Authentication of integrated circuits," U.S. Patent 7840803B2, 2010.

[62] V. van der Leest, G. J. Schrijen, H. Handschuh and P. Tuyls, "Hardware Intrinsic Security from D Flip-flops," in *Proceedings of the Fifth ACM Workshop on Scalable Trusted Computing*, (Chicago, Illinois, USA), 2010, pp. 53–62.

[63] eMemory Technology Inc. (2019). "eMemory," [Online]. Available: https://www.ememory.com.tw/en-US/Products/Product?guid=19081314113656 (visited on 25/11/2019).

[64] Microsemi Corporation. (2020). "Microsemi: SmartFusion2 SoC FPGAs," [Online]. Available: https://www.microsemi.com/product-directory/soc-fpgas/1692-smartfusion2 (visited on 25/11/2019).

[65] M. Integrated. (2019). "Maxim Integrated: Security Devices," [Online]. Available: https://www.maximintegrated.com/en/products/embedded-security.html (visited on 25/11/2019).

[66] C. Tremlet and S. E. Jones, "Systems and methods for authentication based on physically unclonable functions," U.S. Patent 20170005811A1, 2017. (visited on 25/11/2019).

[67] J. Guajardo, B. Škorić, P. Tuyls, S. S. Kumar, T. Bel, A. H. M. Blom and G. J. Schrijen, "Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions," *Information Systems Frontiers*, vol. 11, no. 1, pp. 19–41, 2009.

[68] J. Lee, D. L. D. Lim, B. Gassend, G. Suh, M. V. Dijk and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," *Symposium on VLSI Circuits*, pp. 176–179, 2004.

[69] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *44th ACM/IEEE Design Automation Conference*, 2007, pp. 9–14.

[70] J. Guajardo, S. S. Kumar, G. J. Schrijen and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," *Lect. Notes Comput Sc.*, vol. 4727, pp. 63–80, 2007.

[71] Y. Su, J. Holleman and B. Otis, "A 1.6pJ/bit 96% Stable Chip-ID Generating Circuit using Process Variations," in *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*, 2007, pp. 406–611.

[72] M. Majzoobi, F. Koushanfar and M. Potkonjak, "Techniques for Design and Implementation of Secure Reconfigurable PUFs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 1, pp. 1–33, 2009.

[73] J. Delvaux, R. Peeters, D. Gu and I. Verbauwhede, "A Survey on Lightweight Entity Authentication with Strong PUFs," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–42, 2015.

[74] U. Rührmair, H. Busch and S. Katzenbeisser, "Strong PUFs: Models, Constructions, and Security Proofs," in, 2010, pp. 79–96.

[75] G. T. Becker, "The Gap Between Promise and Reality: On the Insecurity of XOR Arbiter PUFs," in *Lect. Notes Comput Sc.* Vol. 9293, 2015, pp. 535–555.

[76] U. Rührmair, J. Solter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson and S. Devadas, "PUF modeling attacks on simulated and silicon data," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 11, pp. 1876–1891, 2013.

[77] F. Ganji, S. Tajik and J. P. Seifert, "Why Attackers Win: On the Learnability of XOR Arbiter PUFs," in, 2015, pp. 22–39.

[78] M. S. Mispan, B. Halak and M. Zwolinski, "Lightweight obfuscation techniques for modeling attacks resistant PUFs," in *2017 IEEE 2nd International Verification and Security Workshop (IVSW)*, 2017, pp. 19–24.

[79] F. Dan, Y. Xu, Z. Li, J. Wen, B. Liu, S. Chen and B. Li, "A Modeling Attack Resistant R-XOR APUF Based on FPGA," in *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, 2018, pp. 577–581.

[80] B. Halak, *Physically Unclonable Functions: From Basic Design Principles to Advanced Hardware Security Applications*. Springer, 2018, 259 pp., ISBN: 978-3-319-76804-5.

[81] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen and P. Tuyls, "The Butterfly PUF protecting IP on every FPGA," in *2008 Ru, HOST*, 2008, pp. 67–70.

[82] K. Lofstrom, W. Daasch and D. Taylor, "IC identification circuit using device mismatch," in *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*, 2000, pp. 372–373.

[83] M. Kalyanaraman and M. Orshansky, "Novel strong PUF based on nonlinearity of MOSFET subthreshold operation," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 13–18.

[84] M. S. Mispan, B. Halak, Z. Chen and M. Zwolinski, "TCO-PUF: A subthreshold physical unclonable function," in *2015 11th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME)*, 2015, pp. 105–108.

[85] C. Keller, F. Gürkaynak, H. Kaeslin and N. Felber, "Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2740–2743.

[86] F. Tehranipoor, N. Karimian, K. Xiao and J. Chandy, "DRAM Based Intrinsic Physical Unclonable Functions for System Level Security," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, (Pittsburgh, Pennsylvania, USA), 2015, pp. 15–20.

[87] W. Xiong, A. Schaller, N. A. Anagnostopoulos, M. U. Saleem, S. Gabmeyer, S. Katzenbeisser and J. Szefer, "Run-Time Accessible DRAM PUFs in Commodity Devices," in *Cryptographic Hardware and Embedded Systems – CHES 2016*, 2016, pp. 432–453.

[88] B. Gassend, D. Clarke, M. van Dijk and S. Devadas, "Controlled physical random functions," in *18th Annual Computer Security Applications Conference, 2002. Proceedings.*, 2002, pp. 149–160.

[89] K. Kursawe, A. R. Sadeghi, D. Schellekens, B. Skoric and P. Tuyls, "Reconfigurable Physical Unclonable Functions - Enabling technology for tamper-resistant storage," in *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2009, pp. 22–29.

[90] S. Katzenbeisser, Ü. Kocabaş, V. van der Leest, A. R. Sadeghi, G. J. Schrijen and C. Wachsmann, "Recyclable PUFs: Logically reconfigurable PUFs," *Journal of Cryptographic Engineering*, vol. 1, no. 3, pp. 177–186, 2011.

[91] U. Rührmair, C. Jaeger and M. Algasinger, "An attack on PUF-based session key exchange and a hardware-based countermeasure: Erasable PUFs," *Lect. Notes Comput Sc.*, vol. 7035 LNCS, pp. 190–204, 2012.

[92] C. Jin, X. Xu, W. Burleson, U. Rührmair and M. van Dijk, *PLayPUF: Programmable Logically Erasable PUFs for Forward and Backward Secure Key Management*, 2015.

[93] U. Rührmair, "SIMPL systems, or: Can we design cryptographic hardware without secret key information?" In *Lect. Notes Comput Sc.*, vol. 6543 LNCS, 2011, pp. 26–45.

[94] M. Potkonjak and V. Goudar, "Public Physical Unclonable Functions," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1142–1156, 2014.

[95] S. Meguerdichian and M. Potkonjak, "Matched public PUF: Ultra low energy security platform," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, 2011, pp. 45–50.

[96] Y. Guo, T. Dee and A. Tyagi, "Barrel Shifter Physical Unclonable Function Based Encryption," *Cryptography*, vol. 2, no. 3, p. 22, 2018.

[97] 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 *Lect. Notes Comput Sc.*, vol. 7397, 2012, pp. 374–389.

[98] C. Herder, B. Fuller, M. van Dijk and S. Devadas, "Public Key Cryptosystems with Noisy Secret Keys," *IACR Cryptology ePrint Archive*, 2017.

[99] B. Gassend, M. V. Dijk, D. Clarke, E. Torlak, S. Devadas and P. Tuyls, "Controlled physical random functions and applications," *ACM Transactions on Information and System Security*, vol. 10, no. 4, pp. 1–22, 2008.

[100] N. A. Anagnostopoulos, T. Arul, M. Rosenstihl, A. Schaller, S. Gabmeyer and S. Katzenbeisser, "Attacking SRAM PUFs using very-low-temperature data remanence," *Microprocessors and Microsystems*, vol. 71, p. 102 864, 2019.

[101] U. Rührmair and M. Van Dijk, "PUFs in security protocols: Attack models and security evaluations," in *Proceedings - IEEE Symposium on Security and Privacy*, 2013, pp. 286–300.

[102] Y. Dodis, R. Ostrovsky, L. Reyzin and A. Smith, "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008.

[103] M. Hofer and C. Boehm, "An Alternative to Error Correction for SRAM-Like PUFs," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, 2010, pp. 335–350.

[104] M. S. Mispan, S. Duan, B. Halak and M. Zwolinski, "A reliable PUF in a dual function SRAM," *Integration*, vol. 68, pp. 12–21, 2019.

[105] K. Goutsos, "PUF-Based Authority Device Scheme," Newcastle University, 2019.

[106] Yubico. (2020). "Yubico YubiKey," [Online]. Available: https://www.yubico.com/ (visited on 03/07/2020).

[107] Google Inc. (2020). "Titan Security Key," [Online]. Available: https://cloud.google.com/titan-security-key (visited on 03/07/2020).

[108] Kensington. (2020). "Kensington VeriMark Fingerprint Key," [Online]. Available: https://www.kensington.com/p/products/data-protection/biometric/verimark-fingerprint-key-fido-u2f-windows-hello-designed-for-surface/ (visited on 03/07/2020).

[109] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[110] J. Delvaux, D. Gu, D. Schellekens and I. Verbauwhede, "Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 889–902, 2015.

[111] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pp. 128–133, 2011.

[112] U. Carlsen, "Cryptographic protocol flaws: Know your enemy," in *Proceedings The Computer Security Foundations Workshop VII*, pp. 192–200.

[113] B. Blanchet, *Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif*. now, 2016, ISBN: 978-1-68083-207-5. (visited on 30/08/2020).

[114] K. Moriarty, B. Kaliski, J. Jonsson and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2," no. 8017, pp. 1–78, 2016.

[115] D. Johnson, A. Menezes and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)," *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[116] R. Sinha, H. K. Srivastava and S. Gupta, "Performance Based Comparison Study of RSA and Elliptic Curve Cryptography," *International Journal of Scientific Engineering*, vol. 4, no. 5, pp. 720–725, 2013.

[117] A. Bogdanov, M. Knežević, G. Leander, D. Toz, K. Varici and I. Verbauwhede, "Spongent: A lightweight hash function," in *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011.

[118] D. Eastlake 3rd and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)," no. 6234, pp. 1–127, 2011.

[119] M. Saarinen and J. Aumasson, "The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)," no. 7693, pp. 1–30, 2015.

[120] *FIPS 202, SHA-3 Standard: Permutation-Based Hash And Extendable-Output Functions*, 2015.

[121] E. Barker, "NIST Recommendation for Key Management Part 1: General," National Institute of Standards and Technology, 2016.

[122] B. Barak and S. Halevi, "A model and architecture for pseudo-random generation with applications to /dev/random," p. 203, 2005.

[123] S. Devadas, E. Suh, S. Paral, R. Sowell, T. Ziola and V. Khandelwal, "Design and Implementation of PUF-Based "Unclonable" RFID ICs for Anti-Counterfeiting and Security Applications," in *2008 IEEE International Conference on RFID*, 2008, pp. 58–64.

[124] M. Rostami, M. Majzoobi, F. Koushanfar, D. S. Wallach and S. Devadas, "Robust and Reverse-Engineering Resilient PUF Authentication and Key-Exchange by Substring Matching," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 1, pp. 37–49, 2014.

[125] C. Huth, J. Zibuschka, P. Duplys and T. Güneysu, "Securing systems on the Internet of Things via physical properties of devices and communications," in *SysCon 2015*, 2015, pp. 8–13.

[126] M. Majzoobi, M. Rostami, F. Koushanfar, D. S. Wallach and S. Devadas, "Slender PUF Protocol: A Lightweight, Robust, and Secure Authentication by Substring Matching," in *2012 IEEE Symposium on Security and Privacy Workshops*, 2012, pp. 33–44.

[127] Y. Yilmaz, S. R. Gunn and B. Halak, "Lightweight PUF-Based Authentication Protocol for IoT Devices," in *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, 2018, pp. 38–43.

[128] M. Barbareschi, A. De Benedictis and N. Mazzocca, "A PUF-based hardware mutual authentication protocol," *Journal of Parallel and Distributed Computing*, vol. 119, pp. 107–120, 2018.

[129] M. N. Aman, K. C. Chua and B. Sikdar, "Physical Unclonable Functions for IoT Security," in *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security - IoTPTS '16*, 2016, pp. 10–13.

[130] M. H. Mahalat, S. Saha, A. Mondal and B. Sen, "A PUF based Light Weight Protocol for Secure WiFi Authentication of IoT devices," in *2018 8th International Symposium on Embedded Computing and System Design (ISED)*, 2018, pp. 183–187.

[131] Ł. Krzywiecki, "Anonymous Authentication Scheme Based on PUF," in *LNCS*, 2016, pp. 359–372.

[132] U. Rührmair, "SIMPL Systems as a Keyless Cryptographic and Security Primitive," in *Lect. Notes Comput Sc.* Vol. 6805 LNCS, 2012, pp. 329–354.

[133] D. Ranasinghe, D. Engels and P. Cole, "Security and privacy: Modest proposals for low-cost RFID systems," 2004.

[134] Q. Chen, U. Rührmair, S. Narayana, U. Sharif and U. Schlichtmann, "MWA Skew SRAM Based SIMPL Systems for Public-Key Physical Cryptography," in, 2015, pp. 268–282.

[135] T. Xu, J. B. Wendt and M. Potkonjak, "Matched Digital PUFs for Low Power Security in Implantable Medical Devices," in *2014 IEEE International Conference on Healthcare Informatics*, 2014, pp. 33–38.

[136] K. Cohn-Gordon, C. Cremers and L. Garratt, *On Post-Compromise Security*, 2016.

[137] B. Poettering and P. Rösler, "Towards Bidirectional Ratcheted Key Exchange," in, 2018, pp. 3–32.

[138] C. P. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology—CRYPTO'89 Proceedings*, 1990, pp. 239–252.

[139] K. Bhargavan, B. Blanchet and N. Kobeissi, "Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 483–502.

[140] R. Küsters and T. Truderung, "Reducing Protocol Analysis with XOR to the XOR-Free Case in the Horn Theory Based Approach," *Journal of Automated Reasoning*, vol. 46, no. 3, pp. 325–352, 2011.

[141] M. Backes, M. Maffei and D. Unruh, "Zero-Knowledge in the Applied Pi-calculus and Automated Verification of the Direct Anonymous Attestation Protocol," in *2008 IEEE Symposium on Security and Privacy (Sp 2008)*, 2008, pp. 202–215.

[142] W. Wang, J. Liu, Y. Qin and D. Feng, "Formal Analysis of a TTP-Free Blacklistable Anonymous Credentials System," in *Information and Communications Security*, 2018, pp. 3–16.

[143] M. Bhargava and K. Mai, "An efficient reliable PUF-based cryptographic key generator in 65nm CMOS," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, 2014.

[144] M. Igier and S. Vaudenay, "Distance Bounding Based on PUF," in *Cryptology and Network Security*, 2016, pp. 701–710.

[145] M. Bellare, R. Canetti and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *Advances in Cryptology — CRYPTO '96*, 1996, pp. 1–15.

[146] M. Bellare, "New Proofs for NMAC and HMAC: Security Without Collision-Resistance," in *Advances in Cryptology - CRYPTO 2006*, 2006, pp. 602–619.

[147] R. Natti and S. Rangu, "Implementations of Secure Reconfigurable Cryptoprocessor a Survey," in *Information Systems Design and Intelligent Applications*, 2016, pp. 11–19.

[148] I. H. Hazmi, F. Zhou, F. Gebali and T. F. Al-Somani, "Review of Elliptic Curve Processor architectures," in *2015 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2015, pp. 192–200.

[149] M. Varchola, T. Guneysu and O. Mischke, "MicroECC: A Lightweight Reconfigurable Elliptic Curve Crypto-processor," in *2011 International Conference on Reconfigurable Computing and FPGAs*, 2011, pp. 204–210.

[150] NXP Semiconductors N.V. (2019). "Crypto Coprocessor," [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/legacy-mcu-mpus/crypto-coprocessors/crypto-coprocessor:C29x (visited on 19/09/2019).

[151] A. Bogdanov, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw and Y. Seurin, "Hash Functions and RFID Tags: Mind the Gap," in *Cryptographic Hardware and Embedded Systems – CHES 2008*, 2008, pp. 283–299.

[152] Anthony Van Herrewege, *Lightweight PUF-Based Key and Random Number Generation*. 2015, ISBN: 978-94-6018-947-0.

[153] 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, pp. 1198–1210, 2009.

[154] E. Leobandung, "SRAM as Random Number Generator," U.S. Patent 20190182054A1, 2017.

[155] M. Cortez, A. Dargar, S. Hamdioui and G. J. Schrijen, "Modeling SRAM start-up behavior for physical unclonable functions," in *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2012, pp. 1–6.

[156] D. Eastlake 3rd, J. Schiller and S. Crocker, "Randomness Requirements for Security," no. 4086, 2005.

[157] M. Grujić, V. Rožić, D. Johnston, J. Kelsey and I. Verbauwhede, "INVITED: Design Principles for True Random Number Generators for Security Applications," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–3.

[158] A. Althoff and R. Kastner, "An Architecture for Learning Stream Distributions with Application to RNG Testing," in *Proceedings of the 54th Annual Design Automation Conference 2017*, (Austin, TX, USA), 2017, 15:1–15:6.

[159] V. B. Suresh, D. Antonioli and W. P. Burleson, "On-chip lightweight implementation of reduced NIST randomness test suite," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 93–98.

[160] B. Yang, V. Rožić, N. Mentens, W. Dehaene and I. Verbauwhede, "TOTAL: TRNG On-the-fly Testing for Attack Detection Using Lightweight Hardware," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, 2016, pp. 127–132.

[161] S. U. Hussain, M. Majzoobi and F. Koushanfar, "A Built-in-Self-Test Scheme for Online Evaluation of Physical Unclonable Functions and True Random Number Generators," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 1, pp. 2–16, 2016.

[162] Y. Yu, E. Dubrova, M. Näslund and S. Tao, "On Designing PUF-Based TRNGs with Known Answer Tests," in *2018 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2018, pp. 1–6.

[163] J. Kelsey, B. Schneier and N. Ferguson, "Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator," in *Selected Areas in Cryptography*, 2000, pp. 13–33.

[164] Y. Kai, Z. Xuecheng, Y. Guoyi and W. Weixu, "Security strategy of powered-off SRAM for resisting physical attack to data remanence," *Journal of Semiconductors*, vol. 30, no. 9, p. 095 010, 2009.

[165] A. Wild and T. Guneysu, "Enabling SRAM-PUFs on Xilinx FPGAs," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.

[166] N. A. Anagnostopoulos, T. Arul, M. Rosenstihl, A. Schaller, S. Gabmeyer and S. Katzenbeisser, "Low-Temperature Data Remanence Attacks Against Intrinsic SRAM PUFs," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 581–585.

[167] L. Zheng, D. Han, Z. Liu, C. Ma, L. Zhang and C. Tang, "A Low Overhead Error Correction Algorithm Using Random Permutation for SRAM PUFs," in, 2019, pp. 475–493.

[168] 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, 2012, pp. 268–282.

[169] M. D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[170] D. Merli, F. Stumpf and G. Sigl, "Protecting PUF Error Correction by Codeword Masking," *Eprint.Iacr.Org*, pp. 1–16, 2013.

[171] R. Maes, A. Van Herrewege and I. Verbauwhede, "PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator," in *CHES 2012 Lecture Notes in Computer Science*, vol. 7428, 2012, pp. 302–319.

[172] C. Bösch, J. Guajardo, A. R. Sadeghi, J. Shokrollahi and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems – CHES 2008*, 2008, pp. 181–197.

[173] M. S. Mispan, B. Halak and M. Zwolinski, "NBTI aging evaluation of PUF-based differential architectures," in *2016 IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016, pp. 103–108.

[174] S. Rangan, N. Mielke and E. C. C. Yeh, "Universal recovery behavior of negative bias temperature instability [PMOSFETs]," in *IEEE International Electron Devices Meeting 2003*, 2003, pp. 14.3.1–14.3.4.

[175] S. Kleber, F. Unterstein, M. Matousek, F. Kargl, F. Slomka and M. Hiller, "Secure execution architecture based on PUF-driven instruction level code encryption," 2015.

[176] 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 *Proceedings - IEEE International Symposium on Circuits and Systems*, 2011.

[177] R. C. Jaeger, *Microelectronic Circuit Design*. McGraw-Hill, 1997, ISBN: 0-07-114386-6.

[178] D. Karakoyunlu and B. Sunar, "Differential template attacks on PUF enabled cryptographic devices," in *2010 IEEE International Workshop on Information Forensics and Security*, 2010, pp. 1–6.

[179] D. Nedospasov, J.-P. Seifert, C. Helfmeier and C. Boit, "Invasive PUF Analysis," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2013, pp. 30–38.

[180] A. Roelke and M. R. Stan, "Attacking an SRAM-Based PUF through Wearout," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016, pp. 206–211.

[181] S. Mahapatra and M. Alam, "A predictive reliability model for PMOS bias temperature degradation," in *Digest. International Electron Devices Meeting,*, 2002, pp. 505–508.

[182] M. Denais, V. Huard, C. Parthasarathy, G. Ribes, F. Perrier, N. Revil and A. Bravaix, "Interface trap generation and hole trapping under NBTI and PBTI in advanced CMOS technology with a 2-nm gate oxide," *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 4, pp. 715–722, 2004.

[183] R. Maes and V. van der Leest, "Countering the effects of silicon aging on SRAM PUFs," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 148–153.

[184] M. S. Mispan, M. Zwolinski and B. Halak, "Ageing Mitigation Techniques for SRAM Memories," in *Ageing of Integrated Circuits: Causes, Effects and Mitigation Techniques*, 2020, pp. 91–111.

[185] 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 *Proceedings of the 2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013*, 2013.

[186] H. Qin, C. Yu, D. Markovic, A. Vladimirescu and J. Rabaey, "SRAM leakage suppression by minimizing standby supply voltage," in *International Symposium on Signals, Circuits and Systems. Proceedings, SCS 2003. (Cat. No.03EX720)*, 2004, pp. 55–60.

[187] Python Software Foundation. (2019). "Python.org," [Online]. Available: https://www.python.org/ (visited on 18/03/2019).

[188] PyCryptodome. (2019). "PyCryptodome," [Online]. Available: https://www.pycryptodome.org/ (visited on 18/03/2019).

[189] Red Hat Inc. (2019). "KVM," [Online]. Available: https://www.linux-kvm.org/ (visited on 18/03/2019).

[190] S. K. Mathew, S. K. Satpathy, M. A. Anders, H. Kaul, S. K. Hsu, A. Agarwal, G. K. Chen, R. J. Parker, R. K. Krishnamurthy and V. De, "16.2 A 0.19pJ/b PVT-variation-tolerant hybrid physically unclonable function circuit for 100% stable secure key generation in 22nm CMOS," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 278–279.

[191] "IT Security techniques — Hash-functions — Part 3: Dedicated hash-functions," International Organization for Standardization, Standard, 2018.

[192] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici and I. Verbauwhede, "SPONGENT: The Design Space of Lightweight Cryptographic Hashing," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 2041–2053, 2013.

[193] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin and C. Vikkelsoe, "PRESENT: An Ultra-Lightweight Block Cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007*, 2007, pp. 450–466.

[194] J. Guo, T. Peyrin and A. Poschmann, "The PHOTON Family of Lightweight Hash Functions," in *Advances in Cryptology – CRYPTO 2011*, 2011, pp. 222–239.

[195] J. P. Aumasson, L. Henzen, W. Meier and M. Naya-Plasencia, "Quark: A Lightweight Hash," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, 2010, pp. 1–15.

[196] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "Sponge functions," Citeseer.

[197] M. Bellare and P. Rogaway, "Minimizing the use of random oracles in authenticated encryption schemes," in *Information and Communications Security*, 1997, pp. 1–16.

[198] F. Hao (Ed.), "Schnorr Non-interactive Zero-Knowledge Proof," no. 8235, pp. 1–13, 2017.

[199] S. Turner, D. Brown, K. Yiu, R. Housley and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information," no. 5480, pp. 1–20, 2009.

[200] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003, ISBN: 978-0-521-64298-9.

[201] J. Daugman, "The importance of being random: Statistical principles of iris recognition," *Pattern recognition*, vol. 36, no. 2, pp. 279–291, 2003.

[202] Wireshark Foundation. (2019). "Tshark," [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html (visited on 18/03/2019).

[203] Wireshark Foundation. (2019). "Wireshark," [Online]. Available: https://www.wireshark.org/ (visited on 26/03/2019).

[204] M. Kim, J. Ryou and S. Jun, "Efficient Hardware Architecture of SHA-256 Algorithm for Trusted Mobile Computing," in *Information Security and Cryptology*, 2009, pp. 240–252.

[205] G. Bertoni, L. Breveglieri and M. Venturi, "ECC Hardware Coprocessors for 8-bit Systems and Power Consumption Considerations," in *Third International Conference on Information Technology: New Generations (ITNG'06)*, 2006, pp. 573–574.

[206] G. Gaubatz, J. P. Kaps, E. Ozturk and B. Sunar, "State of the art in ultra-low power public key cryptography for wireless sensor networks," in *Third IEEE International Conference on Pervasive Computing and Communications Workshops*, 2005, pp. 146–150.

[207] P. Reviriego, C. Argyrides and J. A. Maestro, "Efficient error detection in Double Error Correction BCH codes for memory applications," *Microelectronics Reliability*, vol. 52, no. 7, pp. 1528–1530, 2012.

[208] N. Ahmad and R. Hasan, "A new design of XOR-XNOR gates for low power application," in *2011 International Conference on Electronic Devices, Systems and Applications (ICEDSA)*, 2011, pp. 45–49.

[209] M. M. Fouda, Z. M. Fadlullah, N. Kato, R. Lu and X. S. Shen, "A Lightweight Message Authentication Scheme for Smart Grid Communications," *IEEE Transactions on Smart Grid*, vol. 2, no. 4, pp. 675–685, 2011.

[210] M. Jelasity, "Gossip," in *Self-Organising Software: From Natural to Artificial Adaptation*, 2011, pp. 139–162.

[211] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Advances in Cryptology — CRYPTO '97*, 1997, pp. 410–424.

[212] K. Xing, F. Liu, X. Cheng and D. H. Du, "Real-Time Detection of Clone Attacks in Wireless Sensor Networks," in *2008 The 28th International Conference on Distributed Computing Systems*, 2008, pp. 3–10.

[213] M. Signorini, "Towards an internet of trust: Issues and solutions for identification and authentication in the internet of things," *TDX (Tesis Doctorals en Xarxa)*, 2015.