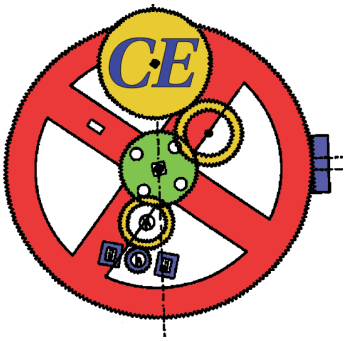


**Computer Engineering**
Mekelweg 4,
2628 CD Delft
The Netherlands
**http://ce.et.tudelft.nl/**

**2017**

# MSc THESIS

# Design & development of public-key based authentication architecture for IoT devices using PUF

**Haji Akhundov**

**CE-MS-2017-02**

## Abstract

Secure communication has been paramount throughout history. Although in the early ages it was mainly found in niche applications such as the military and royal society, today it is an inevitable part of our daily lives. The recent rapid proliferation of Internet of Things (IoT), a diverse set of devices that are connected to the Internet, imposes new challenges in protecting our privacy and security in our daily connected lives. In most cases, one-size-fits-all security solutions are inefficient; therefore, we need high-quality application-specific solutions. This thesis designs, develops and evaluates a secure communication architecture based on Static Random-Access Memory (SRAM) Physical Unclonable Function (PUF) technology and Elliptic Curve Cryptography (ECC) for IoT devices in collaboration with Intrinsic ID, a world leading PUF technology company. SRAM PUF is a popular emerging hardware intrinsic security primitive: its start-up values (SUV) can be used to uniquely identify and authenticate silicon, due to the hard to clone, inherent and device unique deep-submicron process variations. ECC is an approach to public-key cryptography, which can be used to establish shared secret keys among parties; it has been gaining popularity among *lightweight* IoT devices because achieving equivalent security level requires significantly smaller operands when compared to other approaches. Our solution consists of two systematic steps: *(1)* development of a cryptographic protocol which utilizes PUF-derived key as the root-of-trust, while keeping the area constraint into account, and *(2)* design and development of a modular hardware architecture that supports the protocol. We propose four protocol variants with trade-offs related to security versus implementation requirements. Further, we verify and prototype one protocol variant on the Xilinx Zynq-7000 APSoC device, and analyze its practicality as well as its feasibility. The prototype offers interesting insights and lays a solid foundation for future research.

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

# Design & development of public-key based authentication architecture for IoT devices using PUF

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Haji Akhundov
born in Baku, Azerbaijan

Computer Engineering
Department of Quantum Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

# Design & development of public-key based authentication architecture for IoT devices using PUF

by Haji Akhundov

**Abstract**

Secure communication has been paramount throughout history. Although in the early ages it was mainly found in niche applications such as the military and royal society, today it is an inevitable part of our daily lives. The recent rapid proliferation of IoT, a diverse set of devices that are connected to the Internet, imposes new challenges in protecting our privacy and security in our daily connected lives. In most cases, one-size-fits-all security solutions are inefficient; therefore, we need high-quality application-specific solutions. This thesis designs, develops and evaluates a secure communication architecture based on SRAM PUF technology and ECC for IoT devices in collaboration with Intrinsic ID, a world leading PUF technology company. SRAM PUF is a popular emerging hardware intrinsic security primitive: its start-up values (SUV) can be used to uniquely identify and authenticate silicon, due to the hard to clone, inherent and device unique deep-submicron process variations. ECC is an approach to public-key cryptography, which can be used to establish shared secret keys among parties; it has been gaining popularity among *lightweight* IoT devices because achieving equivalent security level requires significantly smaller operands when compared to other approaches. Our solution consists of two systematic steps: *(1)* development of a cryptographic protocol which utilizes PUF-derived key as the root-of-trust, while keeping the area constraint into account, and *(2)* design and development of a modular hardware architecture that supports the protocol. We propose four protocol variants with trade-offs related to security versus implementation requirements. Further, we verify and prototype one protocol variant on the Xilinx Zynq-7000 APSoC device, and analyze its practicality as well as its feasibility. The prototype offers interesting insights and lays a solid foundation for future research.

**Laboratory** : Computer Engineering
**Codenumber** : CE-MS-2017-02
**Committee Members** :

| | |
|---|---|
| **Advisor:** | Prof. Dr. Ir. Said Hamdioui, CE, TU Delft |
| **Chairperson:** | Prof. Dr. Ir. Said Hamdioui, CE, TU Delft |
| **Member:** | Dr. Ir. Zekeriya Erkin, CYS, TU Delft |
| **Member:** | Ir. Erik van der Sluis, Intrinsic ID |

*Dedicated to my family and friends*

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**PKC** Public-key Cryptography

**PUF** Physical Unclonable Function

**IoT** Internet of Things

**TTP** Trusted Third Party

**NVM** Non-volatile Memory

**ECC** Elliptic Curve Cryptography

**SRAM** Static Random-Access Memory

**AC** Activation Code

**HD** Helper Data

**IFC** Integer Factorization Cryptography

**FFC** Finite Field Cryptography

**ALU** Arithmetic Logic Unit

**CR** Challenge-Response

**ECDH** Elliptic Curve Diffie-Hellman

**RTL** Register Transfer Level

**IC** Integrated Circuit

**MITM** Man-In-The-Middle

**DH** Diffie-Hellman

**DHKE** Diffie-Hellman Key Exchange

**CMOS** Complementary Metal Oxide Semiconductor

**ECDLP** Elliptic Curve Discrete Logarithm Problem

**DLP** Discrete Log Problem

**GDLP** Generalized Discrete Logarithm Problem

**ECDLP** Elliptic Curve Discrete Logarithm Problem

**SoC** System on Chip

**FPGA** Field Programmable Gate Array

**AXI** Advanced eXtensible Interface

**ASIC** Application Specific Integrated Circuit

**PS** Processing System

**PL** Programmable Logic

**PCB** Printed Circuit Board

**IETF** Internet Engineering Task Force

**RFC** Request for Comments

**AMBA** Advanced Microcontroller Bus Architecture

**PKI** Public Key Infrastructure

**ASIP** Application-specific Instruction Set Processor

**ISA** Instruction Set Architecture

**RNG** Random Number Generator

**GPP** General Purpose Processor

**AES** Advanced Encryption Standard

**ECDSA** Elliptic Curve Digital Signature Algorithm

**APSoC** All Programmable System on Chip

**ROM** Read-Only Memory

# Acknowledgements

Dear reader,

This project would not have been possible without the support of many people. Therefore, I would like to take a moment to thank everyone who has helped me along the way! Thank you all! Please forgive me if I forget someone!

Working on this project for several intensive months has left a long-lasting, positive impact on me. I mark this period as a period of growth and transition. It was a period of intense learning, in all different aspects: scientific, academic, personal, social, etc. It has helped me *become the better version of myself*, which is the motto I live by.

Foremost, I would like to express my sincere gratitude to my advisor Said Hamdioui. His guidance helped me in all the time of research and writing of this thesis. He has taught me valuable lessons and is a great mentor. Furthermore, I thank Zekeriya Erkin, for the support and encouragement in the course of this project. I would like to thank Rene van Leuken who is always willing to help and open for interesting discussions. I would like to thank Zaid Al-Ars who has encouraged me to come and do my masters at TU Delft, and is always helpful. I would like to thank all members of the CYS and CE groups at TU Delft who have helped!

My sincere thanks go to Intrinsic ID, for giving me this amazing opportunity, which allowed me to work in collaboration with the industry, and therefore get a taste of both worlds. I would like to thank all my colleagues there for being nice to me, always supportive and always willing to help me. I thank Geert-Jan Schrijen for arranging this project; Erik van der Sluis for his continuous supervision, motivation and guidance; Roel Maes for his immense knowledge and thought-provoking comments on my work. Big thanks goes to Sven Goossens who has spent a considerable amount of his time for reading the entire thesis and providing me great feedback.

Thanks to Michael Hutter and Peter Schwabe for responding to my e-mails and providing me with more information about the NaCl core.

I would also like to thank my fellow friends from the MSc lab: we had a lot of fun times together! I thank all my friends in the Netherlands and around the globe for simply being there and always being supportive!

Last but not the least, I would like to thank my family: my parents Rufat and Ziba, for supporting me throughout my life, for their wise counsel, and for always being there for me. I would like to thank my little sister Jamilya for being sweet. Your birth has changed our lives. I thank my grandparents from whom I inherited the love for science and academia.

Thank you all very much!

Haji Akhundov
Delft, The Netherlands
January 27, 2017

# 1 Introduction

*This chapter motivates the work done in this thesis, introduces relevant background information about security in general and cryptography in particular, stresses the main contributions of the work and provides the outline of the thesis. Section 1.1 provides some interesting historical background on secure communication and shows why the field is important today more than ever. Section 1.2 elaborates on the motivation of application driven implementations of secure hardware architectures and authentic communication, which is the topic of this thesis. The major contributions of this thesis are highlighted in Section 1.3, and the chapter concludes by describing the outline for this thesis in Section 1.4.*

## 1.1 Secure Communication

### 1.1.1 The Past

The art of secret writing dates back to ancient times, where keeping secrets far from the prying eyes of the enemies was a matter of *winning or losing a battle, between life and death*. An excellent source of inspiration filled with examples of ancient secret writing is provided in The Code Book [1]. For example, it illustrates how Greeks and Persians of fifth century B.C. used *stenography* to hide secret messages. One such technique was by writing secret messages on a shaved head of a human, letting the hair grow, therefore hiding the message. The book gives another example of an interesting historical event that dates back to the $16^{th}$ century where Mary, Queen of Scots, was trialled and executed for treason when her secret messages were captured, and her secret codes that contained the assassination plan of Queen Elizabeth were easily broken. A painting by Hrana Janto, shown in Figure 1.1 illustrates a scytale transposition cipher used by the ancient Greek and Spartan soldiers during military campaigns [2]. In this cypher, a parchment is wrapped around a rod with a certain diameter $d$, after which a secret message is written on. The diameter of the rod on the recipients side must therefore also equal $d$, so that message can be correctly read. A more recent example is the second world war (WWII). Shortly after WWI, the now famous Enigma machine was developed, which later was the technology used by the Nazi military during WWII for obfuscating their secret strategic communications. Alan Turing was in charge of breaking the intercepted secret messages at the British WWII codebreaking station, Bletchley Park [3]. He invented some devices and techniques that have been able to break the Enigma machine and hence make sense of the enemy's secret communication, which contributed to the victory of the Allies. The Enigma was the state-of-the-art machine used for secure communication. Many believe that without the critical role of Alan Turing, the war could have could have lasted longer or even lost by the Allies. Ever since then, methods for secure communication

Figure 1.1: A Spartan soldier using the scytale transposition cipher. *Spartan Cryptography, by Hrana Janto*

has significantly advanced; but before we talk about that, let us get a glimpse of what is called *cryptography.*

Cryptography, derived from the Greek word *kryptós* & *gráphein*, meaning *hidden* and *writing* respectively, is the art and science of secret writing [1]. In cryptography, *encryption* is the process of making a message unintelligible, therefore hiding its meaning; the opposite is *decryption. Encryption* works as follows: given a *plaintext* message $m$, and a *key* $K$ as inputs to a certain *cipher* or an *algorithm* $f$, the resulting output of the encryption procedure is a *ciphertext* message $c$. The encryption process can be written as $c = f_K(m)$, where $f_K$ is a cipher $f$ that uses key $K$. *Decryption*, on the other hand, is where the ciphertext is transformed back into the original plaintext message. This is done by using an appropriate decryption algorithm $D_K = f_K^{-1}$ and the same key $K$, i.e. $m = D_K(c)$.

Throughout history, different cryptographic algorithms and cyphers $f$ have been developed. In the old times, cryptography was a tool used by a niche market, such as the military or the royalty, as seen from the earlier examples. It often involved simply manipulating the characters and relied on the secrecy of the cryptosystem itself. These are common characteristics of *classic cryptography.* With the advances in computing, we have access to more and more computation power, facilitating the attacks on these algorithms. One way of circumventing this is by making algorithms with longer keys. Consequently increasing the key space became a common practice for a while. The level of security is often measured in terms of the key size in binary representation. Note that, cryptography can provide the means of obfuscating the messages. Yet there are plenty of other security issues that need to be addressed in modern computer systems. The field of computer security, or *cyber security* deals with protecting computer systems, both hardware and software, networks and infrastructure.

### 1.1.2 The Present

Today, the majority of us depend on cryptography on a daily basis without consciously realising it. We live in a highly connected network world and heavily rely on its services in our day to day lives. Bills are paid on-line, and money transfers are easily accomplished with a couple of clicks on a smartphone. Sensitive and important documents such as corporate business plans are sent over these networks. Video calls and instant messaging with friends, family and business partners over the Internet has become commonplace. With the development of smart grids[1], electricity and water meters will be collecting our detailed usage statistics for various reasons e.g. load balancing, managing, and billing. Common house appliances, and other systems, are being connected to the Internet, to enable smart homes[2]. These are just a few examples of some of the critical dependencies that we have on the Internet in the $21^{st}$ century.

Obfuscating the messages that travel through the Internet that traverse various physical channels (networks) spanning the globe has become paramount. This need has fostered a lot of research in the field of cyber-security, and mainly *modern cryptography*. Modern cryptography is different from the classic cryptography because of several reasons:

- It is no longer a manipulation of traditional characters, but rather some operations on binary bit sequences; which is a natural consequence of the digital revolution.
- Another significant difference is due to Kerckhoffs' desiderata [4]; it states that everything about the secure system should be considered as public knowledge except the key. This is the opposite of the *security by obscurity* paradigm, which was often used in the past.
- Furthermore, maintaining privacy and confidentiality were once the central focus of cryptography; however, today this is only a subset of the versatile operations cryptography could equip us with. For example, *authentication* is yet another mechanism that plays a crucial role in the security of our highly connected network infrastructure. Authentication is a process where an entity's (user, device etc.) identity is verified before starting any secure communication.

Fortunately, modern cryptography has the necessary tools to achieve that. Nevertheless, the modern life and the fast pace of technological advancement introduce new challenges, which is the topic of the next section.

### 1.1.3 The Challenges

Not too long ago the Internet comprised of a small set of devices such as simple servers and personal computers. Today the Internet is a big global network hosting a diverse set of devices, commonly referred to as *Internet of Things (IoT)*, which require the use of sophisticated cryptographic protocols, primitives, and beyond. Although there is no concrete definition of IoT, the 'things' could refer to anything connected to the Internet

[1] http://www.intel.com/content/www/us/en/energy/iot-smart-grid-paper.html
[2] https://www.cnet.com/topics/smart-home/best-smart-home-devices/

such as small sensors and/or actuators, mobile devices, vehicles and house appliances such as toasters or even refrigerators! In one of the essays by Bruce Schneier[3], he describes a network of IoT devices as a giant 'robot' spanning the globe. This massive machine which is comprised of various sensors and actuators, connected to each other on a network, where all kinds of computations are done in places such as the cloud, is also commonly referred to as a cyber-physical system. According to an info-graphic provided by Intel[4], the number of IoT devices by 2020 would reach 200 billion. Such rapid development of this segment encourages us to tackle the challenges that are either directly or indirectly caused by it. At least two major challenges have to be carefully worked out when it comes to IoT era: (1) keep up with *privacy and security requirements* to ensure safe Internet; (2) deal with the *diverse nature of IoT*, which makes it hard, or even inefficient to use one-size-fits-all security solutions. They are discussed next.

Firstly, with the rapid proliferation of IoT devices, it becomes a challenging task to protect our privacy and security. An attacker could take control, steal information or even disrupt services[5]. One of the reasons there are problems in the security of the Internet is the fact that security was not the primary requirement when the Internet was first developed and later released for public mainstream use. Originally it was intended for small networks in relatively trusted infrastructures. These days the Internet is no longer a trusted or reliable infrastructure. The global addressing nature makes everyone on the Internet virtually your neighbour, imposing more threats. Imagine you as an owner of a smart home system, which allows you to remotely control the heating system, lighting system, entertainment systems and even access control systems, etc. as shown in Figure 1.2.[6] Clearly, no one would want an adversary to take over the control, switch off your lights and unlock your front door. And if that was only possible by physically being there, in this new paradigm, it is possible to attack from the antipodes. Another example is the smartphone we carry with us at all times. It is not unusual for an average device like that to host a dozen of different sensors such as GPS, accelerometer, fingerprint sensor, barometer, magnetometer, light sensor, heart-rate sensors and of course the front and back cameras and the microphone. Many applications depend on these sensors and often upload them to a cloud infrastructure for some purposes. Understandably, these devices potentially carry sensitive information that we would like to keep *private* and not share with others. For this reason, the importance of privacy and security is evident here. It is also the concerns and the awareness of the general public about their privacy, governmental policies, etc. that foster the developments and the implementation of security in these devices.

Secondly, the challenge is the diversity of IoT devices. It implies that we ought to provide different solutions for the different challenges associated with individual IoT devices. The reason is simple: It would be either inefficient or intractable to have a universal security solution/architecture for all the devices out there. e.g. an IoT device can be 'big' and have no area or power restrictions, or it can be a 'small' wearable,

[3]https://www.schneier.com/blog/archives/2016/02/the_internet_of_1.html

[4]http://intel.ly/1KXhnc2

[5]http://spectrum.ieee.org/telecom/security/how-to-build-a-safer-internet-of-things

[6]This illustration has been taken from an article "Why IoT Security Is So Critical" in TechCrunch. Link: https://techcrunch.com/2015/10/24/why-iot-security-is-so-critical/

Figure 1.2: A smart-home system, where multiple subsystems are connected together.

categorised as *constrained devices*.

In this thesis we are concerned with the constrained devices. Naturally, they have more stringent requirements. They could be limited in their computing resources or restricted in the power resources; or simply have a strict silicon area requirements such as those on RFID cards. Because silicon area footprint directly translates to cost, there is the desire to keep the silicon area footprint as minimal as possible. Reducing cost stimulates the design of cryptographic algorithms that could be implemented with these constraints in mind, without sacrificing for security. *Cryptography engineering* is the discipline of applying and implementing the known algorithms. Naturally, the algorithms could be implemented both in hardware and/or in software. Depending on the use case and the end application in consideration, one implementation would render to be more appropriate than the other. It is often a matter of trade-offs; e.g. in certain cases, the speed or the throughput could have the utmost importance with the 'unlimited' computational power or hardware resources. On the other end of the spectrum we might have very limited computational resources and low power/energy consumption requirements to perform a similar task.

To sum all, with the growth of IoT devices, and their pervasive effect on us in our modern lives, keeping up with the privacy and security requirements is a challenge. The diversity of these devices brings another set of challenges that stimulate researchers and developers to investigate individual use cases and propose specialised solutions. Another key point is that these 'things' further facilitate the emergence of other technologies such as the smart grids, smart cities, etc., that impose even more challenges. The security

of these systems is paramount. Maliciously disabling a smart grid, and hence power downing an entire city or even a country, or interfering with the traffic lights cause disturbance and possible economic loss. Systems must work for us and not be used against us as a weapon by those who wish to do harm. Systems we design and build must not only be reliable, and resilient to failures, but also secure and preserve the privacy of the users.

## 1.2 Need of High Quality and Efficient Solutions

In the previous section, we argued that one of the challenges that we face is the diverse nature of devices on the Internet. The devices can be big or small; could be residing in a server room, or can be simply constrained wearables powered by a battery, etc. This implies different requirements for every device. Other factors such as policies and laws, or even business models could be influencing the required solution with respect to security and privacy. Prominently, the end application, or the *use case*, is what decides about overall requirement, hence the design specification. To identify, clarify, and organise system requirements, we will have to examine a concrete use case. The next section shows the industrial relevant use case that will be investigated in this thesis.

### 1.2.1 Use Case

The use case under consideration, as we will see in details later, is intended for a more secure application, as compared to a modern conventional consumer IoT device application. Nevertheless, with the adoption of certain technologies described later in this thesis, the solution could lend itself to ordinary consumer devices as well in the future.

The solution we ought to design and develop, in the most general sense, must enable secure communication between *lightweight*, resource-constrained IoT devices and a resource-rich party such as a server as shown in Figure 1.3. Assume that the green boxes in the field represent legitimate devices, performing their dedicated function in an *untrusted field*. The reason it is untrusted is because we have very little, or no control over the field at all. Therefore, the two fundamental assumptions about such field are that:

- The communication medium could be wiretapped by an adversary eavesdropper. Regardless of the type of communication (wired, wireless, optical etc.), there are methods to sniff or 'read' such communication e.g. communication between the 'green' devices and the server.

- Furthermore, it is assumed that an adversary is capable of tampering with the data in transit, as well as injecting new data. Therefore, malicious devices can potentially be placed in the field (red box in Figure 1.3), that would pretend to be legitimate, and spoof the sent data and hence possibly disrupt the data on the server side.

Therefore, the need of *authentic* and *secure communication* is evident here. If such system is compromised, the damage could take different forms and could be less or more

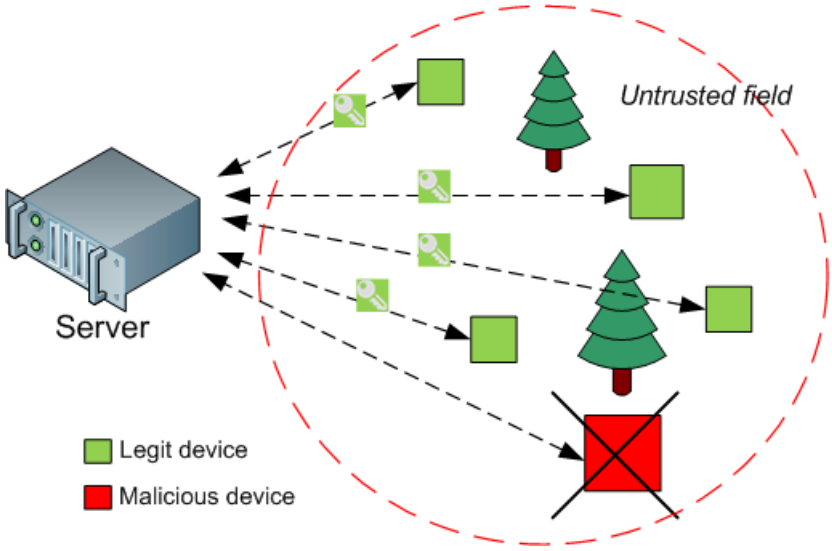



Figure 1.3: Targeted Applications

devastating depending on the exact end-user application. Nevertheless, it is imperative that appropriate security measures are taken, and therefore it is in our interest to achieve maximal security.

To alleviate any damage and address the field assumptions above, we rely on two contributing factors of a secure communication, which are *(1) encrypting* the communication channel, thus making sensitive data unintelligible to those who are not authorized, and *(2) authenticating* legit devices, therefore, rejecting malicious devices that could cause havoc. In our illustration in Figure 1.3, the legitimate 'green' devices are first accepted, and 'red' devices are rejected, and perhaps even reported, using authentication techniques. Later, all communication between 'green' devices in the field and the server are encrypted.

Although the use case described above sets the common ground in designing the solution, there are some other requirements that would shape the design of the solution later. This thesis is partly done at *Intrinsic ID*. Therefore the set of requirements for the project is reflected by the typical use cases in the vision of the company as well as its industrial customers. Intrinsic ID is a world leader in the field of Cyber-Physical Security Systems as a provider of Physical Unclonable Function (PUF) technology. PUF is a *hardware security primitive* [5] that can be used for silicon authentication. PUF is referred to as silicon fingerprint, the analogous to the unique and un-clonable human fingerprint. It uses unique physical hardware properties of a device for generating those fingerprints. Given a *challenge* as the input to the PUF, the output, often called the *response* is generated [6]. By applying certain clever techniques such as error correcting and privacy enhancing techniques to the response from the PUF, it is possible to derive a stable and device unique secret key [7] [8]. Using Intrinsic ID patented PUF technology, secret keys and identifiers are reliably extracted from the physical properties of chips [9].

In the context of this project, the PUF-derived key can be viewed as the primary *asset* of the secure system, and the application of PUF technology for secure communication is

one of the driving factors of this project. PUF technology gives us the unique capability to achieve a secure device-dependant cryptographic key. The PUF-derived secret key on the constrained IoT device can be used as the hardware root-of-trust in authenticating the devices in the field and enabling secure communication.

## 1.3 Problem Statement

As we have seen already, enabling secure communication for *lightweight and constrained* IoT devices is paramount for preserving our privacy, our security and our general safety. With the rapid proliferation of IoT devices, the challenges become even more profound. In general, one-size-fits-all solutions are not efficient, mainly because the use-cases are different and therefore the requirements are different. For example, a YouTube server is expected to perform key-establishment with each and every client; a client on the other hand, only does one key-establishment. Therefore, more often specialised and application dependent solutions need to be designed and developed. Providing a high quality and efficient solution for a specific purpose, that is, satisfying certain requirements such as minimising the silicon footprint, while integrating several components such as PUF hardware security primitive, is the research problem of this thesis. Therefore, the goal is to design and develop an architecture that is scalable, provides secure communication with many resource-constrained IoT devices, and satisfies predefined requirements including low power, small cost, etc. The problem is exacerbated, and remains a challenging task, due to the abundance of protocols, cryptographic primitives, diversity of devices and multiple methodologies such as software, hardware or co-design methods, and due to the lack of off-the-shelf solutions.

Under these circumstances, the problem statement of this thesis is to:

> Design and develop *an architecture* for *secure communication* with IoT devices in untrusted fields, using PUF-derived keys as the root-of-trust, where the key components are implemented in hardware while minimising the impact of the silicon footprint.

Naturally, this leads us to several important steps of the project:

(a) Investigate and analyse existing secure communication protocols (mainly key establishing[10]), and authentication methods using PUF

(b) Derive a protocol that can be used to solve the problem

(c) Investigate an existing appropriate hardware solutions, design and implement the core operation in hardware

(d) Propose a design of a hardware template of an architecture that augments the necessary components for making a self-contained system

(e) Evaluate the solution by simulating certain components, prototyping a demonstrator, experimenting and conducting performance analysis

(f) Further investigate optimisations on protocol level and architectural level

(g) Lay foundation for future research

### 1.3.1 Main Contributions

Designing application specific solutions are necessary in such a diverse environment of IoT devices and use cases. Securely transmitting data between authentic devices and host processors is crucial for preserving our privacy and security, and for general safety. To achieve a high quality and an efficient solution to our problem, a holistic approach is necessary for designing the architecture. To accomplish this, we heavily rely on (1) modern cryptography and a (2) PUF hardware intrinsic security primitive.

Cryptography has the means and tools for establishing shared keys, encrypting the communication as well as authenticating the devices. To have an encrypted channel, the communicating parties must share the same key. In this regard, we can make use of *key-establishing* protocols [10], which works on the principles of public-key cryptography. This key, in turn, can be used for encrypting the channel, by using the primitives that work on the principles of symmetric cryptography. Using certain additional cryptographic primitives, we can also achieve authentication. Using PUF technology, along with the cryptographic primitives described above, we can create a secure and robust authentication scheme for the IoT devices.

The focus of this thesis is secure communication for IoT. Establishing a secure authenticated communication potentially enables a range of other use cases and applications for PUF-embodied devices. Ultimately, the main contributions of this thesis are:

1. *Cryptographic Protocol Design:* A solution for a secure communication based on a cryptographic protocol is developed; it has the following properties:

    - Enables secure communication between *constrained* and a resource-rich devices, by establishing a secure shared key based on public-key cryptography; i.e., Elliptic Curve Cryptography (ECC) [10]. A particular emphasis was done on the constrained devices, where the optimal goal was to minimize the silicon footprint, therefore achieving a lightweight solution.
    - Utilizes PUF-derived key as the root-of-trust. PUF technology allows us to derive device dependant secure keys, and therefore provides the means to *authenticate* the silicon.
    - Uses additional authentication mechanism, i.e., challenge-response protocol based on the derived keys.
    - Additionally, we propose three *protocol variants* with trade-offs related to security versus implementation requirements. The main achievements of the variants are mainly: (1) enabling *mutual-authentication* and (2) *reducing non-volatile memory requirement* on the constrained device.

2. *Hardware Architecture Design and Development:* A modular hardware architecture that can be used for securing IoT devices based on the derived protocol is designed and developed; it is a complete and a self-contained solution, where safety critical

and compute-intensive operations are implemented in hardware to minimize area and keep them within a secure perimeter.

3. *Validation and Evaluation:* A prototype based on an instance of a designed hardware architecture was developed and implemented using a Xilinx Zynq-7000 APSoC device. This allowed us to verify one protocol variant and analyze its practicality and feasibility. Further, this prototype offered interesting insights and provided a solid foundation for an ASIC implementation and future research.

## 1.4 Thesis outline

This chapter motivates the work done in this thesis, introduces some relevant background information about cryptography, and describes the contributions of the work and provides the outline of the thesis. The rest of the thesis is structured as follows:

- *Chapter 2, An Overview of Security Systems.* This chapter motivates why we need secure systems, and provides an overview of what a secure system is. It discusses some of the prevailing threats on the Internet and the incentives of the attackers. The most used security requirements are presented, along with an introduction to cryptographic systems.
- *Chapter 3, Related Work.* This chapter gives a broad overview of public-key cryptography primitives, and shows related hardware implementations from both academia and the industry. Furthermore, we introduce a hardware intrinsic security primitive known as PUF.
- *Chapter 4, Protocol Design.* This chapter begins by providing an in-depth list of requirements for this project. It then proceeds to protocol design and proposes four protocol variants with trade-offs related to security quality versus implementation requirements.
- *Chapter 5, Hardware Design and Validation.* This chapter discusses the design and development of a modular hardware architecture that can be used to support the execution of a protocol variant described in Chapter 4. Further, it discusses the verification and the prototyping of a protocol variant on a Xilinx Zynq-7000 APSoC device, and analyzes its practicality practicality as well as its feasibility.
- *Chapter 6, Conclusion.* This chapter concludes the work done in this thesis.

# An Overview Of Security Systems

# 2

*In the previous chapter we have introduced the importance of secure communication, and outlined the challenges caused by the proliferation of Internet of Things (IoT) devices. Therefore we need to design and develop scalable, high quality and efficient secure solutions that are specially crafted for specific applications. To do so, a deep understanding of secure systems is needed. In this Chapter we take a closer look at security systems in general. In Section 2.1.1 we look at what attacks are, who the adversaries are, and what their incentives might be, along with a gentle introduction to how security system work and elaborates on widely agreed upon security requirements in the community. Section 2.2 briefly describes how some of these requirements can be achieved by giving a gentle introduction to cryptographic systems.*

## 2.1 Understanding Security

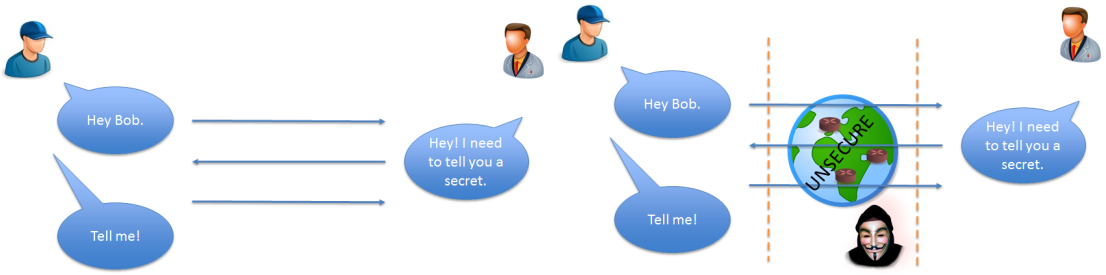
Unless we live in a society where the 'big brother is watching'[1] having a secret conversation in real life is straightforward. Just like in Figure 2.1(a), one can directly initiate a private conversation. If necessary, the two parties can whisper, or even hide far away from prying eyes. However, this is not the case on the Internet. On the Internet, the same secret message might have to traverse through different mediums, different Internet service providers, cross the ocean, get broadcasted via a wireless channels, etc. Anyone on this path with just a little incentive would be capable of tuning in and capturing those messages, as shown in Figure 2.1(b). Matters get even more complicated because in real life identifying the person you are talking to is easy, because you can see or hear the person. On the Internet, without special precaution steps, the recipient cannot be sure of the sender, nor sender can be sure about the recipient of the message. *Message integrity* is also an issue. How can one be sure that the message in transit has not been altered? How can we know that the message is coming from the right person? Answering these questions is exactly the purpose of secure systems. Obviously, understanding the potential threats and common incentives of attacks is critical for the development of appropriate solutions. This helps in defining the requirements and the properties of a secure system. Next, the incentives behind attacks will be discussed. Thereafter, the common security requirements are addressed.

### 2.1.1 Incentives behind threats and attacks

According to a report from the Kaspersky Lab, "On average enterprises pay US$551,000 to recover from a security breach".[2] One big Chicago bank lost $60 million to check-

[1] Book: *1984* by George Orwell

[2] http://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf




(a) Face to face conversation

(b) Message exchange on the Internet

Figure 2.1: Communication paradigms

related fraud by organised crime [11]. Piracy is another threat in the cyber world, since copying bits is virtually easy. Industry's losses due to piracy according to Business Software Alliance (1997) account $15 billion. These are just some of the few shocking facts that reveal the impact of cyber security breaches. What are the motives and incentives behind these attacks?

It is important to realise that the crimes that are done in the cyber-world highly resemble those in the physical world. Moreover, the incentives are usually very similar in nature too; they are referred to as the *unchanging nature of attacks* [12]. Theft, fraud, vandalism and other kinds of crimes can all be manifested in one way or the other into the cyber world. However, there is also the *changing nature of the attacks* in the cyber-world which make them potentially devastating. Schneier characterizes them with three properties: *Automation*, *Action at a Distance* and *Technique Propagation* [12]. No doubt that tasks could be automated on a computer: they are very good at performing repetitive tasks. Furthermore, because of the nature of the Internet and networks in general, actions can be carried out at a distance. A successful attack (e.g. or a well-developed tool) could be easily shared with, or propagated to, someone else. These have some implications that make the attacks in the cyber-world potentially more devastating and very different from the physical attacks. Schneier provides great examples of these implications in [12].

To put it differently, just like in the physical world, adversaries have different intentions, often malicious, such as stealing information by eavesdropping on some communication (*e.g. identity theft*), and modifying data (*e.g. altering bank account balance*). It is important to note that in some cases attacks are not malicious: an example would be an academic research in the field of security. Sometimes attacks are not intended and are simply caused by human error. Attack and their motivation is a vast subject, and an interested reader may refer to Part I of *Secrets & Lies* [11].

It is a difficult task to devise a single universal attack taxonomy. There exist many different attack classifications due to the complexity and the abundance of different technologies. Certain classifications are broad, and some are very specific. Certain articles focus on the software-attack taxonomy and other focus on hardware attacks [13] [14]. Another perspective would be to classify attacks as those coming from insiders versus outsiders. This suggests that not only the attacks but the adversaries could be classified

as well; an interesting classification of adversaries is provided in [15] and also mentioned in [16] is where the adversaries are classified into: *Class I: Clever Outsiders*, *Class II: Knowledgeable Insiders*, and *Class III: Funded Organizations*. Another attack classification is also provided in [17] that classifies attacks into three tiers: *social engineering*, *classical cryptanalysis* and *physical attacks*. Papp et al. [18] conducted a systematic review of public available data on existing threats and vulnerabilities in embedded systems and provide an extensive attack taxonomy.

To put it briefly, there are a huge number of attacks and possible threats to the system; and yet still more to be developed by someone in the future. This naturally leads us to the question such as: can we protect a system against all known and unknown threats and attacks? The answer is clearly a no. It is common to admit in the field of cyber security that given enough time and resources any system can be compromised [17] [14], hence there is no such thing as a 100% secure system. If a system is claimed to be secure by someone, the questions are 'from who'? and 'from what'? [11]

So what is the definition of a *secure system*? Unfortunately, neither the academic world nor the industry has converged on one single definition. Some argue that the system can be considered secure if the investment that is needed for a successful exploitation exceeds the potential gain by a tenfold. According to Pfleeger & Pfleeger, "Computer security is the protection of the items you value, called the *assets* of a computer or a computer system" [19]. This leads us to the question of how much effort needs to be put into protecting an asset. Nuclear plants controlled by computer systems would require much higher levels of security than a radio controlled car toy. Certain governmental organizations have adopted classifications for the assets such as *top secret*, *confidential etc.* The classification is useful because it acts as standardisation and sets a common ground. The different levels have specific guidelines associated with each level of security. Hence, these classification tags help determine the degree of security needed for protecting the asset. In the following section we briefly introduce certain security assumptions.

### 2.1.2 Security Assumptions

There are four common security requirements that are often mentioned in literature; these are *confidentiality, integrity, authentication and non-repudiation* [20]:

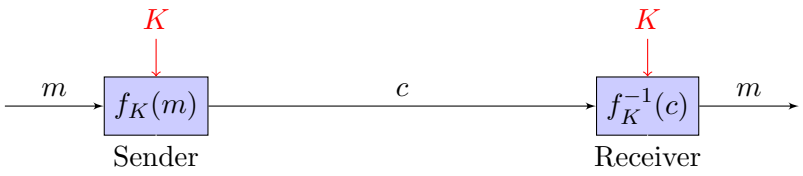
- *Confidentiality* is a service used to keep the information accessible only to the authorized users of the communication. This service includes both protection of all user data transmitted between two points over a period of time as well as protection of traffic flow analysis.
- *Integrity* is a service that requires that system assets and transmitted information can only be modified by authorized users. Modification includes writing, changing the status, deleting, creating, delaying, and replaying of transmitted messages.
- *Authentication* is a service that is concerned with assuring that the origin of a message, date of origin, data content, time sent, etc are correctly identified. This service is subdivided into two major classes: entity authentication and data origin authentication. Note that the second class of authentication implicitly provides data integrity.

- *Non-repudiation* is a service which prevents both the sender and the receiver of a transmission from denying previous commitments or actions.

## 2.2 Cryptographic Systems

To achieve the security properties discussed earlier we ought to use cryptographic algorithms. The majority of cryptographic algorithms that have appeared historically since the birth of cryptography could be categorised as what is known today as *symmetric algorithms*.

In symmetric algorithms both the sender and the receiver need to have a common *secret key* $K$ that is used for encryption and decryption [10]. Some *cipher* $f$ is used to encrypt a plaintext message $m$ and output an unintelligible ciphertext $c = f_K(m)$ to be sent. At the reciever side, an inverse cipher $f^{-1}$ is used to recover the plaintext message from the ciphertext using $m = f_K^{-1}(c)$ as shown in the figure below:




A common example that is used to demonstrate a symmetric-key algorithm in most of the textbooks on this subject is the naïve Caesar cipher. In Caesar cipher, the sender encrypts the plain text by individually rotating every letter a certain amount of rotations. Upon receiving the encrypted text, it is decrypted by rotating by the same amount of rotations in the opposite direction. The number of rotations here is the *key*. A simple example where $key = 3$ is shown in Figure 2.2. In this example $f_K(m)$ where $K = 3$ and $m =$ "$E$" would be $f_3(E) =$ "$B$".

Using this algorithm, two parties who are in possession of the secret key $K$ can now communicate 'secretly' over the Internet. Note, however, that this algorithm is simply used to demonstrate a concept here. If this algorithm is used for the English language, we would have a space of 25 keys, which is clearly a weak algorithm (as one can easily try all these 25 keys), and therefore insecure. Recent developments in cryptography have given us a great set of fast, practical and secure symmetric algorithms that are commonly used today for secure communication such as Advanced Encryption Standard (AES), DES, Blowfish etc. [21]. Their security is due to lengthier key sizes, therefore substantially complicating greedy brute-force attacks i.e., exhaustive key-search [10]. This is a simple

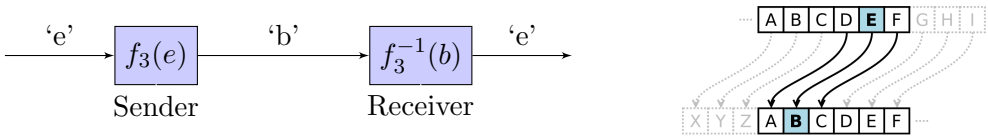



Figure 2.2: Ceaser cipher left shift of 3

attack of trying every possible key, until the correct key is identified. Today, a key size of 112-128 bits is known to be sufficient for a long-term security of several decades; however, advances in quantum computing will hinder this. Note that an exhaustive key-search is not the only way of attacking a system, there can be other weaknesses in a system that can be exploited. AES is a popular cipher that has been standardized by NIST to replace DES and Triple DES [22]. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. AES's data path is well structured, and certain hardware implementations take advantage of that by pipelining the architecture for increased throughput such as the one described in [23].

One of the obvious properties of symmetric cryptography that can be looked at as a drawback is that before establishing a secure communication, the secret key $K$ has to be somehow agreed upon, via some other secure channel e.g. in person or a courier. This particular trait certainly has several disadvantages; for one reason, simply negotiating a key before communication could be cumbersome and unpractical in many cases. This is known as the *key distribution* problem [10]. To illustrate this, let us consider the connected *mesh* network topology with $n$ nodes (e.g. users) as shown in Figure 2.3. In this case the number of keys that need to be negotiated beforehand for individual private communications grow quadratically: $\frac{n(n-1)}{2}$ [20] [10]. Furthermore, a lot of keys need to be stored and remembered by each user (node) on every device, which is often unpractical. Moreover, this could introduce other vulnerabilities to the secure system. As an example, a compromised device will reveal all stored secret keys and therefore all prior communication that has been done with this device would be accessible. There are some other disadvantages; an interested reader could refer to [20] [10]. In summary, the major shortcomings of symmetric algorithms are:

- Key Distribution Problem
- Number of keys that need to be stored

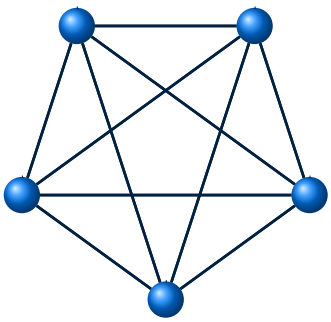


Figure 2.3: Mesh network topology requiring $\frac{n(n-1)}{2}$ links.

Nevertheless, researchers have never proposed to abandon symmetric cryptography; in fact, it is still the best tool for encryption. They rather look into how these shortcomings can be mitigated. A good foundation in understanding secret communication without prior key establishment is the Shamir's no-key protocol [24]. We will use a padlock analogy as shown in Figure 2.4 and explained here. Imagine Alice who wants to send

a secret parcel to Bob[3] using an untrusted shipping company. According to Shamir's no-key algorithms what Alice could do is use her padlock $A$ to lock the parcel with the corresponding $Key_A$ (as shown in Step 1) and ship it to Bob. Once received by Bob, he is still not able to open the parcel because Bob does not have Alice's $Key_A$. Instead, Bob attaches his own new padlock $B$ side by side with Alice's padlock (as shown in Step 2) and ship the parcel back. Alice would then remove her padlock using $Key_A$ (as shown in Step 3) and ship the parcel back to Bob again. This time, however, Bob will be able to open the parcel, simply by using his padlock's $Key_B$ that he owns (as shown in Step 4).

Notice that this protocol in not a key-based algorithm: it does not require Alice and Bob to be in possession of identical keys. Shamir's no-key protocol is the first known *three-pass protocol*. Mathematically, for this scheme to work, we must use *commutative cryptographic functions* [25]. This means that if the message was encrypted twice with different keys, it should be possible to remove the first encryption while leaving the second encryption. In other words, encryption and decryption are order-independent. In the next chapter, we discuss the related work, among which is the Public-key Cryptography (PKC): the building blocks that helps us tackle the issues discussed earlier as well as provide us with interesting new capabilities.

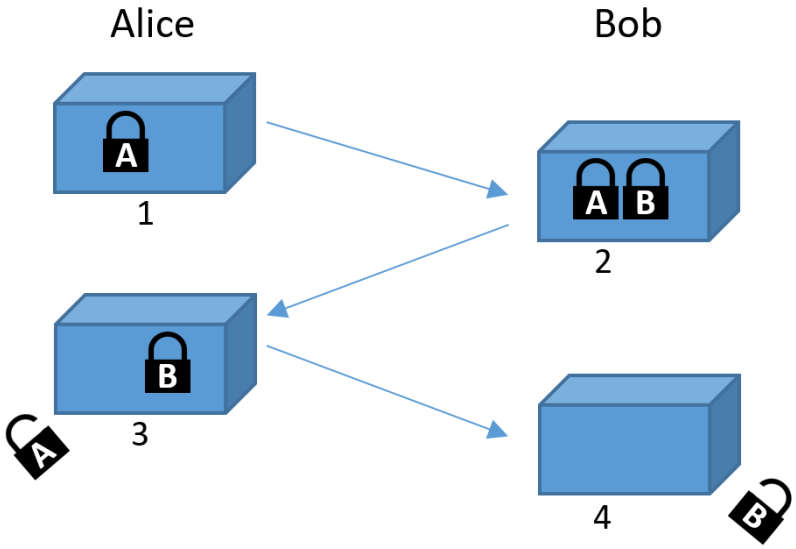



Figure 2.4: Shamir's no-key padlock analogy

[3]Alice and Bob are common placeholder names used in the field of cryptography.

# 3 Related Work

*In the previous chapter we introduced the concept of a secure system. We also introduced the concept of symmetric cryptography, and highlighted its shortcomings such as the key distribution problem. This chapter examines Public-key Cryptography, which mitigates those shortcomings, and equips us with additional tools that can help in achieving a more secure system. In addition, a hardware security primitive known as Physical Unclonable Function (PUF) is discussed in this chapter; they can further enable security as they are able to authenticate the silicon. The first three sections of this chapter discuss public-key cryptography; i.e. Section 3.1 describes the widely known Public-key Cryptography primitives in greater detail, Section 3.2 discusses aspects of Public-key Cryptography (PKC) implementation, and Section 3.3 highlights some commercial and academic hardware cores equipped with PKC primitives. The last section, Section 3.4, discusses Physical Unclonable Function and its current trends.*

## 3.1 Public-key Cryptography

The shortcomings and limitations that were described in the previous chapter have troubled engineers and scientists until Diffie and Helman published their famous paper 'New Directions in Cryptography' [26] in 1976[1]. The paper illustrates the shortcomings of symmetric algorithms and proposes *asymmetric algorithms or Public-key Cryptography (PKC).* Shortly after that, a key transport and digital signing scheme were proposed by Rivest, Shamir and Adleman (RSA) in 1977. Later in 1985, the use of elliptic curves in cryptography was suggested independently by Neal Koblitz [27] and Victor S. Miller [28]. Note that evidence exist that similar public-key techniques were used even earlier by British secret agencies[2], by the trio James Ellis, Clifford Cocks and Malcolm Williamson, who later were recognised and awarded[3].

In PKC the key is comprised of two parts: the *private key*[4] and the *public key.* The private keys are kept secret, and the public keys are made public. A public key $PK$ is used for encryption, and successful decryption is possible only with the corresponding private/secret key $SK$ as shown in Figure 3.1. A good analogy as shown in [10] is a traditional mailbox: anyone can deposit mail, but only the owner who is in possession of the key can unlock the mailbox and fetch all the mails.

PKC is not only used to encrypt by definition but in general, can provide the following mechanisms [10]:

[1] Awarded with the Turing Award, a Nobel Prize equivalent for computer scientists in 2015.
[2] https://www.gchq.gov.uk/note-non-secret-encryption
[3] http://www.bbc.com/news/uk-england-gloucestershire-11475101
[4] Often interchanged with the term *secret key.*

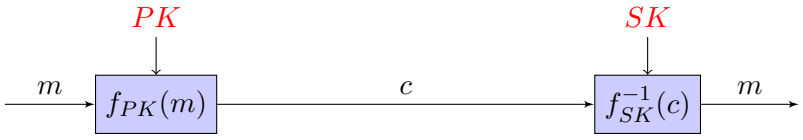



Figure 3.1: PKC Encryption and Decryption, where $SK$ and $PK$ are secret key and public key respectively

- Encryption: Making plaintext messages unintelligible for those who are not in possession of the secret key. The reverse process is decryption.
- Key Establishment: This is the most important operation in the context of this thesis. As stated earlier in Chapter 2, the problem with symmetric cryptography is that the communicating parties who wish to secure their communication must have a key that is known to both. Using this mechanism, two sides who do not share copies of the same key can securely *establish* a shared key, that can be later used for other purposes.
- Identification and Non-repudiation: Non-repudiation is a mechanism which prevents a party denying its actions e.g. a user who has signed a document shouldn't deny this action in the future. Identification, a closely related term can help in identifying messages, as well as aid in authentication. By using digital signatures, which is possible using PKC, we can add these mechanisms to our communication.

Theoretically, encryption using PKC is possible in certain cases e.g. RSA; however, public key algorithms are computationally more demanding than the symmetric algorithms, and hence are usually not used for *session encryption*; i.e., encryption of the ongoing communication session using a *session key*, a key that is valid only for a short period of time. Instead a *symmetric* algorithm is used. As discussed earlier, for symmetric algorithms to work, the principals need a shared secret key. PKC is traditionally used for session *key establishment*; i.e., *establishing secret keys over an unsecure channel*. In general there are two ways of doing this: *(1) key transport* or *(2) key agreement* protocols. In the former, a secure key is derived by a principal and is transferred to others (e.g. RSA key transport); in the latter, the principals involved (two or more) make their individual contributions to derive their shared secure key [29] (e.g. Diffie-Hellman key exchange (DHKE) protocol). Non-repudiation and identification can be realised by specific protocols using digital signatures [10]. A document can be digitally signed by a party using a secret key. The signature can be verified by using the public key. This simple method provides the non-repudiation mechanism, and naturally provides the means to verify the identification of the signee. Moreover, this is also a mechanism for providing integrity, a slight change in the document (e.g. flip of one bit) will not produce the same digital signature. Assuming, of course, that the secret key has not been stolen.

PKC algorithms are classified, according to the underlying mathematically/computationally hard problem that they are based on, into three known families [10]:

- **Discrete Logarithm problem family:** Difficulty based on the discrete logarithm

problem in finite fields. Diffie-Hellman key-exchange is an example from this family.

- **Integer-Factorization problem family:** Based on the difficulty of large integer factorization. RSA is the most prominent example in this family.
- **Elliptic Curve problem family:** These algorithms are based on the generalised discrete log problem in a finite field defined over an elliptic curve group. The underlying problem is commonly referred to as Elliptic Curve Discrete Log Problem (ECDLP).

In the subsequent sections, we briefly examine these families in chronological order of their appearance.

### 3.1.1 Discrete Logarithm problem family: DHKE

The *Diffie-Hellman Key Exchange (DHKE)* was the first PKC protocol introduced to the public that belongs to the Discrete Log Problem (DLP) family. Today, many versatile protocols that are widely used in modern cryptographic software libraries are based on DHKE. In this section we first give a formal definition of DLP, followed by the definition of DHKE. An illustration that helps understand the protocol is given along with a numerical example. We end this section by discussing the potential attack on the protocol by introducing the concept of Man-In-The-Middle (MITM) attacks, to which all PKC protocols are susceptible.

The formal definition of DLP as given in [10] is:

**Definition** Discrete Log Problem (DLP) in $\mathbb{Z}_p^*$
*Given is the finite cyclic group* $\mathbb{Z}_p^*$ *i.e., the set which consists of all integers* $i = 0, 1, \ldots, p-1$ *for which* $gcd(i, p) = 1$*, of order p-1 and a primitive element* $\alpha \in \mathbb{Z}_p^*$ *and another element* $\beta \in \mathbb{Z}_p^*$*. The DLP is the problem of determining the integer* $1 \leq x \leq p-1$ *such that:*

$$\alpha^x \equiv \beta \ mod \ p.$$

Figure 3.2: Discrete Log Problem (DLP) in $\mathbb{Z}_p^*$

If the chosen numbers are sufficiently large, computing the discrete logarithm i.e., $x$ as in Figure 3.2, becomes a very time consuming and a challenging task[10]. There exist several ways that try to compute the discrete logarithm, therefore 'solving' the DLP [10]. One may think of these as ways of attacking the DLP. In [10], the authors classify these algorithms into generic and non-generic algorithms. In the generic class, the general group operations are under the consideration. Example of generic are *brute-force search* algorithm, *baby-step giant-step* algorithm, *Pollard's rho* method, *Pohlig-Hellman algorithm* etc. Nongeneric algorithms, on the other hand, exploit special properties of a particular cyclic group. A prominent example of this kind of attack is the *Index-Calculus Method*. Detailed explanation of these attacks are beyond the scope of this section, an

interested reader may refer to [10] for more information. Note that these are the best-known algorithms for solving the discrete logarithm problem. Even though we know of these attacks, it is worth mentioning that it is still possible to achieve a certain level of bit-level security by choosing the appropriate parameter sizes (e.g. key size). Next we explore the well-known DHKE protocol that belongs to this family.

### Diffie-Hellman Key Exchange (DHKE) Protocol

One of the most famous protocols of DLP family is the Diffie-Hellman Key Exchange (DHKE); it was the first algorithm introduced to the public. DHKE consists of two main steps: (a) set-up or initialization and (b) key-agreement. These are discussed next.

During the set-up step, a set of parameters often called *domain parameters* need to be agreed upon; these are the initial parameters needed by the key exchange protocol in order to determine the secret key and realize the key establishment between the principals. In practice, the domain parameters are defined by either a *standard*, set-up by a trusted third party, or agreed upon by the participating parties in some other way. An interested reader may refer to the Request for Comments (RFC)[5] #2631 standard drafted by the Internet Engineering Task Force (IETF) for detailed requirements, examples and implementation guide.

**Diffie-Hellman Set-up**

1. *Choose a large prime* $p$
2. *Choose a postiive integer* $\alpha$*, such that* $\alpha$ *is a primitive root modulo* $p$
3. *Publish* $p$ *and* $\alpha$

Figure 3.3: Setup of Diffie-Hellman protocol

Figure 3.3 shows the set-up step as described in [10]. In the set-up step shown in Figure 3.3, $p$ and $\alpha$ are the DHKE domain parameters and are made public. Note that $\alpha$ must be a primitive root modulo $p$, also called the base. Using these parameters any two (or more) parties can generate a common secret key $k$ over an unsecure channel using the second step i.e. key-agreement.

[5] https://tools.ietf.org/html/rfc2631

**Diffie-Hellman Key Exchange**

| | **Alice** | | **Bob** |
|---|---|---|---|
| 1 : | *Choose* $a = k_{pr,A} \in_R \{2, ..., p-2\}$ | | $b = k_{pr,B} \in_R \{2, ..., p-2\}$ |
| 2 : | *Compute* $A = k_{pub,A} \equiv \alpha^a \ mod \ p$ | | $B = k_{pub,B} \equiv \alpha^b \ mod \ p$ |
| | | 3 : $A = k_{pub,A}$ $\longrightarrow$ | |
| | | 4 : $B = k_{pub,B}$ $\longleftarrow$ | |
| 5 : | $k_{AB} = k_{pub,B}^{k_{pr,A}} \equiv B^a \ mod \ p$ | | $k_{AB} = k_{pub,A}^{k_{pr,B}} \equiv A^b \ mod \ p$ |
| | ..... Key established: $k_{AB}$ ..... | | |

Figure 3.4: Diffie-Hellman Key Exchange protocol

During the key-agreement step, both participants contribute to the key establishment. Figure 3.4 shows the corresponding algorithm. Assume that both Alice and Bob are aware of the DHKE domain parameters. They proceed by randomly choosing their *private* keys $a = k_{pr,A}$ and $b = k_{pr,B}$ as shown in Line 1 of the protocol; the private key should be selected from a pre-defined set of integers at random; e.g. $a \in_R \{2, ..., p-2\}$. In Line 2, the private keys are used to compute the corresponding *public-keys*; Thereafter, these are exchanged as shown in Line 3 and 4; this is sometimes referred to as individual DHKE *contributions*. Finally, in Line 5, the shared key is computed by both parties. It is easy to see that after the key-agreement step, both Alice and Bob get the same key: Alice computes $k_{AB} \equiv B^a \equiv (\alpha^b)^a \equiv \alpha^{ab} \ mod \ p$, which is equivalent to what Bob computes: $k_{AB} \equiv A^b \equiv (\alpha^a)^b \equiv \alpha^{ab} \ mod \ p$. Remember that computations in the DHKE protocol are exponentiation in $\mathbb{Z}_p$ i.e., arithmetic modulo p. Therefore, the security of the protocol is based on the DLP in $\mathbb{Z}_p$, as stated previously in Figure 3.2

### DHKE Example

Next we will first illustrate the DHKE using a paint bucket analogy and thereafter using a simple numerical example. Figure 3.5 provides an illustration of DHKE using a paint bucket analogy. Alice and Bob start with a common paint, the domain parameters. They choose their secret colors and individuality mix it with the common paint. The resulting paint is their contribution; that is exchanged via an unsecured public transport. The secret colors are then added to the received pain bucket by both sides, resulting in a common paint, a common secret. Note that the secret color is never disclosed to the public nor to the other party at any point in time. The analogous to DLP in this example is that the mixture separation is expensive and unfeasible.

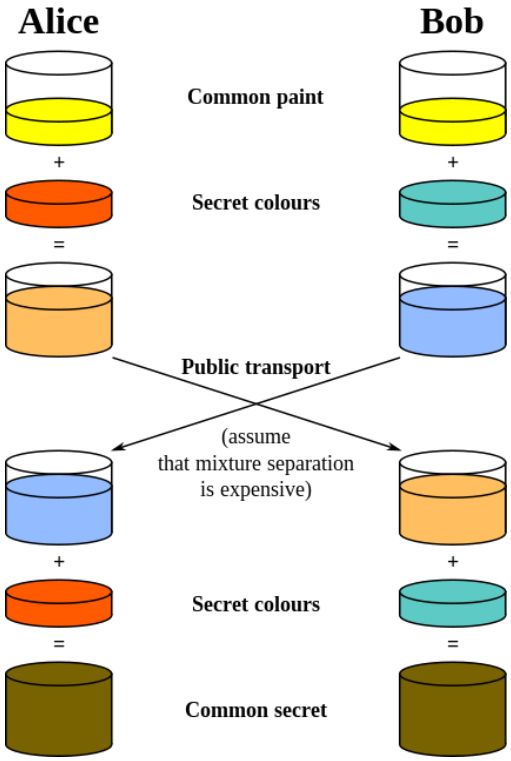



Figure 3.5: Diffie-Hellman Key Exchange illustration

Let us quickly look at a simple numerical example: Initially, Alice and Bob agree on the domain parameters: $p = 23$ and the base $\alpha = 4$. Thereafter, corresponding to Line 1 and 2 of the protocol: Alice chooses a secret integer $a = 6$, and computes $A = 5^6 \ mod \ 23 = 8$. Bob chooses a secret integer $b = 15$, and computes $B = 5^{15} \ mod \ 23 = 19$. After which their contributions are exchanged as in Line 3 and 4. As in Line 5, the shared keys are computed as follows: Alice computes $k_{AB} = 19^6 \ mod \ 23 = 2$, and Bob computes $k_{AB} = 8^{15} \ mod \ 23 = 2$. Alice and Bob now share a common secret $k_{AB} = 2$.

Notice that the individual contributions from Alice and Bob, $A$ and $B$, travel freely and are publicly available to anyone. As mentioned earlier, it is due to the DLP that finding the secret exponent is computationally intractable; i.e., no efficient algorithm exists today that make attacks feasible, as discussed earlier on page 19. Nevertheless, there exist other kinds of attacks on DHKE that are not based on solving the DLP, but still jeopardize the entire communication, one such attack is discussed next.

### Man-In-The-Middle (MITM) Attack

An eavesdropper (Eve) who is tapping the channel can easily reconstruct the secret key $k$ if one of the private keys $k_{pr,A}$ *or* $k_{pr,B}$ is known. However, even though Eve can capture Alice's and Bob's public keys $A = k_{pub,A}$ & $B = k_{pub,B}$, she is unable to calculate Alice's or Bob's private keys due to the DLP discussed above. However, public-

key based protocols are prone to a well-known attack known as the active *MITM* attack. The problem lies in the authenticity of the public keys. In a MITM attack, the attacker sits in the middle of the conversation, secretly relaying messages between communicating parties. Meanwhile, the communicating parties think that they are communicating with each other directly. Let us examine how a MITM attack is possible on the DHKE protocol shown in Protocol 3.4. An attacker, Mallory[6], can pretend to be Bob and perform DHKE with Alice, therefore establishing a shared key $k_{AM}$. At the same time, Mallory will pretend to be Alice and perform DHKE protocol with Bob, establishing a different separate shared key $k_{BM}$. The relaying then becomes trivial, messages $c$ transmitted from Alice are encrypted using some encryption function $c = E_k(m)$, where $k$ is the previously agreed $k_{AM}$ and $m$ is the original plain-text message. Since Mallory is in the possession of the key, she can retrieve the original message $m$ using a decryption function $m = D_k(c)$. Using a similar process, Mallory can now relay the original message $m$ or possibly a modified message $m'$ to Bob using the other, previously establish key $k_{BM}$. This is illustrated in Figure 3.6. The entire process is identical the other way around. As a result, the communication between Alice and Bob is jeopardized and can be eavesdropped and tampered, without their knowledge. Note that this problem is relevant to any public-key protocols; to circumvent this, we need to somehow authenticate public-keys. One way to protect public keys is through *certificates*. A certificate binds an identity of a user to their public key [10]. A hardware intrinsic primitive PUF, discussed in Section 3.4, can be used as a root-of-trust to authenticate the silicon of a device. e.g. using PUF, we can issue certificates that bind certain parameters to the silicon.

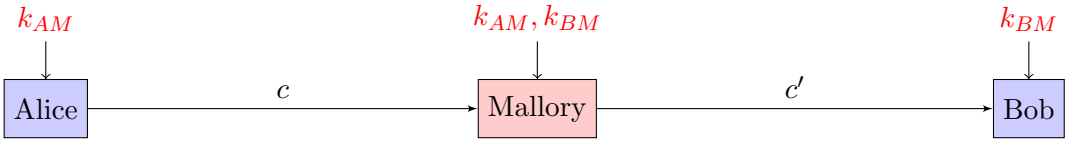



Figure 3.6: Successful MITM attack. Mallory has full access to the communication between Alice and Bob, without them being aware of it.

### 3.1.2 Integer-Factorization problem family: RSA

Shortly after the introduction of DHKE, Ronald Rivest, Adi Shamir, and Leonard Adelman introduced a full-fledged public cryptosystem in [30] that is based on a simple piece of classical number theory. The hardness of which is provided by the integer factorization problem; i.e., it is easy to multiply two integers but extremely hard to factorise the result; e.g. see [31].

There are no known efficient non-quantum algorithms that can do prime factorization; therefore, choosing sufficiently large numbers can guarantee a certain level of security even with the best-known factorization methods and algorithms (e.g. Pollar's rho). This

[6]Mallory is a common placeholder name for an active Man-In-The-Middle (MITM) attacker who can alter messages, delete, or inject messages; *Eve*, on the other hand, is a passive character who is an eavesdropper who can only listen but not modify the messages in transit.

section discusses the RSA construction and the potential attacks it can suffer from. First, we show how *key generation* in RSA is performed; i.e., the generation of private/public key-pair. Thereafter we show how encryption can be done using RSA, and show an easy to understand proof of correctness. A numerical example is given at the end for illustration purposes. Finally, we briefly discuss the potential attacks on RSA.

**RSA Key Generation**

RSA is most often used for [10]:

1. Encryption of small amounts of data: *Key transport*, discussed earlier in Section 3.1 is a prominent example of this.
2. Creating digital signatures: used for non-repudiation, identification, integrity etc.

Before discussing encryption, it is vital to understand the RSA *Key Generation Protocol* shown in Figure 3.7. First, two large prime numbers are chosen, after which their product is computed as shown in Step 1 and 2 respectively. In Step 3, the Euler's function $\phi$ is computed; it returns the number of integers between 1 and $n$ that have no common factors with $n$. After this step, the previously chosen primes can be discarded. The public exponent (part of the public key) is computed as shown in Step 5; i.e., a carefully chosen integer such that $e \in \{2, ..., \phi(n) - 1\}$, and that it is coprime with $\phi(n)$; i.e., $gcd(e, \phi(n)) = 1$. Once the public exponent is computed, we can proceed to computing $d$ (part of the public key), such that $de \ mod \ \phi(n) = 1$, as shown in Step 6. At last, once $e$ and $d$ are computed, $\phi(n)$ can be discarded as in Step 7. The final output of the RSA Key Generation step is the public/private key-pair; The **public key** is $k_{pub}$ =($e$,$n$) where $n$ is called the *modulus* and $e$ the *exponent*. The **private key** is $k_{pr} = (d, n)$.

**RSA Key Generation**

1 : *Choose two large primes p and q*
2 : *Compute* $n = pq$
3 : *Compute* $\phi(n) = (p - 1)(q - 1)$
4 : *p and q can now be discarded*
5 : *Select public exponent* $e \in \{2, ..., \phi(n) - 1\}$ *such that* $gcd(e, \phi(n)) = 1$.
6 : *Compute the private key d such that* $de \ mod \ \phi(n) = 1$
7 : *Computed* $\phi(n)$ *can now be discarded*

Figure 3.7: RSA Key Generation

**RSA Encryption/Decryption**

Now that we have discussed how RSA keys are generated, we can discuss *Encryption* and *Decryption* using RSA. To encrypt the plaintext message $m$ and get the ciphertext $c$, given the public key $k_{pub}$=($e$,$n$) we compute:

$$c \ = \ E_{k_{pub}}(m) \equiv \ m^e \ mod \ n, \tag{3.1}$$

where $c, m \in \mathbb{Z}_n$.
To get the original message $m$, the ciphertext $c$ has to be decrypted using the private key $k_{pr} = (d, n)$ as follows:

$$m = D_{k_{pr}}(c) \equiv \ c^d \ mod \ n. \tag{3.2}$$

Note that all operations are done in the integer ring where $\mathbb{Z}_n$ [10]. Hence the messages must be encoded in such way that they can be represented with the limited space of up to $n$.

The *correctness* of RSA encryption and decryption comes from the fact that

$$D_{k_{pr}}(c) = D_{k_{pr}}(E_{k_{pub}}(m)) \equiv \ (m^e)^d \ = m^{ed} \ mod \ n. \tag{3.3}$$

and by definition: *(see RSA Key Generation, Line 6)*

$$ed = k\phi(n) + 1 \tag{3.4}$$

where $k$ is some constant.
Therefore,

$$m^{ed} = m^{k\phi(n)+1} \tag{3.5}$$

Rewriting the exponents we get the following:

$$m^{ed} = m^{k\phi(n)+1} = m^{k\phi(n)}m^1. \tag{3.6}$$

From Euler's totient theorem [10], we know that $\alpha^{\phi(n)} \equiv \ 1 \ mod \ n$ when $gcd(\alpha, n) = 1$.
Therefore,

$$m^{ed} = m^{k\phi(n)}m^1 = m \ mod \ n. \tag{3.7}$$

■

An interested reader may refer to related literature such as [30] for more details. Next we provide a simple example of encryption and decryption using RSA.

**RSA Example**

Next we will provide a simple numerical 'textbook' example of RSA Key generation, and encryption and decryption using RSA. First we choose two distinct prime numbers,

such as $p = 61$ and $q = 53$, and compute $n = pq$ giving $n = 61 * 53 = 3233$. After that, we need to compute the totient: $\phi(n) = (p-1)(q-1)$ giving $\phi(3233) = (61-1)(53-1) = 3120$.

The next step is to select the public exponent $e$ such that $1 < e < 3120$ and that is coprime to 3120. We randomly choose $e = 17$. Now that the public exponent is chosen, we can compute the private key $d$, such that $de \ mod \ \phi(n) = 1$, $17d \ mod \ 3120 = 1$ resulting into $d = 2753$.

The public key, therefore, is $(n = 3233, e = 17)$. The encryption function is

$$c(m) = m^{17} \ mod \ 3233$$

. For an encrypted ciphertext $c$, the decryption function is

$$m(c) = c^{2753} \ mod \ 3233$$

As an example, to encrypt a message $m$ where $m = 65$, we get a ciphertext $c$ which is $c = 65^{17} \ mod \ 233 = 2790$. To decrypt the $c$, we compute $m = 2790^{2753} \ mod \ 3233 = 65$.

According to [10], there are three types of potential attacks on RSA:

- *Protocol Attacks:* Several protocol attacks have been introduced since the birth of RSA. However, protocol attacks could be mitigated easily if modern standards are properly followed.
- *Mathematical Attacks:* As mentioned earlier, the security of RSA is dependant on the Integer factorization problem. Obviously, the eavesdropper has an immediate access to the public key ($e$,$n$). Decrypting an intercepted encrypted text $c$ requires the private key $d$. As we see from the definition, one must calculate $\phi = (p-1)(q-1)$ to get the private key, where $p$ and $q$ are unknown. The product $n = pq$ is public, however, *factorizing the huge integer n into the prime factors p and q* is a *computationally hard problem.* As it was shown in the beginning, there are no efficient integer factorization algorithms today. A good survey on RSA attacks can be found in [32]. Nevertheless, due to the best-known factorization algorithms as shown in [33], it is important to choose the correct RSA parameters for a long-term security. Note that there exist efficient algorithms for solving the integer factorization problem as well as the DLP on quantum computers as shown in [34]. However, this is a topic for discussion in the future.
- *Side-channel attacks:* This is a different kind of attack, where physical properties such as power consumption or timing information can reveal secret information such as the private key $d$. Certain operations in RSA such as the modular exponentiation can reveal the private key with a simple power analysis [10].

### 3.1.3 Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) had come to the scene around 1985 when it was independently introduced by Neal Koblitz [27] and by Victor S. Miller [28]. ECC has been gaining popularity because achieving the same level of security as RSA in ECC requires much shorter operands [10]; shorter operands translate to shorter arithmetic operations, resulting in simpler hardware implementation. Therefore, ECC is an appealing PKC primitive for constrained devices. In this section, we will give a definition of Generalized Discrete Logarithm Problem (GDLP) and of Elliptic Curve Discrete Logarithm Problem (ECDLP). We will also show an Elliptic Curve Diffie-Hellman (ECDH) protocol. Due to the fact that this family is highly related to the Discrete Log problem family discussed earlier in the chapter, we omit common examples.

Figure 3.2 defined DLP over a multiplicative group $\mathbb{Z}_p^*$. The DLP can be defined over any other cyclic group. As given next [10]:

**Definition** Generalized Discrete Logarithm Problem (GDLP)
*Given is a finite cyclic group $G$ with the group operation $\circ$ and cardinality $n$. We consider a primitive element $\alpha \in G$ and another element $\beta \in G$. The discrite logarithm problem is finding the integer $x$, where $1 \leq x \leq n$, such that:*

$$\beta = \underbrace{\alpha \circ \alpha \circ ... \circ \alpha}_{x \text{ times}} = \alpha^x.$$

Figure 3.8: Generalized Discrete Logarithm Problem (GDLP)

ECC is based on the ECDLP which is the generalization of the DLP. This makes all discrete log based protocols, such as DHKE, suitable for ECC.

As described in [10], the elliptic curve over $\mathbb{Z}_p$, $p > 3$, is the set of all pairs $(x, y) \in \mathbb{Z}_p$ which fulfill

$$y^2 \equiv x^3 + ax + b \ mod \ p$$

together with an imaginary point of infinity $\theta$, where $a, b \in \mathbb{Z}_p$ and the condition $4a^3 + 27b^2 \neq 0 \ mod \ p$.

The security of the primitive is based on the ECDLP, which is defined next [10]:

**Definition** Elliptic Curve Discrete Logarithm Problem (ECDLP)
*Given is an elliptic curve $E$. We consider a primitive element $P$ and another element $T$. The ECDLP problem is finding the integer $d$, where $1 \leq d \leq \#E$, such that*

$$\underbrace{P + P + ... + P}_{d \text{ times}} = dP = T$$

*where $\#E$ is the number of points on the curve.*

Figure 3.9: Elliptic Curve Discrete Logarithm Problem (ECDLP)

Due to the similarities to the DLP, the ECDH highly resembles the DHKE protocol discussed earlier as is shown in Figure 3.10.

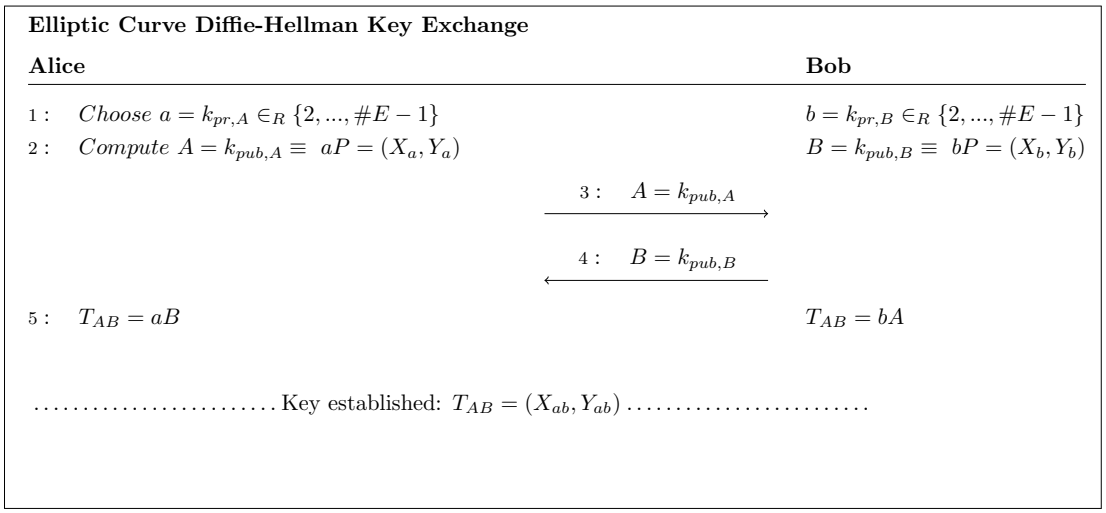



Figure 3.10: Elliptic Curve Diffie-Hellman Key Exchange protocol

Elliptic Curve Cryptography (ECC) implementation is often viewed in a four layered approach as shown in Figure 3.11. On the top level is the protocol; which relies on the *scalar multiplication*. Scalar multiplication is the most computationally intensive operation required by the protocol and is the primary target for optimization in this thesis. This layer depends on the elliptic curve group operations: *point addition* & *point doubling*. On the last (base) layer we have the finite field arithmetic operations such as the multiplication, addition, subtraction and inversion, without which the group operations above are not possible.

Multiple choices at each and every level of the abstraction pyramid are possible. The choices can be different protocols, different algorithms for scalar multiplication, different group operations depending on the curve, its parameters and the coordinate system, finite field arithmetic can also be done in various ways using different algorithms. Furthermore, a choice of a particular curve will directly reflect on the implementation as well. A choice made at a particular level can directly affect the choices in the lower levels. There have been advances made individually on different levels of the pyramid as well as a whole, that target different applications and optimization criteria [35]. There exist commercial and academic hardware IP cores that support ECC and implement the entire stack described above, which will be discussed later in the chapter.

## 3.2 PKC Key Components

Recall that since the introduction of public-key cryptography by Diffie & Hellman [26], multiple other primitives have been introduced such as the RSA by Rivest, Shamir and

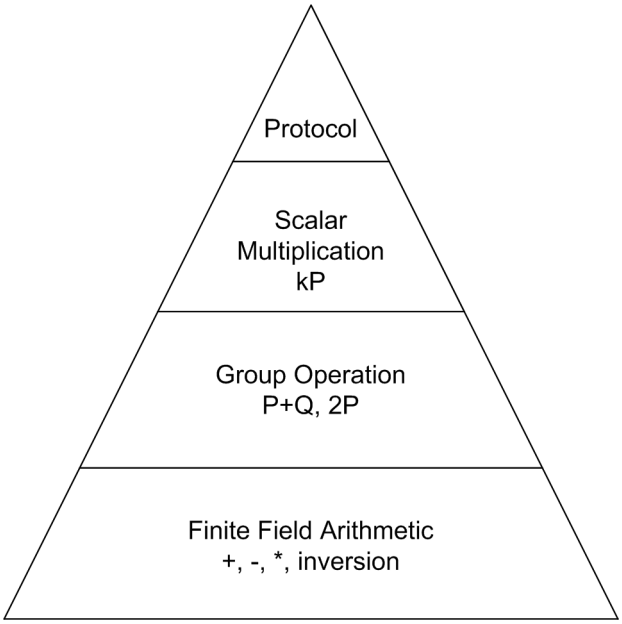



Figure 3.11: ECC four layer approach

Adleman in 1977 and a slightly newer development in 1985 was the ECC. As stated earlier in the chapter, these primitives are based on distinct hard to solve mathematical problems. Diffie-Hellman relies on the discrete log problem; RSA relies on the hardness of prime factorization. ECC is based on elliptic curve discrete log problem. Also, they usually involve a compute intensive function that dominates the majority of computation time such as modular exponentiation and scalar multiplication.

Specifically, in Diffie-Hellman and RSA, the compute intensive operation is modular exponentiation: computing $a^b$ in a finite field $GF(p)$. In other words, given base $b$, exponent $e$, and modulus $m$, the modular exponentiation $c$ is:

$$c \equiv b^e \ (mod\ m). \tag{3.8}$$

Recall from Section 3.1 that these cryptographic systems are usually based on large number arithmetic operations. The most straightforward method for modular exponentiation requires $e-1$ multiplications as shown below, with one modular reduction at the end.

$$c = \underbrace{b \ * \ b \ * ... * b}_{e \text{ times}} \ (mod\ m). \tag{3.9}$$

A simple improvement is to make modular reductions after every multiplication: it is more memory-efficient and maybe even computationally-efficient. Performing these compute intensive operations efficiently in practice is critical. There are a number of other techniques, algorithms and methods [35] that optimize these steps in software and hardware implementations, that target *speed* improvement, *memory access* reduction, *intermediate storage* optimization, and even *side-channel attacks* countermeasures etc.

For example, one technique that speeds up modular exponentiation is exponentiation by squaring, commonly referred to as square-and-multiply algorithms, or the successive squaring method[7], etc. The square-and-multiply method relies on the fact that the exponent $e$ can be represented in a binary form, hence the equation can be efficiently rewritten:

Exponent $e$ is first decomposed into a binary notation, where $n$ is the length of bits of $e$, or $log_2(e)$:

$$e = \sum_{i=0}^{n-1} a_i 2^i. \tag{3.10}$$

Rewriting $b^e$ with the above equation yields,

$$b^e = b^{\sum_{i=0}^{n-1} a_i 2^i} = \prod_{i=0}^{n-1} (b^{2^i})^{a_i}. \tag{3.11}$$

Therefore, the final modular exponentiation is:

$$c \equiv \prod_{i=0}^{n-1} (b^{2^i})^{a_i} (mod\ m). \tag{3.12}$$

Let us look at a simple example of square-and-multiply method for calculating $42^{42}$ $(mod\ 100)$: Firstly, the base $b$ is successively squared $log_2(e)$ times, which is the length of the binary representation, as shown in the Table 3.1 below.

Table 3.1: Successive squaring method example of $42^{42}$ $(mod\ 100)$

| $i$ | $a$ | $a^2$ | $a^2\ mod\ p$ | *remark* |
|---|---|---|---|---|
| 0 | 42 | 42 | 42 mod 100 = 42 | $42^1$ |
| 1 | 42 | 1764 | 1764 mod 100 = **64** | $42^2$ |
| 2 | 64 | 4096 | 4096 mod 100 = 96 | $42^4$ |
| 3 | 96 | 9216 | 9216 mod 100 = **16** | $42^8$ |
| 4 | 16 | 256 | 256 mod 100 = 56 | $42^{16}$ |
| 5 | 56 | 3136 | 3136 mod 100 = **36** | $42^{32}$ |

Then the exponent $e$ is decomposed into successive powers of two, simply put is rewritten in radix 2: $e = 101010_2$. From the simple radix conversion formula for fixed-radix positional number system, the exponent $e = 101010_2$ can be also written as $101010_2 = 2^5 + 2^3 + 2^2$. Our exponentiation can therefore be rewritten as $42^{2^5+2^3+2^1} = 42^{2^5} * 42^{2^3} * 42^{2^1} = 42^{32} * 42^8 * 42^2$. Therefore we must multiply the corresponding values, marked in bold, and make one final reduction in $GF(100)$: $36 * 16 * 64$ $(mod\ 100) = 64$. Using this simple algorithm we were able to calculate $42^{42}$ $(mod\ 100) = 64$. This algorithm is an improvement when compared to the straightforward method. The running time of this algorithm is $\mathcal{O}(\log_2 e)$, while the previous one was $\mathcal{O}(e)$.

[7] http://mathworld.wolfram.com/SuccessiveSquareMethod.html

Another compute intensive operation is *scalar multiplication* which is the fundamental operation in Elliptic Curve Cryptography (ECC). The most straightforward method for scalar multiplication requires $d-1$ point additions as shown below:

$$dP = \underbrace{P + P + ... + P}_{d \text{ times}}. \tag{3.13}$$

where $d$ is a scalar and $P$ is a point on an elliptic curve.

Point addition defined over a finite field elliptic curve group is often a complex function, which depends on the chosen curve and domain parameters, and usually involve field addition, subtraction, multiplication and inversion. Point addition where two points are identical, $P+P$, is also known as point doubling, an is commonly a computationally simpler than point addition. For improvements, it is important to minimize the number of steps as much as possible, and potentially use point doubling instead of point additions. One way of doing this is using the *double-and-add* algorithm, in some sense an algorithm similar to the square-and-multiply algorithm. Achieving a running time $\mathcal{O}(\log_2 d)$, while the previous one was $\mathcal{O}(d)$. Algorithm 1 below is the pseudocode for the double-and-add algorithm.

**Algorithm 1** Double-and-Add algorithm

```
1: procedure DOUBLE-AND-ADD(d, P)                ▷ Scalar Multiplication of dP
2:     Q ← 0
3:     for i:=m downto 0 do
4:         Q := point_double(Q);
5:         if d_i == 1 then
6:             Q := point_add(Q, P);
7:         end if
8:     end for
9:     return Q
10: end procedure
```

The double-and-add has one major drawback: a *side-channel*. Point double and point addition operations are inherently different. Therefore, using differential power-analysis, one can reveal the scalar, which is often the secret key. Consequently, other techniques have been developed to prevent side-channel attacks. One such method that is commonly used is the Montgomery Ladder [36], which is later used in this thesis. It is a constant time algorithm which is resistant to this side-channel attack. Other methods that are resilient to such attacks are described in [37]. Other algorithms exist for scalar multiplications, such as the windowed algorithm, sliding-window algorithm, the w-ary non-adjacent form (wNAF) method, etc.

The algorithms discussed above were optimizations of the scalar multiplication layer as shown in Figure 3.11 RSA also has some well-known high level optimization techniques such as the Chinese Remainder Theorem for decryption.[8] Note that optimisations are

[8]RFC2437 PKCS #1: RSA Cryptography Specifications

possible and have to be done at lower levels too for achieving optimal results. For instance, the modular multiplications that were used earlier can be optimised too. One such optimisation is, for example, the *Montgomery Modular Multiplication* [35]. It is important to note that certain optimisations are more relevant to software implementations. Some optimization techniques are specific for hardware design. For instance, there are several ways of doing modulo reduction in hardware. One way is to attach a modular reduction circuit to an output of a standard binary multiplier. However designing special modular multipliers are often simpler and could be more optimal. In addition, squaring is a special case of multiplication that could also be optimised for better results [38].

Modern standard cryptographic software libraries such as OpenSSL support discrete logarithm and elliptic curve based cryptography and therefore heavily rely on the operations discussed above and their optimizations. OpenSSL is an open source project that provides a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols[9]. In general, PKC is known to be more computationally intensive primitive; hence, new developments are constantly in progress that aim to optimise these operations for efficiency purposes. Often the optimisations that have proven to be efficient without jeopardising the security of a system are adopted by standards and/or are implemented in popular software libraries and hardware implementations. It is important to note that hardware designs often have specific optimisations goals such as optimising for silicon area, power consumption etc. In the following section, we explore some of the existing commercial and academic hardware cores.

## 3.3 PKC Cores

In resource constrained devices, ECC is preferred over RSA, because it requires less storage, less power, less memory, and less bandwidth[10] due to their shorter key sizes and therefore operands compared to other public-key cryptosystems such as RSA and Diffie-Hellman [10].

Figure 3.2, taken from the *Guidance for Cryptographic Algorithm and Key-Size Selection* of *NIST SP 800-57* as is, shows the key sizes for various algorithms. Column 1 shows the estimated maximum security strength in bits for that particular row. Column 2 shows a symmetric algorithm that can provide such strength. Column 3 shows the parameters of Finite Field Cryptography (FFC) (e.g. Diffie- Hellman) for achieving that security level, where $L$ is the size of the public key, and $N$ is the size of the private key. Column 4 shows the parameters that are necessary for Integer Factorization Cryptography (IFC), where $k$ is commonly referred to as the key size, but is the size of the modulus n of the particular IFC algorithm. Lastly, Column 5 provides the parameters for ECC for achieving the security level as shown in column 1. Here $f$ is the range of the size of the order of the base point $G$. This value is commonly referred to as the key size. Note that it is assumed that the keys are generated under specific rules that render them to be secure. BlueKrypt[11] provides a detailed comparison of key-lengths

[9] https://www.openssl.org/

[10] https://www.certicom.com/index.php/the-basics-of-ecc

[11] https://www.keylength.com

| Security Strength | Symmetric key algorithms | FFC (e.g., DSA, D-H) | IFC (e.g., RSA) | ECC (e.g., ECDSA) |
|---|---|---|---|---|
| ≤ 80 | 2TDEA[21] | $L = 1024$<br>$N = 160$ | $k = 1024$ | $f = 160\text{-}223$ |
| 112 | 3TDEA | $L = 2048$<br>$N = 224$ | $k = 2048$ | $f = 224\text{-}255$ |
| 128 | AES-128 | $L = 3072$<br>$N = 256$ | $k = 3072$ | $f = 256\text{-}383$ |
| 192 | AES-192 | $L = 7680$<br>$N = 384$ | $k = 7680$ | $f = 384\text{-}511$ |
| 256 | AES-256 | $L = 15360$<br>$N = 512$ | $k = 15360$ | $f = 512+$ |

Table 3.2: NIST SP 800-57 Pt. 1 Rev. 4, Comparable Strength

for various PKC algorithms based on different methods provided by institutions such as ECRYPT II, NIST and NSA. For an extensive explanation, please refer to the NIST special publication *NIST SP 800-57 Pt. 1 Rev. 4*[12].

A common security requirement nowadays is to achieve a 128-bit security level. As it can be seen from Table 3.2, for that particular level of security, ECC key is 3072/256=12 times smaller than an IFC and FFC keys. Clearly, this has a substantial implication on their implementations. For example, shorter key sizes require less storage area. And most importantly, shorter key sizes mean that arithmetic is done with shorter operands, consequently, reducing the area of hardware components such as the Arithmetic Logic Unit (ALU).

Since our main focus are constrained devices where area is scarce, we would narrow our focus to ECC primitives. In the next section, we explore the current developments of ECC cores in the industry and the academia.

### 3.3.1 Commercial Products

There is an abundance of public-key cryptography IP cores in the market. Their specifications are usually limited to the publicly available datasheets that do not particularly follow any predefined standard; some provide details, some only provide a broad overview. This makes them somewhat difficult to compare to each other. For example, the metrics used for silicon area usage can be different for different IP cores; the overall functionality that they achieve is often different, and their interfaces are often different, etc. For instance, some cores implement an Advanced Microcontroller Bus Architecture (AMBA)[13] bus for an easy System on Chip (SoC) integration. Some cores are optimised to be highly speed-efficient, whereas some are optimised for the area, hence making them slow.

[12] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf
[13] https://www.arm.com/products/system-ip/amba-specifications

| Commercial IP Cores *Description* | Functionality | Area | Scalability |
|---|---|---|---|
| BarcoSilex<br>*Public Key IP core for RSA, ECC, ECDSA & ECDH* | RSA, ECC | 1. as low as 17k gates<br>2. fits into the smalles FPGA families | Scalable DSP Array<br>4 up to 256 Multipliers |
| Fraunhofer<br>*Processor core for Elliptic Curve Cryptography* | ECC | | selectable number of multiplicaiton cores |
| IPCORES<br>*Elliptic Curve Point Multiply and Verify Core* | ECC | less than 8k Actel tiles<br>less than 10K ASIC gates | |
| EnSilica | ECC | selecting between a 'small' and 'medium' gate count architectures | |

Table 3.3: Commercial Public-Key Cores

Certain IP cores come with nice features such as customization, e.g. where the number of arithmetic units can be customised. This feature allows the user to adapt the core to meet their requirements. The majority of suppliers allow you to optionally choose counter-measures against simple and statistical timing side channel attacks, simple power analysis, differential power analysis, doubling attacks etc.

Clearly, commercial cores incur license fees, which are often not listed. The prices can be obtained by request and are typically negotiable, which depend on many factors such as the purchase volume etc. Moreover, some IP cores will incur additional cost due to export regulations of certain countries. These costs play a significant role in choosing the right core.

Some commercial companies that provide such IP cores are BarcoSilex, Fraunhofer, IPCORES and EnSilica. Often these are fullfledged cryptographic processors that can do symmetric and asymmetric crypto, digital signing, hashing, etc. Table 3.3 summarises some of the essential features of these commercial cores from their respective datasheets. As one can see, area comparison is not trivial. Although the number of gates is specified for some cores, the technology node is not. In the case of EnSilica, the area can range between 'small' and 'medium'. IPCores claim that ECC1 core is the smallest ECC core on the market that also achieves a high throughput. All cores are usually delivered with either synthesizable RTL source code or a netlist along with some other supporting deliverables such as test-benches, software libraries, documentation, etc. which allow for a smooth integration process. One other core that is worth looking at is the Dragon-QT core.

Dragon-QT[14] is an IP core that integrates Athena's TeraFire F5200 security microprocessor with Intrinsic-ID's Quiddikey-Flex secure key management. It implements nearly any cryptographic operation, including AES, SHA, elliptic curve cryptography, public key cryptography, advanced true random number generation, SCA/DPA countermeasures, and more. This extends the Quiddikey capabilities by providing an extensive cryptographic solution.

[14]https://www.intrinsic-id.com/wp-content/uploads/2014/09/2015-01-Brochure-Dragon-QT.pdf

### 3.3.2 Academic Implementations

The academia have been doing similar research and has implemented some different implementations. For example, TinyECC is a configurable elliptic curve cryptography software package aimed for constrained devices [39]. Similary, [40] presents an implementation of ECC Curve25519 [41] for MSP430[15] microcontrollers that target low-power applications. However, our focus is on hardware implementations of similar systems. An example of a hardware implementation is [42] that implements ECC using Curve25519 [41], achieving a throughput of more than 32k scalar multiplications per second on a Xilinx Zynq 7020 FPGA. Kumar and Paar investigated the feasibility of ECC on constrained devices and proposed an ASIC implementation of such processor [43] in 2006.

*NaCls Crypto_box in Hardware* [44], which we will refer to as "NaCl core" or simply "NaCl", is an example of low-resource hardware implementation of the widely known *crypto_box* function of the 'Networking and Cryptography'(NaCl) (pronounced "salt") crypto library. The NaCl core is in public domain making it worthwhile for further investigation. NaCl uses Curve25519 elliptic curve which is supported by the popular OpenSSL library and is included in the TLS1.3 draft. This is the only low-resource hardware implementation of Curve25519 to our knowledge. The NaCl core supports the X25519 Diffie-Hellman key exchange using Curve25519, the Salsa20 stream cypher, and the Poly1305 message authenticator. The NaCl core is implemented as an Application Specific Instruction Processor (ASIP), with a silicon area utilization of 14.6kGE. It consumes less than 40uW of power consumption at 1MHz frequency for a 130nm low-leakage CMOS process technology [44].

[15] http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/overview.page

## 3.4 Physical Unclonable Function (PUF) Technology

### 3.4.1 Background on PUF

The need for secure systems is growing, but fortunately there are tools we can use to provide the security. One such tool is a hardware security primitive known as Physical Unclonable Function (PUF), referred to as silicon fingerprint, the analogous to the unique and unclonable human fingerprint. Maes describes PUF as 'an expression of an inherent and unclonable instance-specific feature of a physical object' [45]. The concept of PUF was first introduced by Pappu in his PhD dissertation in 2001 [5]. Given a *challenge* as the input to the PUF, a unique and device dependant output, often called the *response* is generated [6]. It is possible to derive a stable and device unique secret key by applying certain techniques such as error correcting and privacy enhancing techniques to the PUF response [7, 8]. The unique secret key generated by the PUF could be used in multiple different use cases [46]. It has found applications in technology anti-counterfeiting, device authentication & hardware/software binding applications [8].

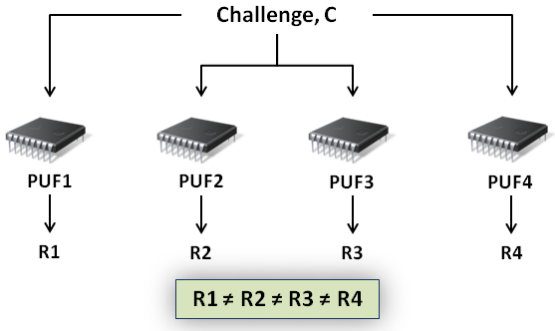



Figure 3.12: PUF challenge-response uniqeness

Some other properties are important to PUF such as reliability, uniqueness, etc. but are outside the scope of this thesis. For more literature on reliability and robustness of SRAM PUF, an interested reader may refer to [17].

Although it is a relatively recent technology, there is a broad taxonomy around PUF today [47]:

(I) *Non-electronic PUFs*: This is the PUF where we rely on the non-electronic PUF properties such as the random fibre structure of the paper or a scattering characteristic in an optical medium.

(II) *Electronic PUFs*: PUFs where we use electric properties such as resistance and capacitance.

(III) *Silicon PUFs*: This is a subclass of the Electronic PUFs i.e. ICs exhibiting PUF behaviour.

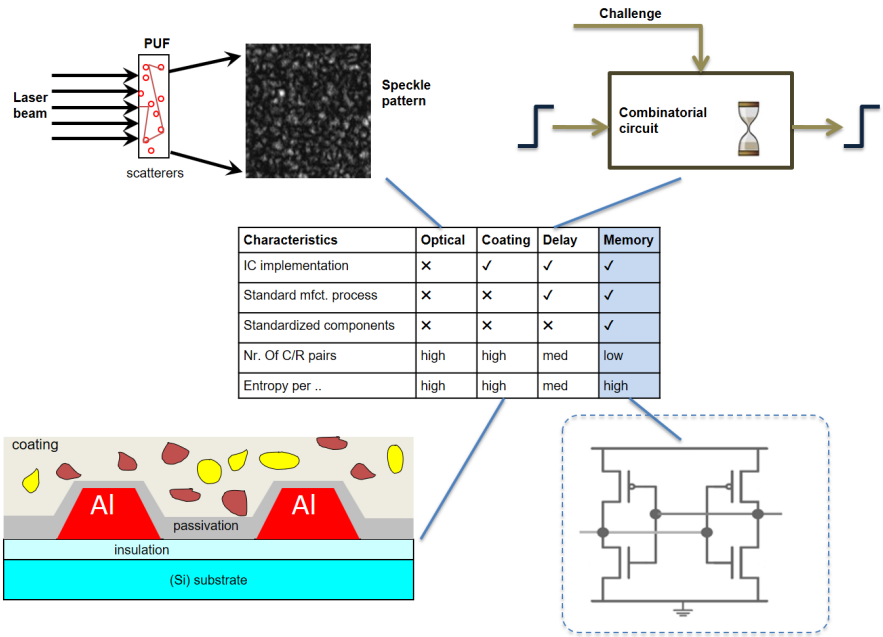
Other classification exist that are based on how the PUF is constructed:

(i) *Non-intrinsic PUF constructions:* These are constructions with explicitly-introduced randomness.

(ii) *Intrinsic PUF constructions:* Predominantly silicon PUFs based on random process variations that are hard to control. They can further be categorized into three classes [47]:

(a) Delay-based silicon PUFs
(b) Memory-based PUFs
(c) Mixed-signal PUFs

The illustration in Figure 3.13 shows some of the different kinds of PUFs and summarises their properties. Optical and Coating PUFs are examples of non-intrinsic (or explicitly introduced) PUF construction. In an optical PUF, a transparent layer is doped with light scattering particles. When a laser beam shines through it, a random and unique speckle pattern can be observed [48]. Although the number of Challenge-Response (CR) pairs and the entropy is high, its downside is that it is not an IC implementation, not a standard manufacturing process nor uses standardised components. In coating PUFs, the coating that covers the Integrated Circuit (IC) is doped with random dielectric particles; that would result in variable capacitance. An array of sensors is used to measure the capacitance of the coating [49]. Delay and Memory PUFs are examples of intrinsic constructions. Delay PUFs are based on the delays in wires and gates.




| Characteristics | Optical | Coating | Delay | Memory |
|---|---|---|---|---|
| IC implementation | × | ✓ | ✓ | ✓ |
| Standard mfct. process | × | × | ✓ | ✓ |
| Standardized components | × | × | × | ✓ |
| Nr. Of C/R pairs | high | high | med | low |
| Entropy per .. | high | high | med | high |

Figure 3.13: PUF Types

Using PUFs, we can authenticate the 'silicon' of the constrained IoT device. Memory-based PUFs are particularly useful in our case, as we will see in the following section.

### 3.4.2 SRAM PUF

We would now focus on the intrinsic, memory-based PUF construction, specifically the Static Random-Access Memory (SRAM) PUF. SRAM PUFs are more widely used in the industry because they can be built using standard digital IC components. An SRAM-based PUF uses the intrinsic randomness PUF of the start-up values in the SRAM. Figure 3.14 shows a typical six transistor SRAM cell design, consisting of two cross-coupled Complementary Metal Oxide Semiconductor (CMOS) inverters using four transistors $M_1$ through $M_4$. Transistors $M_5$ and $M_6$ are known as the pass transistors. The wordline (WL), bitline (BL) and its complement are used to access the cell. For

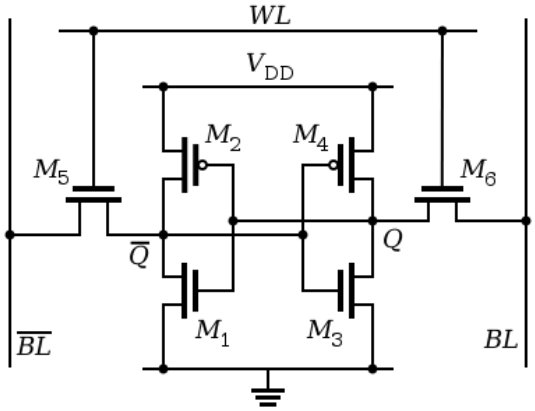


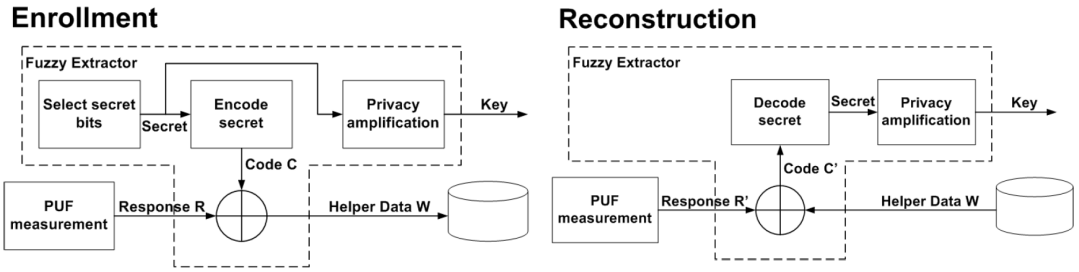

Figure 3.14: Conceptual schematic of a 6T SRAM cell

performance reasons, the two inverters in the SRAM cell are designed in a well-balanced, symmetrical way. However, the small and random submicron process variations due to manufacturing process cause different physical properties of the transistor. These differences in the transistors of the SRAM cell causes a skew. Due to this skew, a cell acquires a preferred state of a logic '0' or a logic '1' when powered on, referred to as one bit of 'electronic' fingerprint. This phenomenon of inherent, device unique variations makes SRAM PUFs construction possible. It is resistant to cloning even if one can get their hands on the circuit design/layout files since the skew is not visible in the layout. With the current manufacturing process variations are inevitable and cannot be controlled; therefore, cloning an SRAM PUF yields to be tough or even impossible [45].

An array of *uninitialized* SRAM cells provides a device fingerprint that uniquely identifies each device [50]. We can use these properties to securely store or generate a cryptographic key on a PUF embodied device. This is referred to as a key derivation based on the SRAM PUF. The key must be reliably extracted every time and must be unpredictable. Techniques such as *entropy extraction* and *error-correcting codes* are used to achieve this [7].

Entropy extraction can be achieved by using key derivation function. A key derivation function, e.g. a cryptographic hash function, is meant to produce a high quality cryptographically secure key.

Upon power-up, the PUF response will be noisy, when compared to initial measurement, referred to as the reference PUF response. Error-correction is a necessary step in




(a) Enrollment (b) Reconstruction

Figure 3.15: The Enrollment and Reconstruction operations of PUF-based Key Derivation.

a key-derivation system due to this noisy *bit error* phenomenon in SRAM PUF. Selimis et al. [51] conducted experiments on a 90nm 6T-SRAM ICs under various stress conditions to demonstrate PUF reliability. Temperature variation, voltage variation, voltage ramp-up, data retention tests as well as an aging test were performed. These statistical measurements were done by taking a reference measurement and the consequent measurements on the same chip. Results showed that temperature variation had the biggest impact on noise, causing a 19% error. Whereas, voltage variation had the least impact, introducing only 6% noise.

Additionally, they examined the relationship between different SRAM devices. For a secure key extraction, the devices must be unique and independent. Such independence can be measured as the *Hamming distance* between the responses of different devices. A Hamming distance $d$ is the number of positions in which two binary strings $x$ and $y$ of the same length $n$ differ, formally defined as:

$$d(x, y) = |\{i | x_i \neq y_i, 0 \leq i \leq n\}| \tag{3.14}$$

A fractional Hamming distance between $x$ and $y$ is defined as:

$$\delta(x, y) = d(x, y)/n \tag{3.15}$$

Let us quickly observe the two extremes: a $\delta(x, y)$ between two SRAM start-up values that equals to 0 signifies that $x_i = y_i \; \forall \; i$. On the other hand, a $\delta(x, y) = 1$ implies that $x_i = NOT(y_i) \; \forall \; i$. Clearly, the desirable value is $\delta(x, y) = 0.5$, which signifies independence between the start-up values of different device. Their results show that this distance is distributed around 0.5 for their selected SRAM, therefore making it a suitable PUF. Note that an SRAM PUF must be profiled in such way to make sure that it can be used.

Typically, for error-correction a code-offset method is used where helper data is generated during the *enrollment* phase, and is later used in the *reconstruction* phase to correct the bit-errors in SRAM as shown in Figure 3.15. The operations are explained in more detail below.

**(a) Enrollment**
The purpose of this stage is to generate a cryptographic key that is based on this unique device fingerprint. First, the SRAM values are read, which is the PUF reference response ($R$). This response is an input to the fuzzy extractor[16] [52, 53, 54] as shown in Figure 3.15(a). The fuzzy extractor in this case constitutes of two parts: *ECC encoding* and *Privacy amplification*. The ECC Encoding is responsible for encoding a random secret ($S$) with a chosen error correction code. The Helper Data [17]($W$) is computed by XORing ($\oplus$) $R$ with the encoded secret as shown in Equation 3.16. The secret key is an output of the privacy amplification block. Privacy amplification is a necessary step for getting a cryptographically secure key with maximum entropy.

$$W = R \ \oplus \ C = R \ \oplus \ Encode(S) \tag{3.16}$$

**(b) Reconstruction**
Reconstruction phase is for recovering the secret key at a later point in time in the field. A new PUF response is measured ($R'$), which is used as an input to the fuzzy extractor, along with the helper data ($W$) that was previously generated during the enrollment stage as shown in Eqn. 3.16. The helper data is XORed with the new PUF response to get the noisy codeword $C'$ as shown below:

$$C' = R' \ \oplus \ W \tag{3.17}$$

Substituting Eqn. 3.16 into Eqn. 3.17 we get Eqn. 3.18, the output of which is the encoded secret $S$ with noise ($R \ \oplus \ R'$).

$$C' = R' \ \oplus \ W = R' \ \oplus \ (R \ \oplus \ Encode(S)) = noise + Encode(S) \tag{3.18}$$

The secret is successfully reconstructed as shown in Eqn. 3.19 if the PUF response is close enough to the reference PUF response, i.e. if it is within the chosen error-correcting code correction capability.

$$S = Decode(noise + Encode(S)) \tag{3.19}$$

### 3.4.3 Current Trends in SRAM PUF Technology

The current trends in PUF technology mostly deal with improving their quality, especially those of the key generators. The quality is primarily determined by two properties: the reliability and the security of this hardware security primitive [7]:

1. *Reliability:* if the occurrence of (bit) errors between PUF reference response and the consequent PUF responses is limited, then with high probability the original key will be properly reconstructed.

2. *Security:* if the reference PUF response is sufficiently unpredictable, then the key is secure even to a party which observes the Helper Data.

[16] https://www.intrinsic-id.com/physical-unclonable-functions/fuzzy-extractor/
[17] The term *Helper Data* can be interchanged with *Activation Code* in this thesis.

Ideally an SRAM PUF should be *unbiased* to prevent any information leakage. In an unbiased PUF, '0' bits and '1' bits are evenly distributed, meaning that each single bit has the same chance to be a '0' or a '1'. In other words a fractional *Hamming weight* of 0.5. A Hamming weight is the number of non-zero elements in a binary string $x$, formally defined as:

$$w(x) = |\{i|x_i \neq 0\}| \tag{3.20}$$

Van der Leest et al. [7] point out that experiments show that most PUFs are not unbiased. A deviation from $w = 0.5$ in SRAM response is known as the *global bias*. A global bias cause entropy leakage, hence reducing the entropy of the cryptographic key. Van der Leest et al. [7] present a number of 'debiasing' methods evaluated in terms of reliability, efficiency, leakage and re-usability where secure secret keys are still generated from PUFs with arbitrary large global bias.

Other research is done in exploring the trade-off between the number of SRAM cells needed for achieving a full entropy key using different error-correcting codes. Leest et al. propose new methods for reducing existing known method where 4700 SRAM cells are needed for securely storing a 128-bit key to 3900 SRAM cells (17% reduction) and to 2900 SRAM cells (38% reduction) at a cost of a more sophisticated decoders.

There are numerous open problems concerning PUF technology, and typically several papers are published per week. Current research typically investigates different types of PUFs, applications of PUFs, new attacks on PUFs etc.

## 3.5 Discussion

It is a challenging task to identify the exact state-of-the-art solution because the project encompasses a broad scope, comprising of several different topics and layers of abstraction. Therefore we investigate the state-of-the-art of individual components and subsystems of the entire architecture. At the highest level are the protocols that can solve our problem. On lower levels is the implementation of the arithmetic operations in hardware that are necessary for the appropriate computations, with all the other steps in between. In this chapter, we have looked at some prior work done in public-key cryptography as well as some of the existing hardware implementations that implement those cryptographic primitives. We have also looked at a hardware security primitive known as PUF. In the next chapter, we will be designing a flexible architecture for enabling a secure communication between lightweight devices and host processors using public-key based primitives and PUF technology as the root-of-trust. A prototype will be built as a proof-of-concept, and a recommendation on a high-level hardware design will be drafted.

# 4 Protocol Design

*Security becomes a major concern with the proliferation of lightweight IoT devices. In fact, we need to design architectures that are tailored for new applications and meet new and innovative security system requirements. In our case, it is establishing a secure connection between a resource-constrained device that has an embodied Physical Unclonable Function (PUF) and a resource-rich host processor. In this chapter we will propose a protocol that best solves the problem, and in addition propose several protocol variants that are better in certain aspects. In Section 4.1 we first reintroduce the use case application and proceed with defining a set of requirements for this project which are influenced by Intrinsic-ID and its vision on the industry. Section 4.2 demonstrates the procedure of selecting and achieving a protocol that fulfils the given set of requirements. Section 4.3 proposes several protocol variants and compares them.*

## 4.1 Use Case and Application

As described earlier in Section 1.2, the goal is to authenticate lightweight devices in the field and establish secure communication between them and a host computer. For that purpose, we need to design a secure solution that enables that. This includes designing a protocol as seen in this Chapter and designing a hardware architecture as we will see in the next Chapter. Below is a set of requirements that defines the scope of the project and serves as the guideline in designing such complex secure solution.

**Requirements**
In this section, we elaborate on the design requirements that allow us to make appropriate decisions throughout all stages of design i.e. when deriving the protocol and designing the hardware template.

(i) **Authenticate the Silicon**

The goal is to authenticate the device's silicon. Using Physical Unclonable Function (PUF) technology we are able to do just that. For that reason, a Static Random-Access Memory (SRAM) PUF-derived key must be used as the root-of-trust in our architecture. From the security perspective, this PUF-derived key is the primary asset in this system and must be well protected.

(ii) **Key-length Security Level**

The desired security level is 128-bits. Therefore, the cryptographic algorithms and protocols must be chosen such that it guarantees this security level. A higher security level is unnecessary.

(iii) **Self-Contained Solution**

It is required that all the necessary operations are done within our security boundary as much as possible, meaning that all the operations are done under our control. Therefore, the architecture must be self-contained. Reliance on the end user's compute resources must be minimal.

(iv) **Rely on best practice in Public-key Cryptography (PKC)**

An elliptic curve to be chosen must be an acceptable standard, meaning that is well-known and is well-researched, and that has low-overhead properties.

(v) **Side-channel Resistance**

In the course of design, all countermeasures against side-channel attacks are preferable as long as they do not significantly impact the size, but are not strictly required.

(vi) **Area constraints** Because the target application is a constrained lightweight Internet of Things (IoT) device, silicon area consideration is necessary. Ideally, the area needs to be as small as possible. A gate count of $\leq 10k$ gates TSMC 65nm GP is an acceptable target.

(vii) **Timing Requirements**

Timing requirements specify bounds under which a certain operation should be executed. There are no strict *timing* requirements for this project. Nevertheless, we wish not to have an authentication mechanism that takes infinitely long time. In general, the timing requirement is specific to end user's application. Important to note here is that authentication and key agreement happens only a few times, and therefore is not usually the bottleneck. Nevertheless, faster designs could potentially allow more applications: it is therefore in our interest to optimise for timing too.

(viii) **Technology Independent**

The end-product implementation ought to be a hardwired based solution that is technology independent. Therefore no device / technology specific optimisations are possible. Optimisations must be generic and done on the architectural and/or Register Transfer Level (RTL) level. Furthermore, the circuitry must be purely digital Integrated Circuit (IC) implementation using standard manufacturing process and utilizing standard components.

(ix) **Patent/License free**

Any component reuse (IP, source code etc.) must be in public domain which implies that there is no requirement of disclosure. Furthermore, technologies that are covered by known patents should be avoided in the design.

Some of the requirements above are rigid; others are more flexible. For example, in certain cases a slight increase in the area could be viable if the gain in speed is noticeably

bigger; or overall security is substantially increased. Therefore, various trade-offs such as this one is to be analysed during the design, and a balanced choice must be made to achieve an elegant solution that best satisfies the requirements.

Besides the requirements that are listed above, it is also important to consider the parties involved in the protocol, as well as the attack models in consideration. This plays a significant role in designing the protocol, which is the subject of the next section.

## 4.2 Protocol Design

Establishing a secure communication between a device and a host processor requires a secure protocol. Authentication and key establishment are the two key components that need to be achieved by the cryptographic protocol that we are going to design. Authentication is required for assuring that the identity of an IoT device that is trying to communicate to the host processor is legit. This can be achieved by using PUF technology and additionally through some sort of proof as we will see later. Note that malicious devices cannot be completely ruled out, because even an authenticated device can be malicious e.g., a device that was hacked. As one of the requirements, the PUF hardware security primitive must be used as the root-of-trust, which provides authentication of the silicon. Furthermore, to keep the data transmission confidential and accessible only by authorized parties, it is necessary that the communication is encrypted. To achieve this, symmetric cryptography can be used. However, as we know by now, symmetric cryptography requires that a copy of the same key is available on both ends of the communication channel, which can be achieved by a key-establishment protocol such as the one shown in Section 3.1.1. In the rest of the section, we show the steps in deriving a protocol that establishes a secure shared key between the IoT devices and the host processor, and authenticates device's silicon using PUF, therefore establishing a secure communication.

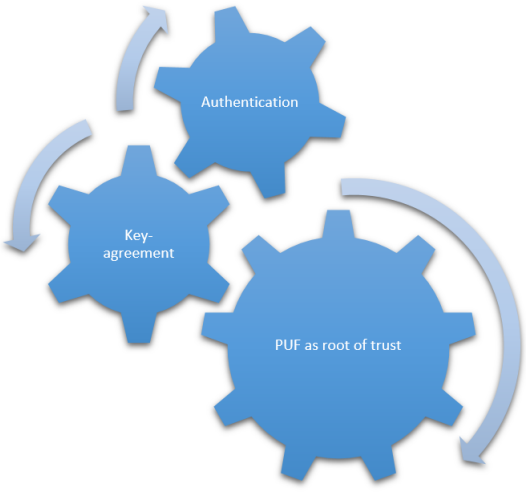



Figure 4.1: Key components for secure communication

### 4.2.1 Key Agreement

Earlier in Section 3.1, we showed that public-key primitives and their respective protocols are typically used for key-exchange purposes: both key transportation and key agreement methods have been introduced. We also argued that Elliptic Curve Cryptography (ECC), whose underlying computationally hard problem is the Elliptic Curve Discrete Logarithm Problem (ECDLP), is the choice for constrained devices at the moment of writing. The reason is due to its inherently shorter operand sizes; Furthermore, since ECDLP is just a generalisation of the Discrete Log Problem (DLP), all existing DLP protocols are suitable under ECC e.g. Elliptic Curve Diffie-Hellman (ECDH).

The ECDH Key Exchange is a key-agreement protocol that can be used for deriving a shared key. Although described in Section 3.1.3, shown again below in Protocol 4.1 for convenience. Let $\#E$ be the group order of $E$ (domain parameter) and $a, b$ the private keys of Alice and Bob respectively, that are chosen at random from a set i.e. $\{2, ..., \#E - 1\}$. Another domain parameter is a $P$, which is the *base point* for a given curve. Let $(X, Y)$ denote points on the given curve $E$. The output of the protocol is $T_{AB}$ which is the shared key. Unfortunately, that protocol has obvious shortcomings. Mainly, it lacks an authentication mechanism, and is therefore vulnerable to a multitude of attacks e.g. Man-In-The-Middle (MITM) attack on the key agreement as described in Section 3.1.1. As it was stated earlier, authentication is one of the key components in establishing a secure channel. Furthermore, according to the requirements, the PUF hardware security primitive must be used as the root-of-trust, for authenticating IoT device's silicon. Therefore, it is necessary that this protocol is modified to an extent that it satisfies the requirements described in Section 4.1. In the following subsection we will examine how the PUF security primitive can be used in our advantage, e.g. in a protocol for authentication.

**Elliptic Curve Diffie-Hellman Key Exchange**

| | **Alice** | | **Bob** |
|---|---|---|---|
| 1 : | *Choose* $a = k_{pr,A} \in_R \{2, ..., \#E - 1\}$ | | $b = k_{pr,B} \in_R \{2, ..., \#E - 1\}$ |
| 2 : | *Compute* $A = k_{pub,A} \equiv aP = (X_a, Y_a)$ | | $B = k_{pub,B} \equiv bP = (X_b, Y_b)$ |
| 3 : | | $A = k_{pub,A} \longrightarrow$ | |
| 4 : | | $\longleftarrow B = k_{pub,B}$ | |
| 5 : | $T_{AB} = aB$ | | $T_{AB} = bA$ |

..... Output: $T_{AB} = (X_{ab}, Y_{ab})$ .....

Protocol 4.1: Elliptic Curve Diffie-Hellman (ECDH) key-exchange protocol

### 4.2.2 Authentication using PUF

Let us first examine and understand how PUF technology can be used for authentication. We first examine an existing protocol that uses PUF that we could possibly improve on or extend for our purpose. One of the simple authentication methods we can use for IC authentication using PUF is described in [55] and is shown in Protocol 4.2. The protocol goes as follows: Before operation in the field, a number of Challenge-Response (CR) pairs associated with the device's PUF is recorded in a database by a Trusted Third Party (TTP)'s as shown in *(I) Enrollment of the sensor* stage of the protocol. In Step 1 the TTP chooses a random challenge and sends it to the sensor node. The sensor node processes the challenge using PUF and replies with a response as shown in Steps 2-4. In Step 5, the TTP stores this CR pair in the database. TTP can then repeat this procedure as much as necessary as shown in Step 6. Later in the field, the host computer (authenticator) can use one of the recorded challenges to challenge the device and compare the received response with the real recorded response in the database to authenticate the device as shown in *(II) Sensor Authentication* stage . First the host processor must choose a CR pair that corresponds to that particular device and send the challenge to the device as show in Steps 7-8. The sensor will process the challenge using the PUF and respond with a corresponding response as shown in Step 10. To decide whether the device is authentic or not, the host processor can compare the response to the reference response retrieved from the database, as shown in Step 11. This protocol is susceptible to a MITM replay attack, where the CR pairs can be captured and later reused for malicious purposes, therefore, [55] suggests that every pair is used at most once.

Although this protocol provides a good understanding of how a PUF primitive can be used for authentication, it clearly lacks a key-exchange mechanism if used independently, therefore is limited. At first it might seem that this protocol can be used in conjunction with an existing key-agreement protocol described earlier to achieve our goal: first a key-agreement using ECDH (Protocol 4.1) followed by the described CR PUF authentication (Protocol 4.2); however, this construction is vulnerable to a MITM attack on the key-agreement as described in Section 3.1.1. Adding the CR PUF protocol would therefore be useless: after successfully placing himself in the middle, the attacker would simply relay the challenges and the correct responses, hence jeopardizing the entire system, and making this step useless. There are several other disadvantages of Protocol 4.2:

- *Scalability:* This protocol does not scale well with the number of devices, since the database is required to store a large number of CR pairs: number of devices multiplied by $N$ as in Protocol 4.2. It is also limited because new enrollments must be done if CR pairs for a particular device are depleted.
- *Suitability:* The SRAM PUF does not have the capability of providing a large amount of CR pairs. The CR pairs depend on the size of available SRAM cells allocated specially for the purpose of PUF. Therefore, this protocol is not suitable for SRAM PUF.

In the following section we examine a protocol that uses a PUF-derived key along with the Diffie-Hellman Key Exchange (DHKE) to achieve our goal.

**(I) Enrollment of the sensor**

| **Trusted Third Party (TTP)** | | **Sensor(ID)** |
|---|---|---|
| 1 : *Choose* $C_{ID}$ | | |
| \| $C_{ID} \in_R$ {*set of all possible challenges*} | | |
| | 2 : *Challenge* $C_{ID}$ $\longrightarrow$ | |
| | | 3 : *PUF processing* $R_{ID} = f(C_{ID})$ |
| | $\longleftarrow$ 4 : *Response* $R_{ID}$ | |
| 5 : *Store* $(C_{ID}, R_{ID})$ *pair to a database* | | |
| 6 : *Repeat Steps* 1 *to* 5 $N$ *times* | | |

**(II) Sensor Authentication**

| **Host Processor (HP)** | | **Sensor(ID)** |
|---|---|---|
| 7 : *Choose* $(C_{ID}, R_{ID})$ *pair from database* | | |
| | 8 : $C_{ID}$ $\longrightarrow$ | |
| | | 9 : *PUF processing* $r_{ID} = f(C_{ID})$ |
| | $\longleftarrow$ 10 : $r_{ID}$ | |
| 11 : $Verify\ R_{ID} \stackrel{?}{=} r_{ID}$ | | |
| 12 : *Remove* $(C_{ID}, R_{ID})$ *pair from database* | | |

Protocol 4.2: Device authentication using PUF

### 4.2.3 Protocol for key-agreement and authentication using PUF-derived key

So far in this Chapter we have briefly shown a key-agreement protocol and a simple protocol that uses PUF CR pairs for authentication, both of which have certain disadvantages and therefore not useful for our special purpose application. As it was shown in Section 3.4.2 it is possible to derive a random, device unique, cryptographically secure key from an SRAM PUF. We propose to derive a protocol based on DHKE, that uses a PUF-derived key.

As we have seen in Section 3.4.2, one way to use PUF for key derivation is to use a code-offset method. Hence, the protocol is naturally divided into two stages: the *enrollment* stage, and the *key-agreement and authentication* stage. The first stage is used to record all the necessary data from device's PUF, that is used to generate certificates that bind those measurements to the identity, which is used in later stages of the protocol

for authentication. The second stage is when the device is in operation in the field. It is, therefore, important to identify the different assumptions and consider the attack models for each stage individually.

**General Assumptions**
The adversary can eavesdrop all the messages in the communication channels; *passive attacks* are possible. In passive attacks an intruder can only eavesdrop on the communication. Therefore, we shall not transmit any sensitive data along the communication channel in plaintext thus creating an unsecure system. In fact, this is the main motivation for that we need to establishing a *secure communication* in the first place.

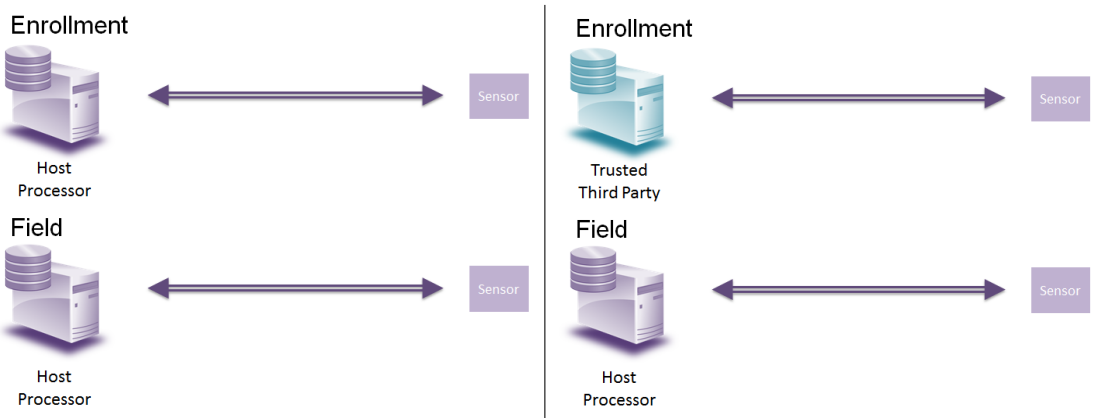
**Field Assumptions**
The field is untrusted, therefore it is assumed that both *active* and *passive* attacks are possible. In an active attack an intruder may transmit, replay, modify, or delete any selected messages in a communication.

**Enrollment Assumptions**
In this stage, the security assumptions are less stringent as compared to the field. We assume that active attacks are not possible during enrollment; an eavesdropper is not able to modify any messages, but one is capable of recording the messages in transit. The motive behind this assumption is that the device is enrolled in a controlled environment such as at the manufacturer's facilities.

The first step towards a solution is a protocol[1] that is shown in Protocol 4.3. In essence, the ECDH protocol is modified such that the PUF is used as the hardware root-of-trust. The PUF derived secret key on the device is used as the secret key in Diffie-Hellman key generation. Unfortunately, this particular protocol is limited because it does not use a Public Key Infrastructure (PKI) and therefore does not inherit all its benefits. In this scenario the key-agreement must be done at enrollment with a host processor as shown in in Figure 4.2(a). Only that same host processor is able to communicate with the sensor later in the field.

Using a PKI and a TTP as shown in Figure 4.2(b) adds substantial flexibility. It allows all devices to be enrolled by one single TTP at a very early stage such as at the factory or during chip test. A modified protocol that uses a TTP is shown in Protocol 4.4. In that protocol, during the enrollment phase, the necessary components for key reconstruction in the field at a later time, such as the Activation Code along with the device's ID and the newly generated public key, is sent to the TTP for signing and certificate generation. Later in the field, the authenticator is able to verify if the certificate belongs to a device that has been previously enrolled, by inspecting the received certificate. If the verification is successful, both parties proceed with ECDH for key-agreement, and finalize it with an authentication step. The following is a detailed explanation of the protocol.

[1]Relevant notations are described on Page 51.




(a) Scenario 1 (b) Scenario 2

Figure 4.2: Different Scenarios

**(I) Enrollment of the sensor**

| **Host Processor (HP)** | | **Sensor (ID)** |
|---|---|---|
| 1 : $(SK_{HP}, PK_{HP}) \leftarrow DH$ KGen | | 2 : $x,\ AC \leftarrow PUF - enroll$ |
| | | 3 : $PK_{ID} \leftarrow DH$ KGen$(x)$ |
| | 4 : $ID, AC, PK_{ID} \longleftarrow$ | |
| | $\longrightarrow$ 5 : $PK_{HP}$ | |
| 6 : $Store\ (ID, AC, PK_{ID})$ | | 7 : $Store\ (PK_{HP})$ |
| ..... Sensor enrollment complete ..... | | |

**(II) Key-agreement**

| **Host Processor (HP)** | | **Sensor (ID)** |
|---|---|---|
| | 8 : $Session\ request,\ AC \longrightarrow$ | |
| | | 9 : $x \leftarrow PUF - reconstruct$ |
| 10 : $w = SK_{HP} * PK_{ID}$ | | 11 : $w = x * PK_{HP}$ |
| ..... Output: Shared secret $w$ ..... | | |

Protocol 4.3: Diffie-Hellman Key Exchange (DHKE) using Physical Unclonable Function (PUF)-derived key

**Notation**
Notations used in Protocols 4.3 through 4.7:

- *PUF-enroll:* generates a (PUF-derived) cryptographically secure key $x$ and an Activation Code (Helper Data) $AC$ that is used later in the decoding stage of the key reconstruction as described in Section 3.4.2.
- *PUF-reconstruct:* is the reconstruction process of the secure key that was enrolled earlier as described in Section 3.4.2.
- *DH KGen(x):* calculates the public key based on the private key $x$. In ECC it corresponds to a scalar multiplication of the scalar $x$ and the base point of a particular elliptic curve as per curve's specifications.
- *Ephemeral DH KGen:* this function generates an ephemeral key-pair suitable for a chosen elliptic curve.
- *KDF():* is a Key Derivation Function that improves the quality of the key.
- $E_k$, $D_k$: are symmetric encryption and decryption functions respectively, using key $k$.
- *ID:* is an identification number unique to a device.

**Protocol 4.4 Detailed Explanation**
The protocol is divided into three stages. In the first stage, the device is enrolled by a TTP in a relatively secure environment. In the second and third stages, key-agreement and authentication respectively, happen in the field. Individual stages are described in more detail below:

**(I) Enrollment** All sensor nodes[2] must first be enrolled with a TTP before their operation in the field. The enrollment phase happens as follows: In Step 1, the sensor's PUF is enrolled, and the key-generation subsystem generates a cryptographic key $x$. In Step 2, the generated key $x$ is used as the private key component in calculating the corresponding public key. This computation is done using *scalar multiplication* in a suitable elliptic curve group, the result of which is a point on the curve. In Step 3, the sensor sends its identifier *ID*, Activation Code (AC) and the computed public key to the TTP. The AC is required for the reconstruction of the private key $x$ from the PUF in the later stage. The TTP signs the received data using its private key and generates a certificate as shown in Steps 4 and 5. The certificate binds the sensor's ID to its PUF based public key, and the AC needed to generate it. The signed certificate is sent back to the sensor and is stored in the Non-volatile Memory (NVM) of the sensor, as shown in Steps 6 and 7 respectively.

[2]Note that *sensor* here refers to a constrained IoT device.

**(II) Key-agreement** In the field, the sensor must be able to establish an authenticated secure communication with the host processor. In this variant of the protocol, the host processor initiates the protocol by sending a *session request* message as shown in Step 8. In Step 9, the sensor replies with the certificate that has been stored previously during the enrollment phase. Because the Host Processor possesses TTP's public key, it can verify the certificate; meanwhile, the sensor can engage in the reconstruction of the PUF-based private key that corresponds to the certificate. After successful verification, the Host Processor chooses a random secret key and calculates the corresponding public key in Step 12. In Step 13, the Host Processor transmits its computed public key (its Diffie-Hellman contribution) to the sensor. Now that both parties involved have all the required contributions, they can compute their shared secret $w$ as shown in Steps 14 and 15. In Step 16 a *Key Derivation Function* is used for privacy enhancing purposes for deriving a cryptographically secure session key.

**(III) Authentication** Until this point any sensor will succeed. Both parties will calculate $k$ as in Step 16. But will those calculated keys be the same on both sides of communication? Only legit and authenticated devices will have the expected keys on both sides. It is therefore paramount to check that the sensor has indeed the correct session key. The authentication step is necessary to weed out unauthenticated devices, and deny any further communication. There are multiple ways of how this can be achieved, one of which could be done by using a simple challenge-response based authentication protocol based on the derived key. The host processor creates a challenge by randomly picking a number from a predefined set and encrypting it with the derived key $k$ and sends this challenge to the sensor as shown in Steps 17-19. The sensor decrypts the message, increments the number and encrypts it back using the same session key $k$ and sends it back to the host processor as shown in Steps 20-23. The host processor decrypts the response and verifies if it is the expected incremented value as shown in Steps 24 and 25. Obviously, the verification will succeed if and only if the sensor was able to correctly decrypt the initial challenge and therefore properly increment the value.

**Discussion**

The above authentication protocol (Stage III) is a simple challenge-response based authentication. Alternatively, Message Authentication Code (MAC) can be used. It is important to note that this is a *one-way authentication*. Only the host-processor can be sure that the sensor is legit. There is no way for the sensor to be sure about the authenticity of the host processor using this protocol. That said, from the sensor's point of view, a MITM attack cannot be ruled out. A malicious host can initiate a communication with such sensor. In the following section we propose protocol variants that mitigates this issue, and moreover, introduces other advantages.

**(I) Enrollment of the sensor**

| **Trusted Third Party (TTP)** *has* $SK_{TTP}$ | | **Sensor (ID)** |
|---|---|---|
| | | 1 : $(x,\ AC) \leftarrow PUF - enroll$ |
| | | 2 : $PK_{ID} \leftarrow\ DH\ \mathsf{KGen}(x)$ |
| | 3 : $ID, AC, PK_{ID}$ $\longleftarrow$ | |
| 4 : $\sigma_{ID} \leftarrow \mathsf{Sig}_{\mathsf{sk}_{TTP}}(ID, AC, PK_{ID})$ | | |
| 5 : $Cert_{ID} = [ID, AC, PK_{ID}, \sigma_{ID}]$ | | |
| | 6 : $Cert_{ID}$ $\longrightarrow$ | |
| | | 7 : $Store(Cert_{ID})$ |

Sensor enrollment complete

**(II) Key-agreement**

| **Host Processor (HP)** *has* $PK_{TTP}$ | | **Sensor (ID)** |
|---|---|---|
| | 8 : *Session request* $\longrightarrow$ | |
| | 9 : $Cert_{ID} = [ID, AC, PK_{ID}, \sigma_{ID}]$ $\longleftarrow$ | |
| 10 : $\mathsf{Vf}_{\mathsf{pk}_{TTP}}(Cert_{ID})$ | | 11 : $x\ \leftarrow PUF - reconstruct(AC)$ |
| 12 : $(SK_{HP}, PK_{HP}) \leftarrow Ephemeral\ DH\ \mathsf{KGen}$ | | |
| | 13 : $PK_{HP}$ $\longrightarrow$ | |
| 14 : $w = SK_{HP} * PK_{ID}$ | | 15 : $w = x * PK_{HP}$ |
| 16 : $\mathsf{k}\ = KDF(w)$ | | $\mathsf{k}\ = KDF(w)$ |

Key Agreement complete

**(III) Authentication**

| | | |
|---|---|---|
| 17 : *Choose* $N\ \mid\ N \in_R \{Set\ of\ integers\}$ | | |
| 18 : *Encrypt* $N$ : $C = E_k(N)$ | | |
| | 19 : $C$ $\longrightarrow$ | |
| | | 20 : *Decrypt* $C$ : $n = D_k(C)$ |
| | | 21 : $n := n + 1$ |
| | | 22 : *Encrypt* $n$ : $R = E_k(n)$ |
| | 23 : $R$ $\longleftarrow$ | |
| 24 : *Decrypt* $R$ : $n = D_k(R)$ | | |
| 25 : $Verify\ n \stackrel{?}{=} N + 1$ | | |

Authentication complete

Protocol 4.4: Diffie-Hellman Key Exchange (DHKE) using Physical Unclonable Function (PUF)-derived key and a Trusted Third Party (TTP) (Variant A)

## 4.3 Protocol Variants

In the previous section we show how a suitable protocol is derived. The derived protocol tackles the challenges discussed earlier in Chapter 1 while satisfying the requirements discussed in Section 4.1 at the same time. Although the proposed Protocol 4.4, which will be referred to as 'Variant A' from here on, is satisfactory, it can be modified and improved in terms of security and area requirements. The improvement could be more fit for certain use cases as we will see later. In this section, we propose and examine three protocol variants. At the end, a summary is provided that highlights their distinguishing characteristics along with some measurements. In short, the all protocol variants are distinguished by the following properties:

- *Variant A:* This is the originally proposed protocol that achieves only a one-way authentication.
- *Variant B:* This protocol is modified in such way that a mutual-authentication is achieved. This is done by enrolling and issuing a certificate to a Host Processor that can be verified by the IoT device in the field.
- *Variant C:* This variant adds a cloud infrastructure, therefore reducing the NVM requirements on the device and furthermore improves certificate management.
- *Variant D:* This variant combines variant B and C.

Similarly to Variant A, all variants are divided into three stages. In the first stage *(I) Enrollment*, the devices are enrolled by a TTP, which is typically done only once during the life-cycle of a device. The second *(II) Key-agreement* and third *(III) Authentication* stages are performed in the field whenever necessary: communicating parties establish a shared key, followed by an authentication step. Since the following protocol variants are all similar to Variant A, we describe only the important differences for the sake of brevity. Note that the notations used in the following protocols are similar to those described on Page 51, therefore not repeated here.

**Variant B Explanation** *Protocol 4.5 as shown on Page 59*
In Variant A, only the sensor node is enrolled, therefore it is capable of offering only a one-way authentication. The host processor is confident in the sensor's authenticity, but not vice versa. In many use cases we ought to have a mutual authentication. In fact, one can argue that sending data to the 'right' server is more important. For example, privacy issues can be avoided in mutually authenticated communications. Clearly, not everyone on the Internet should be able to 'securely' access ones GPS sensor. Only those intended, should be able to.

Fortunately with slight modifications in Variant B we achieve mutual authentication. To achieve this, the host processor needs to be enrolled (issued a certificate) by a TTP similarly as the sensor, shown in Steps 8-13 of this Protocol. The sensor must verify the identity of the HP in the field by verifying its certificate, therefore an additional step in sensor enrollment is storing the TTP's public key $PK_{TTP}$ on the sensor. In the field, the host processor sends its signed certificate to the sensor for verification. Only

after a successful signature verification, the communicating parties can proceed to the usual ECDH for key agreement. Followed by an authentication stage, however this time, challenges going in both directions.

Note that the signature scheme can be implemented using Elliptic Curve Digital Signature Algorithm (ECDSA). A verification step is considered to be computationally intensive because it requires two scalar multiplications of the form $k_1xP + x_2xQ$ [35]. This is known as multi-scalar multiplication. There exist tricks that accelerate this procedure such as in [56].

**Variant C Explanation** *Protocol 4.6 as shown on Page 60*
In Variant A, the sensor node is required to store a certificate, that can take up a 'significant' amount of space on a constrained device. In certain cases, reducing the NVM or completely eliminating our dependency on it can bring tremendous benefits. More over it is one of the requirements to reduce silicon area.

This proposed protocol, Variant C, requires zero NVM storage on the sensor by leveraging a *cloud infrastructure*. The notable differences during the enrollment phase is that the sensor does not store any information locally. Instead, the TTP publishes the sensor's certificate to a cloud as shown in Step 6. In the field, the host processor retrieves the certificate from the cloud and verifies it as shown in Steps 7 and 9 respectively, and proceeds with PUF-based ECDH. Note that because of the addition of the online cloud infrastructure, we get additional benefits such as an easier *certificate management* e.g. sensors can be easily blacklisted if necessary.

**Variant D Explanation** *Protocol 4.7 as shown on Page 61*
Thus far, two variants have been proposed that independently achieve a mutual authentication and zero NVM storage on sensor, along with their advantages.

It is worth examining a variant that can do both. Basically, by merging the two, i.e. enrolling the host processor and adding a cloud infrastructure, we achieve this Variant D that has a mutual authentication, very little NVM requirement, and adds the certificate management flexibility via the cloud infrastructure.

**Discussion**
The four protocol variants (A, B, C, and D) have been proposed so far that use ECDH and that are based on PUF-derived keys. Although the differences between the protocol variants are simply slight changes, the results e.g. communication performance, functionality etc. can be significantly different as we will see. Table 4.1 shows basic measurements such as the number of transfers, data transfer size, and NVM requirement. Note that the estimations are chosen to be based on realistic values. The size of ID is chosen to be identical to the size of a Media Access Control address[3] which is 48 bits. The size of AC is based on the latest Intrinsic-ID Quiddikey technology[4]. A key with 256 bits of entropy needs approximately 720 bytes of SRAM and 752 bytes AC. Moreover,

[3]IEEE Standard.
[4]https://www.intrinsic-id.com/products/quiddikey/

the following analysis is focused more on the sensor side and the interactions with the sensor due to its constrained nature.

The *Number of Transfers* shows the number of message transactions during enrollment and reconstruction. As we can see, Variant B has the most data transfers, whereas Variant C has the least. The *Data Transfer Size* shows size of the messages in terms of the number of bits needed to be communicated during the transactions. This metric is especially important for the constrained devices because every bit sent is counted towards consumed power. Clearly, Variant B transmits the most amount of bits back and forth with the sensor, whereas Variant C the least. The *NVM Requirement* shows how much data need to be 'permanently' stored on the sensor. Variant B requires the most since it needs to store the certificate as well as the TTP's public key, whereas Variant C requires no storage. The main properties of the protocol variants A-D are summarized in Table 4.3. Options/properties with (+) are welcome whereas (-) are not so welcome.

Additionally, Figure 4.4 visually compares the different protocol variants. We have chosen to compare the protocol variants based on the following four dimensions:

- *Communication Performance:* This directly reflects the transfer size in bits. Less data transferred we consider as better performance.
- *NVM Dependency:* The amount of NVM storage required on the sensor. Shown inversely on the plot: the highest point on the curve corresponds to zero storage requirement on the device.
- *Certification Management:* This is a binary value. It is either online (end of axis) or offline (middle of the axis).
- *Authentication:* This is a binary value. Middle of axis corresponds to one-way, whereas the peak corresponds to a mutual authentication.

All values have been normalized and scaled accordingly, origin represents worse, whereas perimeter is better. Therefore, a bigger area of the spiderweb represents a better protocol variant in terms of these dimensions. In the next section, we discuss the hardware architecture that is required on the sensor side to support one of these protocol variants.

| | **Variant A** | **Variant B** | **Variant C** | **Variant D** |
|---|---|---|---|---|
| Number of Transfers | | | | |
| Enrollment | 2 | 4 | 1 | 4 |
| Reconstruction | 5 | 5 | 4 | 4 |
| *Total:* | *7* | *9* | *5* | *8* |
| Data Transfer Size in bits (sensor) | | | | |
| Enrollment | 12896 | 13152 | 6320 | 6576 |
| Reconstruction | 6928 | 13312 | 6368 | 6736 |
| *Total:* | *19824* | *26464* | *12688* | *13312* |
| NVM Requirement (sensor) | 6576 | 6832 | 0 | 256 |

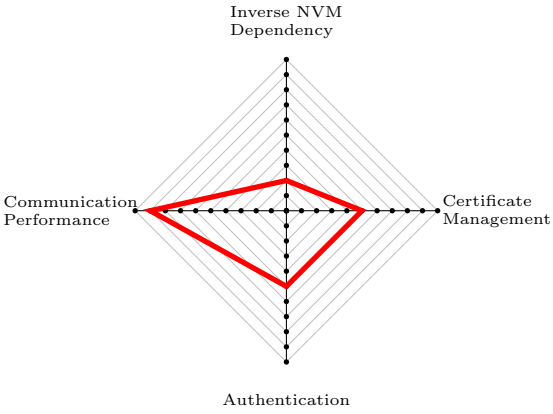
Table 4.1: Properties of Protocol Variants A-D

| **Packet** | **Size (bits)** | **Variant A** | | **Variant B** | | **Variant C** | | **Variant D** | |
|---|---|---|---|---|---|---|---|---|---|
| | | Sent | Total | Sent | Total | Sent | Total | Sent | Total |
| *Enrollment:* | | | | | | | | | |
| ID | 48 | 2 | 96 | 2 | 96 | 1 | 48 | 1 | 48 |
| AC | 6016 | 2 | 12032 | 2 | 12032 | 1 | 6016 | 1 | 6016 |
| PK | 256 | 2 | 512 | 3 | 768 | 1 | 256 | 2 | 512 |
| Sig | 256 | 1 | 256 | 1 | 256 | 0 | 0 | 0 | 0 |
| *Total:* | | | *12896* | | *13152* | | *6320* | | *6576* |
| *Reconstruction:* | | | | | | | | | |
| syn | *32* | 1 | 32 | 1 | 32 | 1 | 32 | 1 | 32 |
| ID | 48 | 1 | 48 | 2 | 96 | 0 | 0 | 1 | 48 |
| AC | 6016 | 1 | 6016 | 2 | 12032 | 1 | 6016 | 1 | 6016 |
| PK | 256 | 2 | 512 | 2 | 512 | 1 | 256 | 1 | 256 |
| Sig | 256 | 1 | 256 | 2 | 512 | 0 | 0 | 1 | 256 |
| C, R | 32 | 2 | 64 | 4 | 128 | 2 | 64 | 4 | 128 |
| *Total:* | | | 6928 | | 13312 | | 6368 | | 6736 |

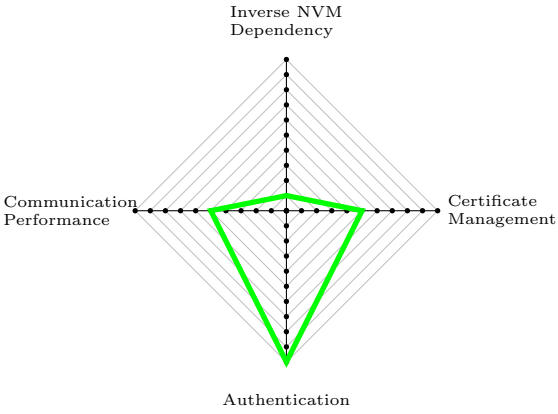
Table 4.2: Communication Analysis (sensor) of Protocol Variants A-D

| Protocol | Sensor Authentication | HP Authentication | NVM Requirement | Cloud Infrastructure | Certificate Management | Sig. Verification on Sensor |
|---|---|---|---|---|---|---|
| Variant A | + | - | large (–) | | offline (+/-) | |
| Variant B | + | + | large (–) | | offline (+/-) | required (–) |
| Variant C | + | - | none (++) | required (+/-) | online (+) | |
| Variant D | + | + | negligible (+) | required (+/-) | online (+) | required (–) |

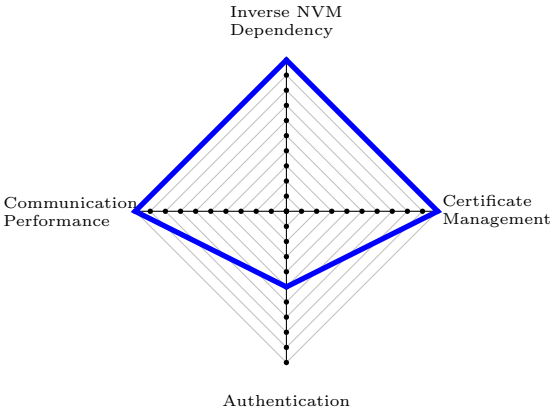
Table 4.3: Distinguishing Properties of Protocol Variants A-D



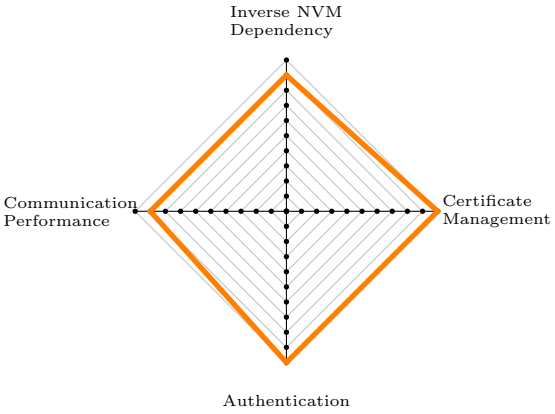

(a) Variant A




(b) Variant B

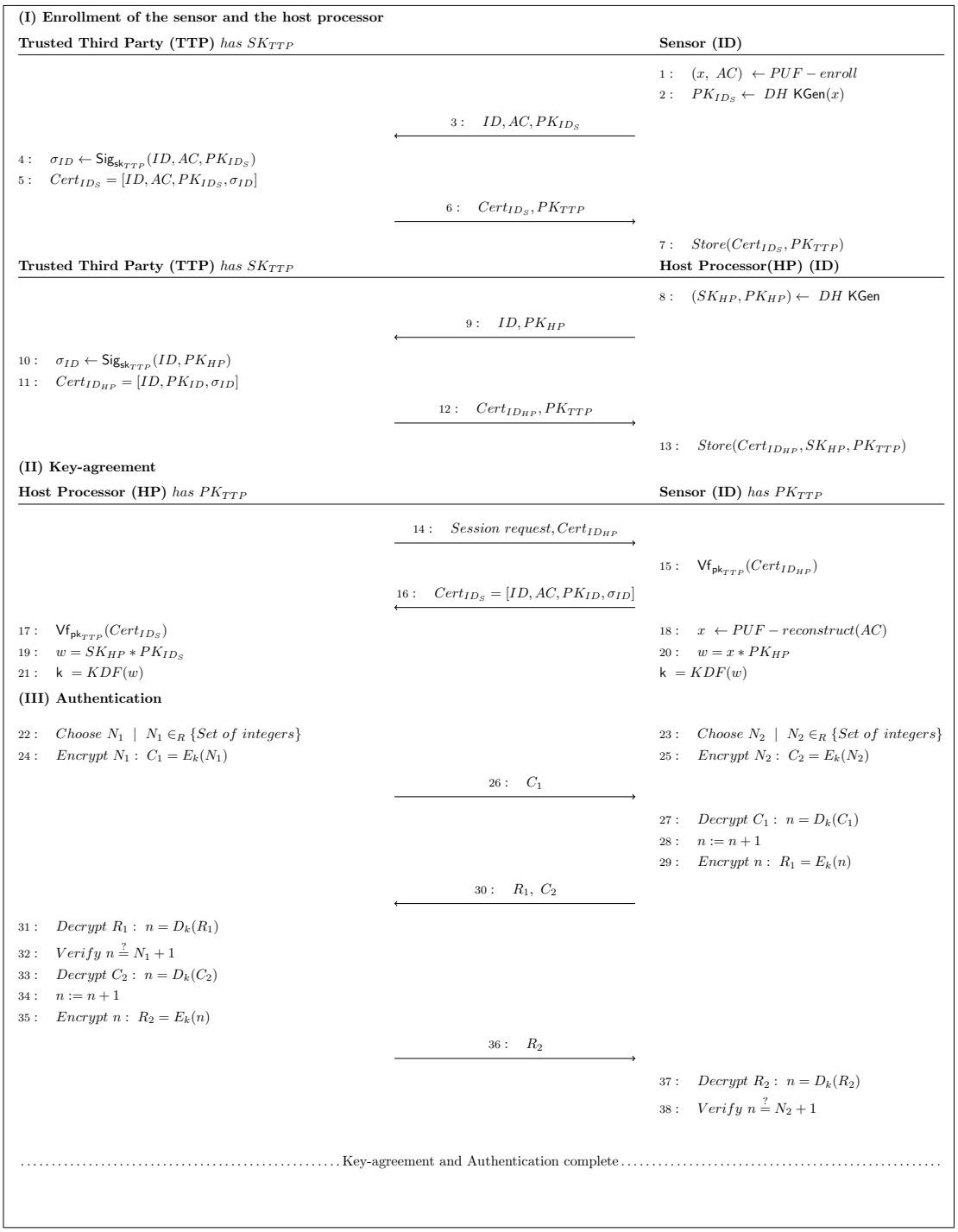



(c) Variant C

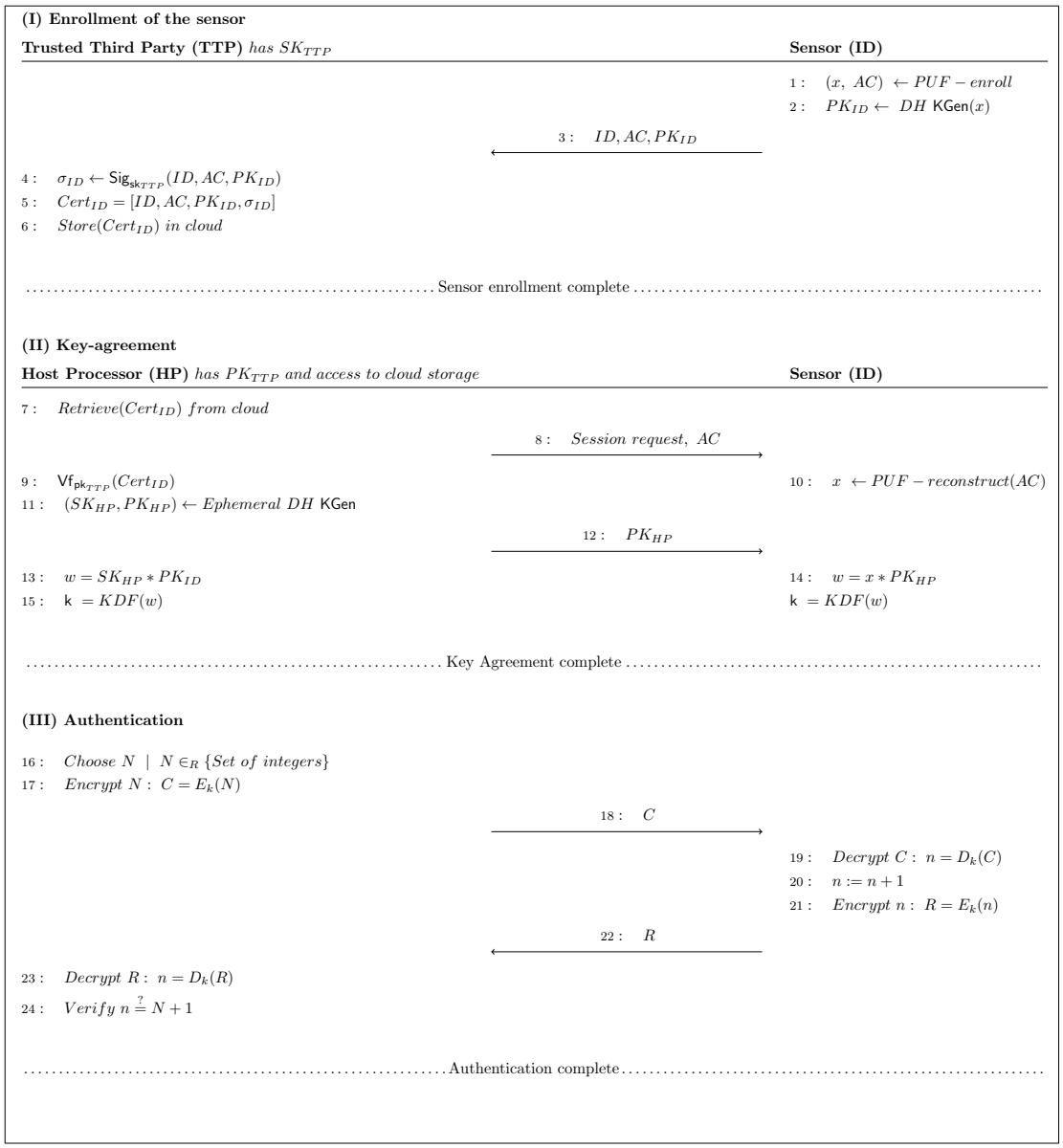



(d) Variant D

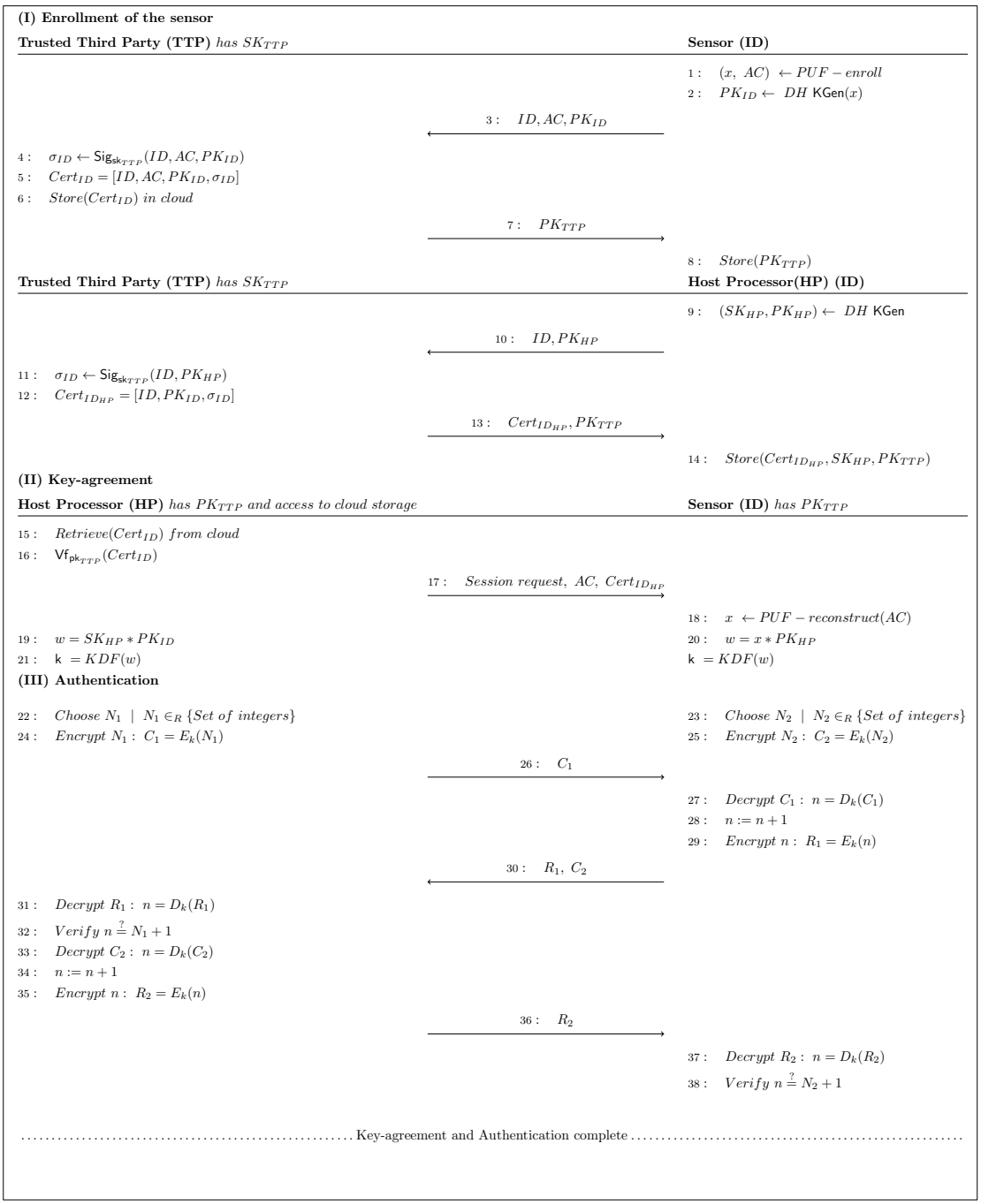
Figure 4.4: Comparison of Protocol Variants A-D in terms of Communication Performance, NVM Dependency, Certificate Management and Authentication




Protocol 4.5: Mutual Authentication Diffie-Hellman Key Exchange (DHKE) using Physical Unclonable Function (PUF)-derived key and a Trusted Third Party (TTP)(Variant B)




Protocol 4.6: Cloud storage Diffie-Hellman Key Exchange (DHKE) using Physical Unclonable Function (PUF)-derived key and a Trusted Third Party (TTP) (Variant C)




Protocol 4.7: Mutual Authentication Cloud storage Diffie-Hellman Key Exchange (DHKE) using Physical Unclonable Function (PUF)-derived key and a Trusted Third Party (TTP) (Variant D)

# Hardware Design and Validation

# 5

*In the previous chapter we proposed several protocol variants that enable establishing a secure connection between a resource-constrained device with an embodied Physical Unclonable Function (PUF) and a resource-rich host processor. In this chapter, we first investigate the key components that are necessary for enabling such security system, based on a proposed protocol. Then, we design and develop an application-specific architecture, a hardware template, that can be used as a guide in implementing such systems for IoT devices. In Section 5.1, we present the system level approach for this design. In Section 5.2, we show a proof-of-concept realisation of an instance of the proposed hardware template on a Xilinx Zynq-7000 family APSoC.*

## 5.1 Design and Development of the Architecture

In this section, we design and propose a hardware template of an architecture that adds secure system capability to an IoT device. Previously we have seen that to establish a secure connection with the constrained IoT devices in an untrusted environment we need to *(1)* authenticate them and *(2)* perform a key-exchange operation with the host processor. A key-exchange protocol based on Elliptic Curve Diffie-Hellman (ECDH) and Physical Unclonable Function (PUF)-derived key has been proposed in Section 4.2.3 along with additional protocol variants as can be seen in Section 4.3. PUF technology is used to authenticate device's silicon. Elliptic Curve Cryptography (ECC) is chosen due to its suitability for lightweight devices. These choices satisfy the requirements imposed earlier in Section 4.1.

By inspecting these protocols, we can identify the key components that are necessary for this secure application, and therefore we can design a high-level hardware architecture. Next we discuss the high-level system architecture and the components it comprises of, followed by a discussion on the various design choices that can be made.

### 5.1.1 High-level system architecture

Based on the Protocol 4.4, the constrained device must host a minimal set of primitives as elaborated below and shown in Figure 5.1:

- **Scalar Multiplication unit:** The protocols are based on ECDH; by inspecting the protocols, we can identify that the device performs scalar multiplication operation during enrollment as well as later in the field during the key-agreement stage of the protocol. As it was mentioned earlier in Section 3.2, scalar multiplication is the most compute intensive operation in ECC. Furthermore, it is the most critical component in terms of security. Therefore, its implementation in hardware, within our *security boundary/perimeter* is vital. This is discussed later in the section.

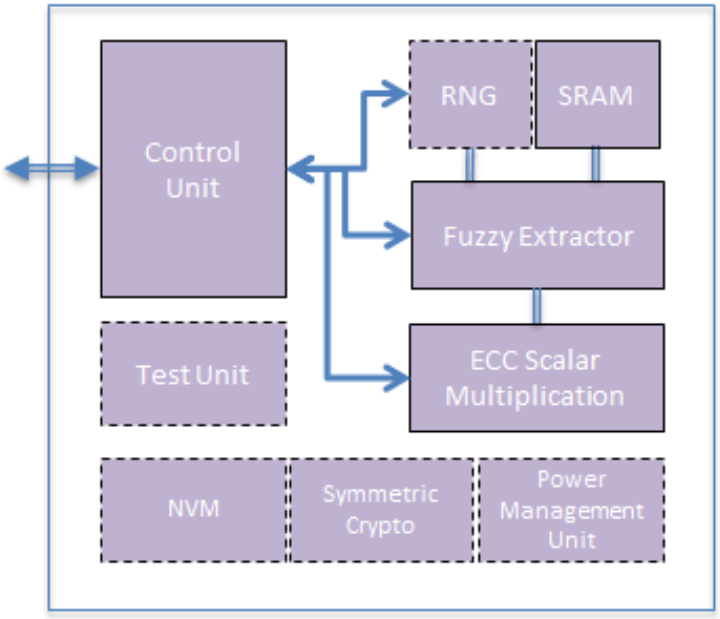



Figure 5.1: Conceptual Hardware Architecture

- **PUF System:** One of the objectives in this thesis was to use PUF-technology for silicon authentication. Therefore, the protocols designed in the previous chapter are all based on PUF-derived keys. In that regard, a PUF system must be present in the system. The choice of PUF is Static Random-Access Memory (SRAM) PUF as motivated earlier in Section 3.4.2. The following are the integral parts of the SRAM PUF system:
  - **SRAM:** Since we are focused on SRAM PUF, a block of uninitialized SRAM must be available in the system.
  - **Random Number Generator (RNG):** During PUF enrollment stage a key must be supplied for the purpose of *key-programming*. One such way can be by using an RNG, which indirectly can be a source of such key. Interestingly, a noisy PUF can be a good source of entropy when seeding an RNG [57].
  - **Fuzzy Extractor:** As shown in Section 3.4.2, SRAM start-up values are usually noisy, mostly due to environmental factors e.g. temparature variation. Therefore, a fuzzy extractor is needed to compensate for the noise present in SRAM start-up values. Only after that, a reliable and stable secure key can be reconstructed.
- **Control unit:** A control unit is essential for orchestrating all components, and interfacing with the outer world. This can be a microprogram for implementing the protocol, as well as an interface to the outside e.g. AMBA bus interface.

Optionally the final design might or might not include additional modules such as a

*test unit*, *Non-volatile Memory (NVM)*, *Symmetric crypto unit*, and a *Power management unit*. These modules are discussed next in more detail.

(i) **Test unit:** Testability is essential when making a system. Because we use a secure application, there are specific implications on the test unit. For example, a typical test solution scan-based design for test(DfT) *scanchain* is not designed for security. Yang et al. propose a secure scan method that does not compromise the security, while still maintaining a high test quality similar to the traditional scan DfT [58]. Testability is outside the scope of this thesis project.

(ii) **NVM**: Protocol 4.4 and Protocol 4.5 require data to be stored on the device. This NVM storage can either be part of the architecture or provided externally by the user.

(iii) **Symmetric Crypto Unit:** A symmetric crypto unit such as Advanced Encryption Standard (AES) can be added. Adding this block to the architecture will allow the user to accelerate the symmetric encryption/decryption process with the *shared key* that is produced by the protocol. It can also be used as the *KDF* as used in Protocol 4.4 and its variants.

(iv) **Power Management Unit:**

Most (or even all) of the components in the system described here, are only used occasionally, that is, during *enrollment* and *key agreement* during session establishment. Therefore, it is possible to power down the circuitry when the system is not needed. Depending on the final realization product, other techniques such as *power gating* [59] or *clock gating* [60] can be implemented for low-power applications. Furthermore, we should also be able to power down the SRAM. In fact, since we are interested in the PUF's power-up behaviour, it might be very beneficial have this functionality in the first place. Details of power management are outside the scope of this project.

### 5.1.2 Design Space Exploration

The main challenge in designing a secure system, as described in the previous section, is the large design space due to the plethora of possible options. It is paramount to pick the right sub-components/sub-systems and combine them in a way that best satisfies the requirements, and therefore producing an efficient solution. There exist different design paradigms that we can employ, this is discussed next.

Certain components such as SRAM can only be realised in hardware. However, for example scalar multiplication and the fuzzy extractor units can be designed in various ways. The architecture of these individual components can be based on different paradigms i.e., *hardware design*, *software design*, and *hardware/software co-design*. These are described below.

1 *Hardware design:* This is usually a highly optimised application-specific architecture. An example of such design is a finite-state machine implementation of a certain algorithm/datapath. The main advantage of a hardware design is high

performance. The downside is that that the architecture loses flexibility and does not allow for easy modification.

2 *Software design:* This is an implementation, where the software application is executed on a General Purpose Processor (GPP). Contrary to hardware design, software design is very flexible. A GPP can execute a variety of applications, which can be easily modified, updated, etc. The downside however is worse performance, lower silicon area efficiency, and higher power usage.

3 *Hardware/Software Co-design:* A Hardware/Software co-design paradigm involves the right allocation of functionality between software and hardware designs. In essence, it is an alternative that benefits from both paradigms described above. One example that employs such paradigm is an Application-specific Instruction Set Processor (ASIP). An ASIP is designed with an intention that it is a purposely crafted processor that has a clever thought-of Instruction Set Architecture (ISA) that is most suitable for the application.

Note that the above are merely architectural paradigms, and must be kept separate from and not confused with *hardware realisation* paradigms. The above architectures can be realized as an Application Specific Integrated Circuit (ASIC), on an Field Programmable Gate Array (FPGA), or even on a combination of both [61]. Typically, one realization is chosen over the other based on the ultimate goals. For a lightweight solution with the smallest form factor, and better speeds an ASIC realisation is preferred. This option, however, is usually economically feasible only if the volume of production is high, due to the manufacturing technology we have today. Naturally, high volume production also minimises the per-unit costs. On the other hand, for prototyping and for a low volume production an FPGA is usually a better and and a popular option. Moreover, since FPGAs are of-the-shelf products, deploying them provides a fast time to market.

Certain modules such as the scalar multiplication unit and the fuzzy extractor unit can be implemented in the various paradigms discussed earlier: hardware, software, or co-design. With a complete hardware solution, we might be able to achieve the smallest silicon footprint and relatively better performance. However, in a co-design paradigm, we can still do well regarding the area, and in addition, have a flexible solution, which is desirable. One way is to use an ASIP. An ASIP can be compared to a GPP but is tailored for a particular application. It can have a special ISA with dedicated instructions and optimised accelerators for certain specific tasks, e.g., big-integer multiplication with modular reduction.

Now that we have discussed the different paradigms individual components can be designed in, we are going to explore what is known as the *security boundary or the perimeter.* It is an important aspect to focus on, since it plays a crucial role in the system's security, and that is also very related to the *self-contained solution* requirement. Security boundary dictates which components are included in our hardware template, or otherwise shared with external or third-party resources. From a security standpoint, a design, where all components are within the secure boundary, is best. The reason is

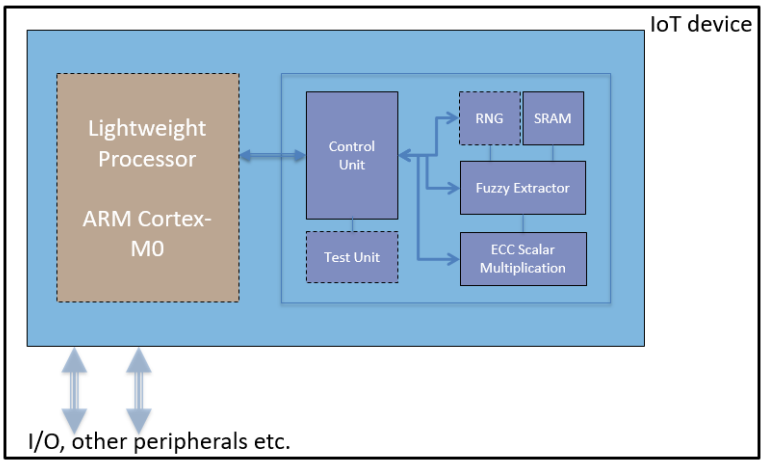



Figure 5.2: Conceptual IoT device for a secure application

that components within our boundary are also within our full control. This is the way to achieve maximal security. But once resources are shared, control becomes shared and typically having things under control becomes complicated. On the other hand, sharing resources can have advantages; for example, possibly a better hardware utilization, therefore decreased hardware area.

In this regard, we will closely look at SRAM and RNG modules, and discuss the consequences of them being within or outside the boundary. In order to get a better understanding we can examine a possible attack and its consequences. For example, altering or reading SRAM start-up values is clearly unsecure. Altering the start-up values can be seen as artificially injecting noise. The fuzzy-extractor will only be able to compensate for that noise to a certain extent, after which a stable and reliable key would not be reconstructed. Reading the SRAM values poses a security risk. However, if one can guarantee that an 'outsourced' SRAM module will not be tampered or read-out by unauthorized people, then it can be a suitable solution. As mentioned earlier, the SRAM can either be included within our architecture or provided/shared from outside. Although the security issue is evident here, relying on an externally provided SRAM values can possibly be a better option due to area savings. For the purpose of PUF, we are only interested in the uninitialized start-up values. Therefore, another benefit is that memory can be reused for other purposes after the initial values are read out. Similarly, the RNG is used only once during enrollment. Therefore, an externally provided random value under a controlled environment can be a possible solution as well, but a less secure one, due to the increase of the attack surface. Unfortunately, as we move our components outside the 'perimeter', we lose control over them, and trust that other parties will take care of the security at that level. In practice users tend to ignore security measures, or disable them altogether.

Let us examine a conceptual design of an IoT device, which can resemble something as shown in Figure 5.2. Its system's architecture constitutes of a lightweight processor such

as an ARM Cortex-M0 (M0)[1] augmented with a *secure core* that we aim to design and develop in this chapter. The most unsecure option perhaps would be to share as much resources as possible and even offload most of the operations onto the host processor. Besides the fact that this option is unsecure, this option has several other disadvantages and clearly violates some of the requirements set earlier. First of all, it contradicts the notion of having a self-contained solution, which is one of the requirements. Furthermore, an IoT device does not require to have a processor by definition i.e., it might comprise of only a simple circuitry that does not even need a processor as lightweight as ARM Cortex-M0. Therefore, proposing a design that is tailored for ARM Cortex-M0 would make the solution hardly portable. A similar but an alternative solution is to place a lightweight processor in our architecture and execute all the security related operations on that processor. In general we avoid using a GPP altogether in this project, because it does not provide an efficient use of the area.

## 5.2 Implementation and Evaluation

In Section 4.2, we demonstrated an ECDH-based protocol that use PUF-derived keys for hardware root-of-trust, and later proposed protocol variants that can be considered as improvements. In Section 5.1 we identified the key components that are needed to enable the protocol, and examined the design choices that can be made. Then, we proceed to the development of a proof-of-concept prototype system, where we aim to validate the idea, get more insights into the prototype, further explore the design possibilities and eventually make suggestions for future improvement. Finally, we build a prototype system based on the hardware template described earlier in Figure 5.1, to realize, validate and evaluate this secure solution.

### 5.2.1 Building a Prototype

The purpose of building a prototype is to demonstrate the protocol and show that a possible solution that satisfies all requirements can be developed using off-the-shelf components. As a prototyping platform, we choose to use platform with a Xilinx Zynq-7000 family All Programmable System on Chip (APSoC) device. This platform comprises of a GPP and an FPGA, which gives the flexibility to design and develop for hardware, software, and hardware/software co-design paradigms discussed earlier. For this prototype, we chose to use the NaCl core for the ECC scalar multiplication, introduced earlier in Section 3.3.2. In the following, we take a closer look the *Zynq device* and at the *NaCl core*.

#### Xilinx Zynq-7000 family APSoC

After a considerable evaluation and some experimentation, we decided that the most suitable platform for prototyping this system is the Xilinx Zynq APSoC [62] product family, which provides a fast time to market, flexibility and upgrade-ability when compared to traditional System on Chip (SoC). A SoC is usually referred to an ASIC which

[1] ARM Cortex-M0 Processor: https://www.arm.com/products/processors/cortex-m/cortex-m0.php

hosts heterogeneous components needed for a particular purpose on a single die, rather than separate subsystems connected to each other on a Printed Circuit Board (PCB). Benefits of SoC are lower cost, smaller physical size and better reliability, faster and more secure data transfer between various system elements, lower power consumption etc. [62] But obviously, ASIC SoCs are rigid and lack flexibility because they cannot be modified after production. Moreover, such systems require substantial non-recurring engineering costs and time. This is a feasible choice for high volume production, where no future upgrades are necessary. On the other hand, FPGAs have been a great solution, so far, that have the flexibility and no non-recurring costs. FPGAs are programmable devices where logic can be reconfigured after production. For more more information about FPGA technologies, refer to [63].

The architecture of Zynq constitutes of primarily two segments: the Processing System (PS) and the Programmable Logic (PL) as shown in Figure 5.3 The PS consists

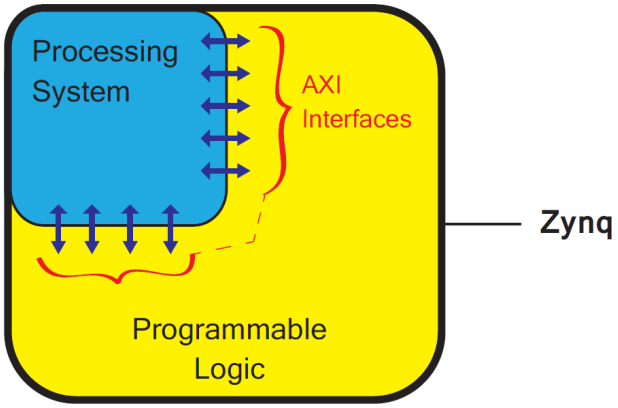



Figure 5.3: Zynq platform

of a 'hard' ASIC implementation of ARM® dual Cortex®-A9 based processor cores along with other supporting peripherals such as a dedicated and optimized silicon element, whereas the PL side is the Xilinx 7-series FPGA fabric. Both sides are closely integrated. An industry standard Advanced eXtensible Interface (AXI) interface, which provides high bandwidth, low latency between the two parts, is used to connect them. An excellent source of reference where one can find all kinds of information about the Zynq platform is the Zynq Book [62].

This architecture is ideal for prototyping and proof-of-concept demonstration of this project. It allows us to implement a self-contained solution that hosts all the necessary custom hardware modules on the fabric side and makes it possible to use conventional software methods to interface with those hardware modules.

### NaCl hardware core

We will use the *NaCls Crypto_box in Hardware* [44] (NaCl core) for the scalar multiplication, which is one of the key operations in the Diffie-Hellman Key Exchange (DHKE) protocol as described earlier in Section 4.2.3. As mentioned earlier in Section 3.3.2,

NaCl core is an example of low-resource hardware implementation of the widely known *crypto_box* function of the 'Networking and Cryptography' (NaCl) crypto library. The NaCl core supports the X25519 Diffie-Hellman key exchange using Curve25519, the Salsa20 stream cypher, and the Poly1305 message authenticator [44]. There are several reasons why this particular core is chosen. The NaCl core is in sync with all the requirements listed in Section 4.1: Firstly, it is a *technology independent* hardware implementation targeting highly *resource-constrained* devices i.e., *optimized for area*. Secondly, the VHDL code of the core is in the *public domain* and therefore freely available for the public. This allows us to modify the existing solution to best fit our need if necessary. Moreover, by using the NaCl core, we build on top of previous academic work and reduce development time.

The NaCl core is designed and developed as an ASIP, which is, as we advocated, the most suitable for our need. The NaCl core constitutes of a memory unit, a controller, and a NaCl-specific Arithmetic Logic Unit (ALU). The NaCl core was specifically designed for a lightweight implementation; therefore, several decisions and trade-offs were made by the authors already for that reason. They implement a 32-bit single-port RAM, due to its smaller area sizes compared to dual-port RAMs. To compensate for the reduced throughput of single-port memories they have incorporated other techniques, e.g. optimised single-port memory arithmetic by using an additional register at the ALU to buffer the fetched values from the RAM etc. The controller hosts two Read-Only Memory (ROM) modules for storing the program, a program counter, instruction decoder, dedicated multiplication controller and a memory management unit. The two ROMs contain separate code, first for Curve25519 and the second for XSalsa20 and Poly1305. The multiplication controller implements big-integer multiplication using the smaller 32x32 bit multiplications. The multiplier is a customizable digit-serial-multiplier. This NaCl core supports 46 instructions, 26 of which are general purpose and 20 are NaCl-specific. The programs stored in ROM are hardcoded in a specific machine-code. A special program is provided, that generates the machine-code from an assembly looking like program. In addition, the NaCl core comes with a SELFTEST module that can be enabled or disabled. The SELFTEST module is a hardware implemented state machine that supplies hardware burned-in input test vectors and compares them with corresponding output values.

### 5.2.2 System Integration and Experiments Performed

For this experiment, we chose to partition the functionality among the different parts of the Zynq board as shown in Figure 5.4. The most complex, non-trivial and time consuming operation in the protocol is the *scalar multiplication* that is present in ECDH. Therefore, we focus on placing a hardware implementation of this element on the FPGA, and augment it with all other necessary components either in hardware or software. The prototyping board has an on-chip SRAM, which theoretically can be used as a PUF. However, accessing it and reading its non-initialized start-up values is not straightforward. Therefore, for the sake of simplicity, an external SRAM module is connected to the board. The application stack responsible for executing the protocol and implementing communication, as well as a fuzzy-instructor are implemented in software on the GPP

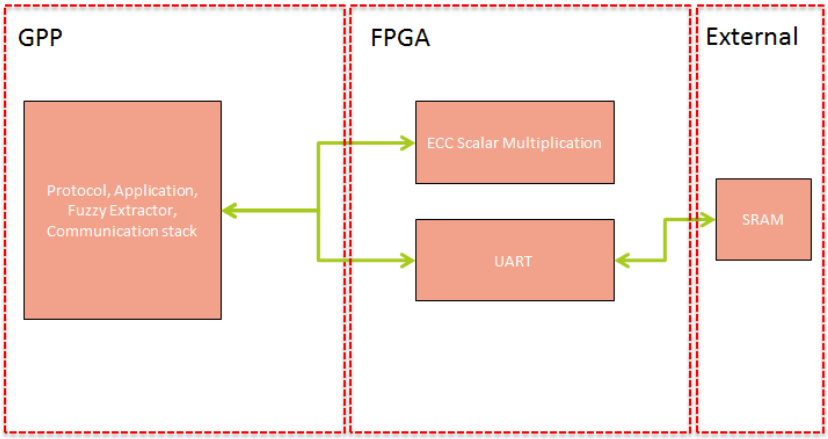



Figure 5.4: Mapping functionality onto the Zynq board

side of the Zynq device.

In the following, we show a step-by-step way of building the proof-of-concept system, building it in a bottom-up fashion. First we look at the (I) scalar multiplication operation using the NaCl core. Thereafter, we (II) develop software support system e.g., drivers for the Zynq device, where the key operations are abstracted for the programmer. Thereafter, we (III) add secure communication, and finally, (IV) integrate it with PUF technology.

**(I) Scalar Multiplication using NaCl core.** We begin by integrating the ECC scalar multiplication unit. Obviously, one of the essential steps was to read the documentation, understand the interfaces as well as the different components within the core; followed by, simulations, and eventually a synthesis and verification of the core on an FPGA board. Prior to deciding to use the Zynq device for prototyping, we used a Xilinx Kintex-5 FPGA. In order to understand the core, we have made a special set-up, which will be briefly discussed next.

**Basic Verification.** The NaCl core by itself is 'empty': the ROM contents need to be filled in and certain configuration parameters need to be configured. A NaCl compiler, supplied along with the source code[2], reads in the application program and configuration parameters, and generates a corresponding ROM content file as well as a configuration file in VHDL. There are multiple configurations that are possible such as including a SELFTEST unit or not, using a special multiplication controller circuit or not, and configuring the multiplier itself, that is the 32-bit multiplier present in NaCl can be set to be a 2, 4, 8, and 16 cycles.

To be able to interface, and therefore test the NaCl core, we needed set up the system in way that allowed us to take control of the core. This was made possible by connecting

[2]NaCl hardware code publicly available at http://mhutter.org/research/vlsi/#naclhw and at http://cryptojedi.org/crypto/#naclhw

the NaCl core to the BOARD TESTER[3] component as shown in Figure 5.5. Using the BOARD TESTER one can issue commands, read and write to the NaCl core. The BOARD TESTER component hosts a UART unit that allows external control. Using a special dedicated BOARD TESTER interpreter, we can write custom programs for testing the NaCl core. With this properly configure system we can *write*, *read*, *check* and *poll* signals that we dedicated and connected to the NaCl core. Having this kind of interface allowed us to experiment and test with the NaCl core. Certain tests were performed and the results were reproduced as in the original paper [44].

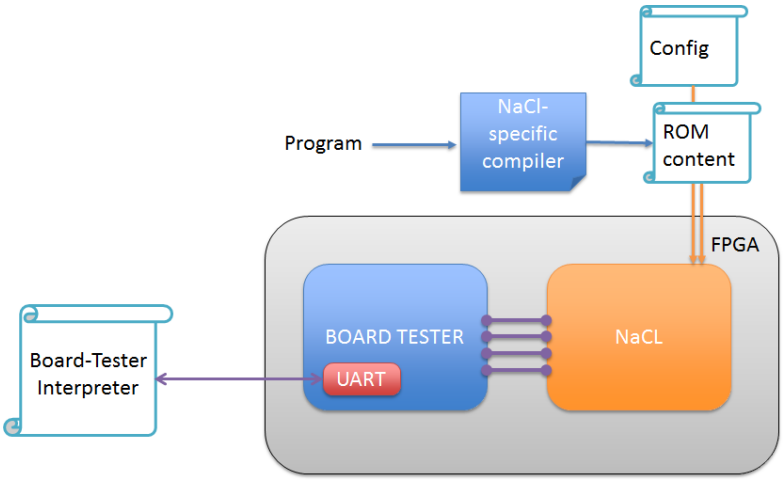



Figure 5.5: NaCl core verification

**Area and Performance.** The performance of the NaCl core mainly depends on the chosen multiplier configuration. The fastest two-cycle version of the core utilises 2754 Slice LUTs on a Xilinx Artix®-7 FPGA and takes approximately 830882 cycles for a scalar multiplication. The original core contains other functionality such as the XSalsa20 and Poly1308 code [44], compiled and stored in the ROM program that we do no need for our project. Because we do not need these primitives in our protocols, we, therefore, reduced the program to its minimum, recompiled it and re-synthesized the core. This reduced the ROM size by approximately a factor of two. The new reduced utilisation becomes 2080 Slice LUTs. We skip the intermediate versions here and proceed to the slowest version, configured as a 16 cycle multiplier. This configuration takes 3475123 cycles, and obviously the least area utilisation of 946 Slice LUTs. This is summarized in Figure 5.6.

**(II) Abstracting Key Operations.** The most obvious choice for a prototyping platform is the Xilinx Zynq APSoC platform described earlier. This platform tightly couples together a processor with the fabric, and the communication is possible via the AXI-peripheral. The first step was to wrap the NaCl core into an AXI-peripheral.

[3]The BOARD TESTER is an Intrinsic-ID in-house developed tool that facilitates testing and verification of hardware components.

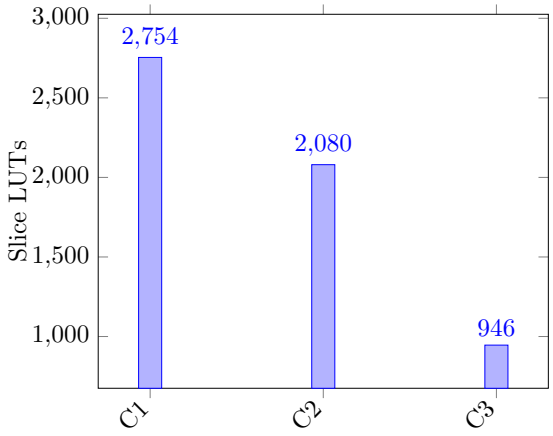



Figure 5.6: Minimizing the NaCl core on a Zynq APSoC (Artix®-7 FPGA) C1: Two-cycle Multiplier; C2: Two-cycle Multiplier and a reduced ROM; C3: 16-cycle Multiplier

The second important step was to abstract the hardware and expose only the high-level operations to the programmer, by creating *hardware interface drivers*. Some of the high-level operations API are reading from and writing to NaCl registers and performing the scalar multiplications as shown in Listing 5.1.

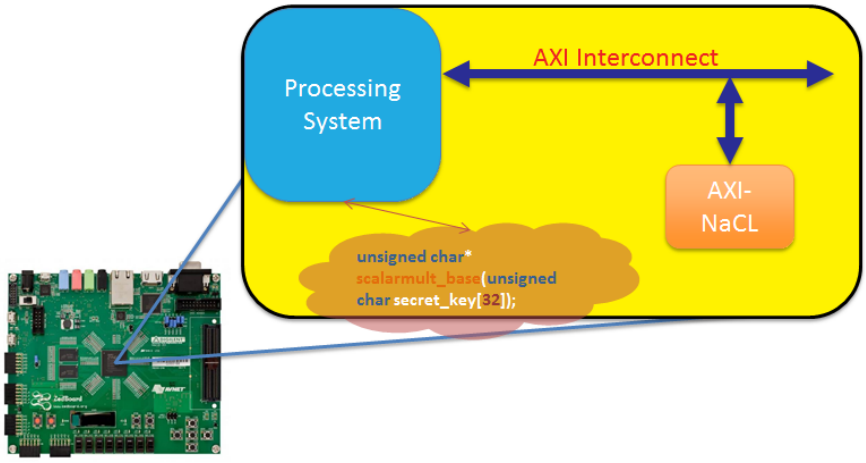



Figure 5.7: Setup on a ZedBoard, hosting a Xilinx Zynq device

```
1  void write_to_reg(char reg, unsigned char value[32]);
   unsigned char* read_from_reg(unsigned char reg);
3  unsigned char* scalarmult_base(unsigned char secret_key[32]);
   unsigned char* scalarmult(unsigned char secret_key[32], unsigned char
   PK_point[32]);
5
```

Listing 5.1: Hardware Abstraction: NaCl C interface

**(III) Adding Secure Communication.** At this stage, we have the NaCl core embodied into the fabric, with its operations exposed to the programmer. An essential part of this system is communication to the outside world. The actual way or the medium of communication is irrelevant for this project. Therefore, simple serial communication with a Host Processor is sufficient. The next step is to integrate security into this communication. We implement Protocol 4.4 with a slight twist. At this point in time, there is no PUF-system on the prototype board yet. Therefore the secret-key values are forced. We verify that both the device and the Host Processor can achieve a common shared key. The host processor app was developed as well and is briefly discussed later in this Section.

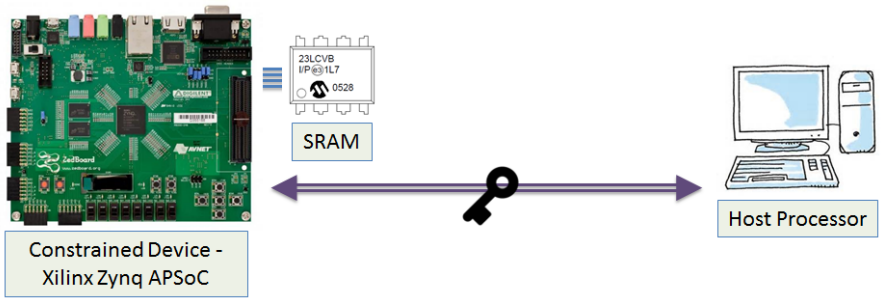



Figure 5.8: The final setup showing the proof-of-concept

**(IV) Integrate PUF Technology** To finalize the implementation of the proof-of-concept, we need to integrate the SRAM PUF technology into our prototype. For demonstration purposes, we would be using an external SRAM chip, namely 23LCV1024 from Microchip[4]: it is an off-the-shelve '1 Mbit SPI Serial SRAM' module IC in an 8-pin DIP package. To make this project open source so that anyone could benefit from it, we have decided to avoid using Intrinsic-ID proprietary PUF technology, which is developed to work under extreme conditions, and has high error correcting capabilities, etc. Instead, as a proof of concept, we implement our simple mock-up fuzzy-extractor, which is based on simple error correcting codes, that is still able to reconstruct the key under normal conditions. Optimising a particular fuzzy-extractor is not within the scope of this project. Therefore, we implement and use a very simple error correcting code. The error correcting code we use for encoding and decoding during the enrollment and reconstruction phases respectively is a repetition code. The fuzzy extractor used in the prototype is shown in Figure 5.9[5].

The rep(64,1) encoder turns one bit into a 64 bit codeword. Therefore, 2kB chunk is produced when a 256-bit key is input to the encoder. To reverse this operation, the counterpart decoding logic for a repetition code is implemented by a majority decoder logic. The Hamming weight of the 64 bit codeword at the input will determine whether it is a zero or a one. Although it is a naive error correcting code, *rep(r,1)*'s error

[4]Data Sheet: http://ww1.microchip.com/downloads/en/DeviceDoc/25156A.pdf

[5]The repetition code is an even number because it allows for an easier implementation on the chosen platform.

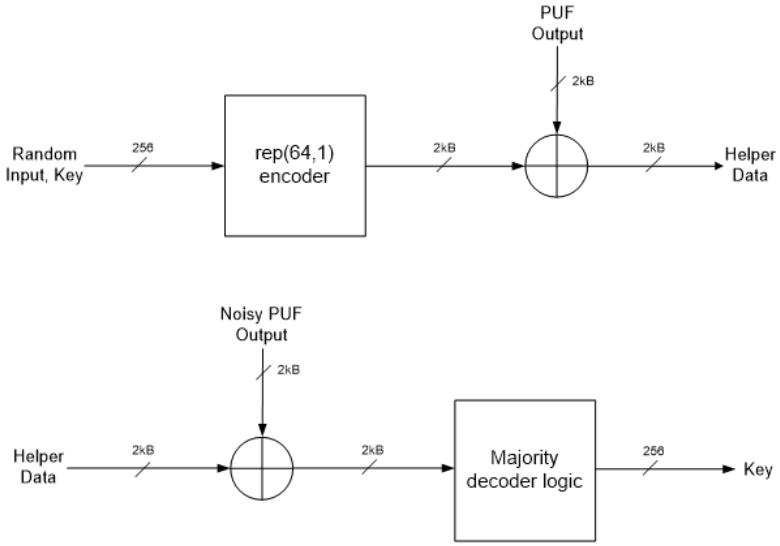



Figure 5.9: Fuzzy-Extractor used in the proof-of-concept

correcting capabilities are $(r-1)/2$. A $rep(63,1)$ code is able to correct up-to 31 errors in a 64-bit codeword. Therefore the error tolerance is $31/63 = 49\%$. However, a quick experiment showed that a conservative and safe threshold is around 30% error as shown in Figure 5.10. Another profiling experiment revealed that the same SRAM chip used in this setup has the following properties:

(a) Noise range of $5-10\%$ under normal room conditions. Meaning that the fractional hamming distance $d(R, R')$(see Eq. 3.14), where R is the reference measurement and R' are subsequent measurements taken from the same device is 0.05 *to* 0.10.

(b) Hamming weight (see Eq. 3.20) of approximately 68%.

(c) Hamming distance between devices around 44%. Meaning that the $d(R_1, R_2)$ where $R_1$ and $R_2$ are the reference measurement of different devices is 0.44.

This experiment shows that the fuzzy extractor used in the experimental setup could be used under normal room conditions since the SRAM startup noise is within the error correcting capabilities of the fuzzy extractor. However, the Hamming weight is 68% which indicates a global bias; therefore, a debiasing technique must be used to compensate for that as discussed in Section 3.4.3.

Figure 5.8 shows the conceptual view of the overall system and Figure 5.11 shows the block design of the integrated system on the Zynq APSoC. The ZYNQ7 Processing System block connected to two IP blocks via the AXI peripheral. The NaCl_0 block is the minimal configuration NaCl core. Two additional signals led0 and led1 are added for debugging purposes. The AXI peripheral and the NaCl core run at two different frequencies. Therefore an external clock is supplied to the NaCl core. The clock domain crossing had to be taken care of. The axi_uartlite_0 block is used to interface an external SRAM chip.

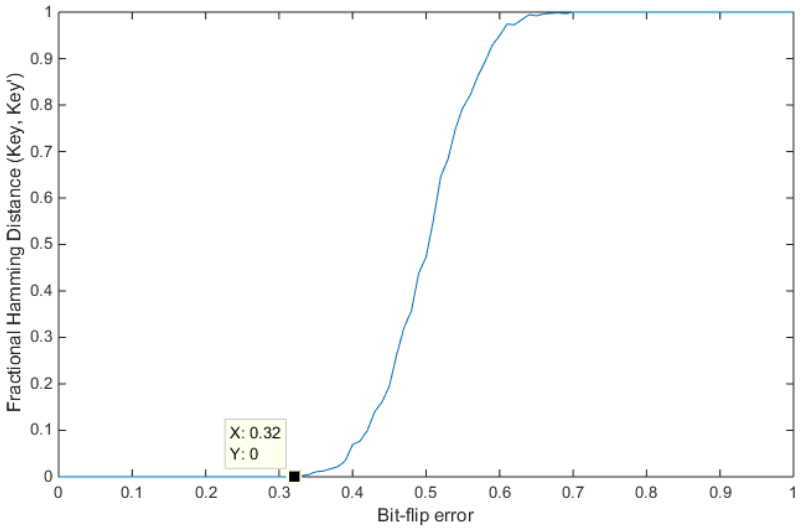



Figure 5.10: Behaviour of the Fuzzy-Extractor using rep(64,1) encoder. Note that the decoder fails beyond the 32% error.

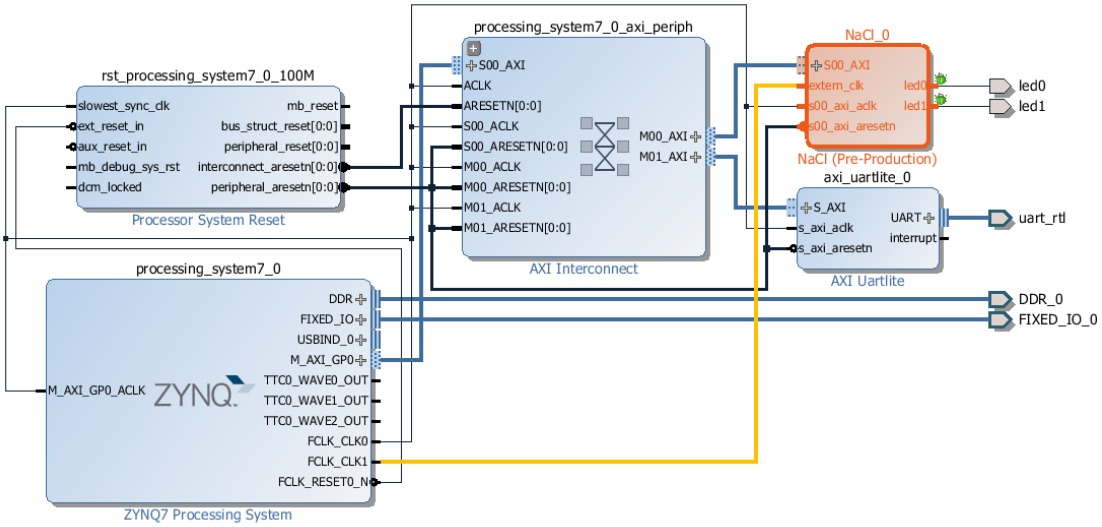



Figure 5.11: Top level block design of the prototype on the Zynq APSoC

### Host Processor

In order to emulate the protocol, and the communication with the host processor it was decided to put together a GUI that will do that and also provide a 'screen' to the prototyping board. A screenshot can be seen in Figure 5.12. A procedure is as follows; when a user clicks the enroll button, an 'enroll' command is sent to the device. When this command is received on the device, it begins by reading out the SRAM values from

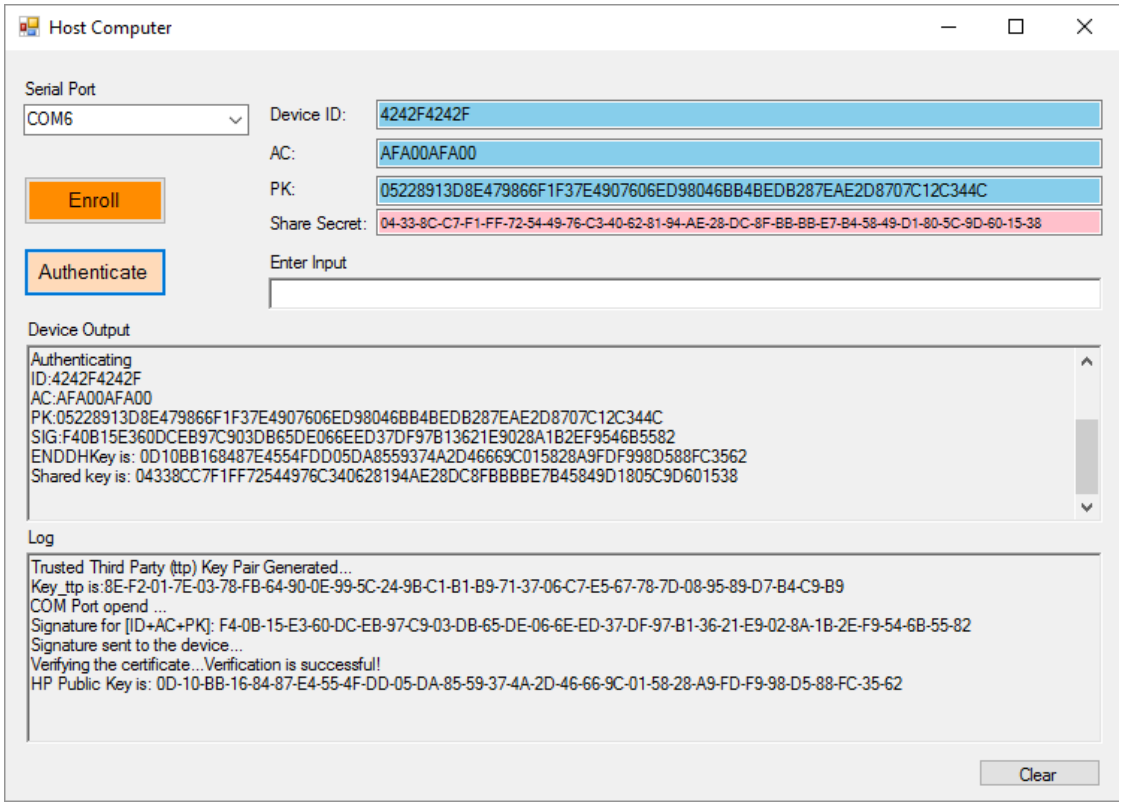



Figure 5.12: Host Processor GUI; used to *enroll*, *authenticate* and provide the output of the device.

the external chip and feeds it into the fuzzy-extractor which produces the Activation Code (AC). The secret key is used to produce the corresponding public key using the scalar multiplication unit. Since Protocol A has been implemented for this experiment, the necessary information is sent back to the host processor, which then signs the data and send back a certificate. The certificate is stored on the device.

When an authenticate command is sent to the device, the device first reads the SRAM values, and uses it along with the previously stored activation code to reproduce the secret key as before. Once the secret key is recovered, it is used by the scalar multiplier to produce the corresponding public key. Both parties then proceed with the DHKE as in Protocol 4.4 Steps 14-15 to derive a shared key. From here on, the communication can be encrypted using this shared key.

### 5.2.3 Discussion

The challenge was to enable a secure communication with authenticated resource-constrained IoT devices. Several cryptographic protocol variants were proposed in the previous chapter that solves this challenge. In this chapter we focused on designing a hardware architecture that supports these proposed protocols. Thereafter, we went through a systematic way of building a prototype of this secure communication system.

The prototype allowed us to verify, evaluate, and analyze the feasibility of such a system. Further, it has given us insight on what can be done, what can be improved and optimised. Despite that it is only a prototype, it is a big step towards making a real product out of it; since it lays foundation for both low-volume FPGA and high-volume ASIC production. For instance, the entire IoT device along with the designed security related parts can be rolled out for production on a small Cost-Optimized Zynq devices such as Single-Core *Z-7007S* or Dual-Core *Z-7010*[6] if needed. For example, Zynq-7000 family such as XC7Z010-1CLG225C can be bought for 57.75 USD per piece[7]. As stated earlier, this has the potential to be a viable option for a low to medium production, and a fast time to market. Alternatively, for a higher production volume, we might need to design an ASIC. Although the non-recurring costs are known to be high for such design, the resulting per-unit price can be substantially minimised this way. Furthermore, ASIC design can be optimised for an area, resulting in the smallest form factor.

[6]Zynq-7000 Product Selection Guide: https://www.xilinx.com/support/documentation/selection-guides/zynq-7000-product-selection-guide.pdf

[7]DigiKey Online product price: http://www.digikey.com/product-detail/en/xilinx-inc/XC7Z010-1CLG225C/122-1855-ND/3925788.

# 6 Conclusion

## 6.1 Summary

In this thesis, we showed the importance of secure communication throughout history. Although in early ages secure communication was used in niche applications such as the military, today it is an essential part of our connected online world. The Recent proliferation of Internet of Things (IoT) devices, their diversity and the need for security, impose many new challenges. One such challenge is that one-size-fits-all solutions are not efficient. Therefore, we need to design and develop application-specific solutions. In this thesis, we examined a use-case: secure communication with resource-constrained devices; with the primary focus of key-agreement and authentication of the silicon. We showed what a secure system is, discussed some incentives behind the attacks and provided a broad taxonomy of them. We discussed the security requirements such as integrity, authenticity, confidentiality, and non-repudiation, and what they mean; and, how certain cryptographic systems can equip us with tools that can guarantee these properties. In the related work chapter, we discussed public-key cryptography and its key components. Furthermore, we looked at some of the Public-key Cryptography (PKC) cores that exist in academia and the industry. We go in-depth on this subject because this primitive is responsible for key-establishment, the crucial mechanism for enabling a secure communication in the use-case under investigation in this thesis. With regards to PKC, a gentle overview of integer factorization, discrete logarithm and elliptic curve discrete logarithm problem based public-key primitives such as RSA, Diffie-Hellman (DH), and Elliptic Curve Diffie-Hellman (ECDH) respectively are discussed. Elliptic Curve Cryptography (ECC) has been gaining popularity in the community as a suitable candidate for constrained devices. The reason is due to much shorter operands that are required for achieving the same level of security as compared to others. Shorter operands imply simpler hardware. Next, we introduced PUF technology. PUF is an intrinsic hardware security primitive that is an essential part of this thesis. It is the hardware root-of-trust, which provides a strong authentication mechanism to authenticate the 'silicon' of the device. Among the different types of PUFs, we have mainly discussed the SRAM PUF. Therefore, a set of requirements is presented, influenced by the academia and the industry. Eliciting a set of requirements is essential for a project like this; it shaped how the protocol and the consequent hardware architecture is designed and developed. The first proposed protocol was a modification of a Diffie-Hellman Key Exchange (DHKE), which used a PUF-based key. The analyziz showed that it was a one-way authentication with a non-negligible Non-volatile Memory (NVM) requirement on the device's side. Three more protocol variants were proposed after that to mitigate those issues. A thorough comparison of the variants was provided at the end; showing the trade-offs related to security versus implementation requirements. The designed protocol served as a roadmap

in drafting a modular hardware architecture that can be used as a guideline in designing architectures that support the protocol. One such instance was chosen to be verified and prototyped on the Xilinx Zynq-7000 APSoC device. This allowed us to analyze its practicality as well as its feasibility. Furthermore, the prototype offered interesting insights and laid a solid foundation for future research, which is the subject of the following section.

## 6.2 Future Work

This works lays a solid foundation for future research and investigation. Different alternative paths can be pursued; which are discussed next.

(I) *Related to the NaCl-core:*

- *Expanding the NaCl core:* The NaCl core used in this project is a public-domain ASIP implementation, used to perform ECC scalar multiplications on Curve25519. Although this core has application specific instructions, it still mostly depends on the software code stored in ROM.

  Perhaps equipping this core with additional accelerators, or more complex instructions may be feasible. In other words, shifting some functionality towards 'state-machine' implementation, and conducting an extensive design-space exploration can be done.
- *Fuzzy-Extractor on the NaCl:* As stated before, the NaCl Core is an ASIP. It consists of general purpose as well as an application specific instruction set. It is worth investigating to see whether a suitable error-correcting scheme can be ported on to the NaCl core without modifying or with few modifications to the core.

(II) *Related to the current implementation on a Zynq device:* Further investigations can be done that target the Zynq device, such as:

- The Zynq platform has an on-chip memory module such as the 256KB SRAM[1] which in theory can be used as a PUF. Further investigation is required to see if the on-chip SRAM can be 'read' in its uninitialized state; and profiled to check its feasibility as a PUF.
- Leverage other available technologies. In particular, the on-chip ARM core is equipped with TrustZone. Therefore, all security related software operations should eventually leverage that. Furthermore, The Zynq platform provides mechanisms for power management, and clock throttling, which can be employed for power optimization in future implementations.

(III) *Evaluating the solution using other technology:*

[1] Zynq Product Specification: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

- The proposed hardware template can be further spun into an ASIC design and implementation.
- The Microsemi SmartFusion2 SoC FPGA Family[2] hosts an on-chip Intrinsic-ID's Quiddikey PUF technology. An interested party might wish to evaluate or even deploy a product on this platform.

[2]Product Overview: http://www.microsemi.com/products/fpga-soc/soc-fpga/smartfusion2#overview.

# Bibliography


[1] S. Singh, *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. Anchor Books, 2000. [Online]. Available: https://books.google.nl/books?id=skt7TrLK5uYC

[2] T. Kelly, "The myth of the skytale," *Cryptologia*, vol. 22, no. 3, pp. 244–260, 1998. [Online]. Available: http://dx.doi.org/10.1080/0161-119891886902

[3] B. Copeland, "Alan turing." [Online]. Available: http://www.britannica.com/biography/alan-turing

[4] A. Kerckhoffs, *La cryptographie militaire, ou, Des chiffres usités en temps de guerre: avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef*, ser. Extrait du Journal des sciences militaires. Librairie militaire de L. Baudoin, 1883. [Online]. Available: https://books.google.nl/books?id=VbQBAAAAYAAJ

[5] P. S. Ravikanth, "Physical one-way functions," Ph.D. dissertation, Cambridge, MA, USA, 2001, aAI0803255.

[6] R. Maes and I. Verbauwhede, "Physically unclonable functions: A study on the state of the art and future research directions," in *Towards Hardware-Intrinsic Security*. Springer, 2010, pp. 3–37.

[7] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, "Secure key generation from biased pufs," Cryptology ePrint Archive, Report 2015/583, 2015, http://eprint.iacr.org/.

[8] V. van der Leest and P. Tuyls, "Anti-counterfeiting with hardware intrinsic security," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, March 2013, pp. 1137–1142.

[9] "Intrinsic id," https://www.intrinsic-id.com/, accessed: 2016-04-20.

[10] C. Paar and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[11] B. Schneier, *Secrets & Lies: Digital Security in a Networked World*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[12] ——, *Digital Threats*. Wiley Publishing, Inc., 2015, pp. 14–22. [Online]. Available: http://dx.doi.org/10.1002/9781119183631.ch2

[13] S. Hamdioui, G. Di Natale, G. van Battum, J.-L. Danger, F. Smailbegovic, and M. Tehranipoor, "Hacking and protecting ic hardware," in *Proceedings of the Conference on Design, Automation & Test in Europe*, ser. DATE '14. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2014, pp. 99:1–99:7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2616606.2616728

[14] S. P. Skorobogatov, "Semi-invasive attacks: a new approach to hardware security analysis," Ph.D. dissertation, University of Cambridge Ph. D. dissertation, 2005.

[15] D. G. Abraham, G. M. Dolan, G. P. Double, and J. V. Stevens, "Transaction security system," *IBM Systems Journal*, vol. 30, no. 2, pp. 206–229, 1991.

[16] S. P. Skorobogatov, "Semi-invasive attacks – a new approach to hardware security analysis," 2005.

[17] A. Monteiro Oliveira Cortez, "Reliability assessment and test methods for anti-counterfeiting technology," Ph.D. dissertation, TU Delft, Delft University of Technology, 2015.

[18] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *Privacy, Security and Trust (PST), 2015 13th Annual Conference on*. IEEE, 2015, pp. 145–152.

[19] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing (4th Edition)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2006.

[20] S. S. Kumar, "Elliptic curve cryptography for constrained devices," Ph.D. dissertation, Ruhr University Bochum, 2006.

[21] B. Mandal, S. Chandra, S. S. Alam, and S. S. Patra, "A comparative and analytical study on symmetric key cryptography," in *Electronics,Communication and Computational Engineering (ICECCE), 2014 International Conference on*, Nov 2014, pp. 131–136.

[22] C.-C. Lu and S.-Y. Tseng, "Integrated design of aes (advanced encryption standard) encrypter and decrypter," in *Proceedings IEEE International Conference on Application- Specific Systems, Architectures, and Processors*, 2002, pp. 277–285.

[23] A. Hodjat and I. Verbauwhede, "A 21.54 gbits/s fully pipelined aes processor on fpga," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*. IEEE, 2004, pp. 308–309.

[24] C. H. Gebotys, *Security in Embedded Devices*. Springer, 2010.

[25] K. Huang and R. Tso, "A commutative encryption scheme based on elgamal encryption," in *2012 International Conference on Information Security and Intelligent Control*, Aug 2012, pp. 156–159.

[26] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1976.1055638

[27] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987. [Online]. Available: http://www.jstor.org/stable/2007884

[28] V. S. Miller, *Use of Elliptic Curves in Cryptography.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 417–426. [Online]. Available: http://dx.doi.org/10.1007/3-540-39799-X_31

[29] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[30] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. [Online]. Available: http://doi.acm.org/10.1145/359340.359342

[31] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik *et al.*, "Factorization of a 768-bit rsa modulus," in *Annual Cryptology Conference*. Springer, 2010, pp. 333–350.

[32] D. Boneh *et al.*, "Twenty years of attacks on the rsa cryptosystem," *Notices of the AMS*, vol. 46, no. 2, pp. 203–213, 1999.

[33] P. L. Montgomery, "A survey of modern integer factorization algorithms," *CWI quarterly*, vol. 7, no. 4, pp. 337–366, 1994.

[34] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999. [Online]. Available: http://dx.doi.org/10.1137/S0036144598347011

[35] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[36] M. Joye and S.-M. Yen, *The Montgomery Powering Ladder.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 291–302. [Online]. Available: http://dx.doi.org/10.1007/3-540-36400-5_22

[37] T. Izu, B. Möller, and T. Takagi, *Improved Elliptic Curve Multiplication Methods Resistant against Side Channel Attacks.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 296–313. [Online]. Available: http://dx.doi.org/10.1007/3-540-36231-2_24

[38] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs.* Oxford, UK: Oxford University Press, 2000.

[39] A. Liu and P. Ning, "Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, April 2008, pp. 245–256.

[40] G. Hinterwälder, A. Moradi, M. Hutter, P. Schwabe, and C. Paar, *Full-Size High-Security ECC Implementation on MSP430 Microcontrollers.* Cham: Springer International Publishing, 2015, pp. 31–47. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16295-9_2

[41] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.

[42] P. Sasdrich and T. Güneysu, *Efficient Elliptic-Curve Cryptography Using Curve25519 on Reconfigurable Devices*. Cham: Springer International Publishing, 2014, pp. 25–36. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-05960-0_3

[43] E. S. Kumar and C. Paar, "Are standards compliant elliptic curve cryptosystems feasible on rfid," in *In Proc. of RFIDSec06*, 2006.

[44] M. Hutter, J. Schilling, P. Schwabe, and W. Wieser, *NaCl's Crypto_box in Hardware*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 81–101. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-48324-4_5

[45] R. Maes, *Physically Unclonable Functions - Constructions, Properties and Applications*. Springer, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-41395-7

[46] V. van der Leest and A. Schaller, "Physically unclonable functions found in standard components of commercial devices," 2013, http://www.intrinsic-id.com/wp-content/uploads/2014/09/Unclonable-functions.pdf.

[47] M. Roel, "Physically unclonable functions: Constructions, properties and applications," Ph.D. dissertation, Ph. D. thesis, Dissertation, University of KU Leuven, 2012.

[48] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[49] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters, *Read-Proof Hardware from Protective Coatings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 369–383. [Online]. Available: http://dx.doi.org/10.1007/11894063_29

[50] A. Cortez, A. Dargar, G. Schrijen, and S. Hamdioui, "Modeling sram start-up behavior for physical unclonable functions," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Austin, USA, October 2012, pp. 1–6.

[51] 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 *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. IEEE, 2011, pp. 567–570.

[52] X. Boyen, "Reusable cryptographic fuzzy extractors," in *Proceedings of the 11th ACM Conference on Computer and Communications Security*, ser. CCS '04. New York, NY, USA: ACM, 2004, pp. 82–91. [Online]. Available: http://doi.acm.org/10.1145/1030083.1030096

[53] Y. Dodis, L. Reyzin, and A. Smith, *Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 523–540. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24676-3_31

[54] J.-P. Linnartz and P. Tuyls, *New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 393–402. [Online]. Available: http://dx.doi.org/10.1007/3-540-44887-X_47

[55] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *2007 44th ACM/IEEE Design Automation Conference*, June 2007, pp. 9–14.

[56] K. Okeya and K. Sakurai, *Fast Multi-scalar Multiplication Methods on Elliptic Curves with Precomputation Strategy Using Montgomery Trick.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 564–578. [Online]. Available: http://dx.doi.org/10.1007/3-540-36400-5_41

[57] A. Van Herrewege, V. van der Leest, A. Schaller, S. Katzenbeisser, and I. Verbauwhede, "Secure prng seeding on commercial off-the-shelf microcontrollers," in *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, ser. TrustED '13. New York, NY, USA: ACM, 2013, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/2517300.2517306

[58] B. Yang, K. Wu, and R. Karri, "Secure scan: A design-for-test architecture for crypto chips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2287–2293, Oct 2006.

[59] H. Jiang, M. Marek-Sadowska, and S. R. Nassif, "Benefits and costs of power-gating technique," in *2005 International Conference on Computer Design*, Oct 2005, pp. 559–566.

[60] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 3, pp. 415–420, Mar 2000.

[61] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of fpgas," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[62] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, *The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc.* UK: Strathclyde Academic Media, 2014.

[63] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of fpgas," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, Aug 2007.