CERIAS Tech Report 2012-04
Secure Physical System Design Leveraging PUF Technology
by Sam Kerr
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

SECURE PHYSICAL SYSTEM DESIGN LEVERAGING PUF TECHNOLOGY

A Thesis

Submitted to the Faculty

of

Purdue University

by

Samuel T. Kerr

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2012

Purdue University

West Lafayette, Indiana

ACKNOWLEDGMENTS

A lot of people have had an impact in my life, both academic and personal. I wish to thank all of you that have influenced it in a positive way.

I would like to especially thank Professor Elisa Bertino. She took a risk on me as a freshmen and allowed me to get into research. I have greatly enjoyed learning and growing under her direction these past few years. Without such a great mentor, I don't think I would be where I am today.

Michael Kirkpatrick also deserves a lot of thanks. He helped guide some of my initial work with PUF devices, pointing out things I missed and helping me mature as a researcher. His lessons were invaluable, not only about the research topic themselves, but also about how research is done.

Sypris Electronics and its staff were also an invaluable asset to have during my course of study as well. They funded many of the projects that I worked on as well as made engineers available to help me learn and move forward with my work.

Finally, my parents deserve a special thanks. Without the wonderful upbringing and strong sense of values they instilled in me, none of this would have been possible. I love you Mom and Dad, thanks for pushing me.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

PUF	Physically Unclonable Function
PEAR	Physically Enhanced Authentication Ring
ROK	Read Once Key
RS	Reed-Solomon Codes
ZKPK	Zero Knowledge Proof of Knowledge

ABSTRACT

Kerr, Samuel T. M.S., Purdue University, May 2012. Secure Physical System Design Leveraging PUF Technology. Major Professor: Elisa Bertino.

Physical systems are becoming increasingly computationally powerful as faster microprocessors are installed. This allows many types of applications and functionality to be implemented. Much of the security risk has to do with confirming the device as an authentic device. This risk can be mitigated using a technology known as Physically Unclonable Functions (PUFs). PUFs use the intrinsic differences in hardware behavior to produce a random function that is unique to that hardware instance. When combined with existing cryptographic techniques, these PUFs enable many different types of applications, such as read once keys, secure communications, and secure smart grids.

1. INTRODUCTION

There are many different computer systems in the world today. Many of these systems are general purpose computing systems, such as consumer desktops and laptops. However, there are many more systems with a very specific use and that interact with the world in a physical way. Examples of this include sensor arrays, surveillance systems, or utility pipelines. These are called "physical systems". These systems incorporate a large amount of computation to perform their tasks, but their main tasks are accomplished by interacting with the physical world in some way.

These physical systems are beginning to become more and more complex. Originally, these systems computational abilities were very limited, maybe being restricted to a few hard coded operations, potentially only accessible by an on site technician. Today, many of these systems have a much greater computing capacity, have more dynamic capabilities, and, especially, are more connected to some sort of network, such as the Internet. These improvements have allowed for much greater control over systems, better remote interfacing, and greater efficiency.

With all the improvements however also comes security risks. Suddenly, physical systems are vulnerable to malicious control from an adversary connected over the Internet. Besides just a networked adversary, it is possible that malicious code, such as a virus, might infect the system. Due to the increased processing power and more generalized computing resources, these viruses would have a greater attack surface and more opportunities to compromise such a system.

One of the main problems of physical systems is that it is difficult to uniquely identify them. That is, when allowing remote communication with a physical system, parties are not completely sure that the system they are communicating with is authentic. An attacker might have made a copy of the device and could be impersonating the device. Alternatively, the device could be a counterfeit. What is needed

is an approach to ensure that the device is actually the intended device.

A novel technology called Physically Unclonable Function (PUF) provides the sort of device identification that is needed to solve the previous issue. A PUF is a device that can be used to generate a response that is unique to a given device. PUFs are made by leveraging small inconsistencies in the manufacturing process. As such, it is impossible to duplicate a PUF. Since the PUF cannot be duplicated, if a device ever returns the expected response from its PUF, the other party can be confident that the device is the intended device.

The rest of the thesis is structured as followed.

Chapter 2 describes physical systems and their nature, including several of the difficulties that are involved with them, in more depth.

Chapter 3 introduces some of the necessary cryptography background needed for understanding the rest of the paper.

Chapter 4 introduces PUF technology and creates an initial connection to physical systems. Several different PUF architectures are presented as well as a discussion of some implementation issues.

Chapters 5, 6, and 7 all describe an applications of PUF technology as it incorporated into physical systems. These applications demonstrate the use of PUF as a way of resolving the issues facing physical systems.

Chapter 5 describes a project called Read Once Keys. These are keys that once being read are destroyed and are irrecoverable. The PUF device is used in this case as a way of providing trusted execution.

Chapter 6 describes an authentication approach called Physically Enhanced Authentication Ring. This approach uses a PUF to combat a potentially compromised communication channel, such as when a key logger is installed.

Chapter 7 describes an approach for key management in smart grids and smart meters. A PUF is incorporated into a smart meter and is used to securely authenticate with a utility company, in spite of potential threats.

Chapter 8 draws final conclusions and presents closing thoughts.

2. PHYSICAL SYSTEMS

Physical systems present an interesting problem domain for study. In contrast to software systems, they are subjected to multiple different factors that all require consideration during design. Physical systems frequently must be able to cope with environmental factors such as temperature change, moisture, or questionable power systems.

A purely software system may be able to assume it will only receive input from a standard input and output channel. In contrast, a physical system must be able to account for multiple different input sources, especially input types that might not have been intended. A physical system might receive input directly from end users, networking devices, sensors. A physical system could also consider environment changes as a sort of secondary, unintended input. For example, the device's power may fluctuate, potentially changing the behavior of internal circuits. A temperature change could cause the sensitivity of a certain component to increase or decrease, which will in turn alter the behavior of the system. These are but a few examples of the various factors that a physical system must be properly designed to handle and account for.

2.1 Typical Organization and Use Case of Physical Systems

Physical systems are a very broad category that covers a large set of devices, applications, and use cases. Because of this, it is difficult to discuss physical systems generically. Rather, this section details some of the common configurations that physical systems take. The rest of the work will regard physical systems as belonging to one of the configurations discussed.

Despite it being difficult to characterize physical systems in general terms, their

operation can be viewed using the mathematical relation below. Physical inputs are inputs from physical interfaces, such as buttons, control terminals, or radio signals. Other inputs might be information received over the network or from some sort of attached peripheral.

$$Output = System_{Physical}(PhysicalInputs, OtherInputs, Environment)$$

The three configurations of physical systems that will be considered are that of the standalone, deployed, or peripheral physical systems. Each is distinct based upon how much interaction it has, not only with the physical world, but also with other physical systems or remote devices. Peripheral systems have the most interaction with remote parties and other systems while standalone systems have the least. In terms of the equation above, the three categories vary based on what sort of 'Other Inputs' are passed to them.

2.1.1 Standalone Physical Systems

One common configuration of a physical system is that of a standalone physical system. This means that the physical system is not reliant on communicating with another physical system; it is deployed and functions independently. An example of this could be a garage door opener. There might be a control pad on the side of the garage which can open or close the garage door. Additionally, there could be an option to open the door remotely using some sort of radio frequency device.

Standalone physical systems are more straightforward to deal with in a lot of cases. The garage opener example has a very specific use case, defined inputs (control pad and remote control) and defined outputs (open or close the garage door). These qualities typically do not change nor are updated often, if ever. As such, it is typically easy to create a sort of state diagram to model the behavior of standalone physical systems.

2.1.2 Deployed Physical Systems

Another category of physical systems is that of the 'deployed physical system'. This is a type of physical system that not only interacts with the environment it is in, but may also communicate with another physical system or some sort of remote server. A cash register is a good example of a deployed physical system. It takes input from cashiers, who can record transactions, print receipts, and insert or remove currency from it. However, it also communicates with remote servers in certain cases, such as when a credit card is used. It must interact with the physical environment, but also must interact with remote servers to verify the credit card transactions.

Because a deployed physical system must potentially interact with a remote party, it is more complex than a standalone physical system. It must contend with the same sorts of issues that standalone systems do, but also has to deal with issues that could relate to the remote communication or other physical system. As such, it is more complex and difficult to model a deployed physical system than a standalone physical system.

2.1.3 Peripheral Physical Systems

Peripheral physical systems are the most complex type of physical system. These are normally called 'peripherals'. That is, they do not provide the main functionality of a system, but augment its ability in some way. An example could be a programmable sensor array. The sensor array could be connected to a network through which it receives commands. The array would then take sensor readings and communicate them back over the network. Not only does the array have to interact with the physical environment to take readings, but there is also the component of dealing with the command and control element from the network connection.

Peripheral physical systems are characterized by the fact that they not only require interaction with the physical world, but also with other physical systems or with a remote connection. Because of this, it is very difficult to model the system, since

there are a very large number of ways that the other communicating party could potentially behave, in addition to any difficulties involved with modeling the physical inputs themselves.

2.2 Failure Considerations

There are multiple ways that a physical system or device could fail. There are multiple benign ways that a system could fail. That is, the system is not attacked in any way, but some circumstances cause the system to fail or degrade in some way. There any many different ways to reduce these risks, as discussed below.

2.2.1 Device Failure

One failure model is for a complete device failure. In this instance, the device has failed to such a point that it is no longer able to perform any of its intended function. This typically occurs to some catastrophic component failure or a lack of preventative maintenance as a system's performance degrades over time.

To mitigate the danger of a complete system failure, it is necessary to impose a schedule for periodic maintenance and monitoring of the operating environment for dangerous conditions. An example of this would be inspecting all the moving parts and springs on a garage door opener to ensure they are not cracking or otherwise at risk of failing. Monitoring the environment is critical to ensure that a system is not operating in conditions it was not designed for. If a sensor array was designed for operating indoors and it is placed outside and subjected to weather, of course it will fail.

2.2.2 Device Degradation

One of the few "good" aspects about a complete device failure is that it is readily noticeable. If the system fails completely, it is not possible to interact with it any

more. In contrast, if a device *degrades*, the degradation may not be noticed for a long time, while in the interim, the degraded system will be used under the assumption it is properly functioning.

An example of this is if a sensor array were to be degraded in some way, its readings might be skewed. The skewed readings would then be recorded and fed into a processing program or used by some other party. Depending on the application, this could cause the intended application to then function improperly. In certain instances, this degradation can even prove to be life threatening, such as when temperature sensors in the Fukishima nuclear plant were incorrectly reporting the internal temperature of nuclear reactors. [1]

2.3 Attacks

In addition to the problems that are inherently present in physical systems, it is necessary to consider problems that may occur from attackers maliciously using the system. They may be attempting to gain unauthorized access to the system, prevent legitimate users from using the system, disable the system entirely, or any number of other motivations.

2.3.1 Denial of Services Attacks

It is entirely possible that a malicious entity wants to simply disable and disrupt access to a physical system, preventing legitimate access to the system. If the physical system is unable to deliver valid services to its intended users, it is essentially worthless.

Denial of service attacks against physical systems are unique with regards to the denial of service attacks against software. Some are very complex, while others are very simple. In the simplest case, an attacker can simply use a hammer to smash the system. More complex denial of service attacks may include inputting erroneous data, which may crash or slow the system. Attackers might also disrupt the environment

that the physical system resides in, such that it is not useful. For instance, an attacker might put a heating element near a thermometer, which would essentially mean the system is unusable for its intended purpose.

As far as defenses against these types of attacks go, a first step is usually to ensure that the system is protected against a reasonable amount of tampering. This could include protective cases, placing the system behind a fence, or having a guard present. As mentioned previously, proper maintenance can also be helpful, to prevent an attacker from manipulating and disturbing the surrounding environment.

2.3.2 Man in the Middle Attacks

There is a class of attack known as Man in the Middle (MITM) attacks. This is when an attacker sits between two parties and eavesdrops on their communications. The attacker is then able to learn sensitive data that the two parties are transmitting.

This type of attack is especially relevant for physical systems. Physical systems frequently transmit data over cables, infrared, radio, or other wireless communication methods. If an attacker was able to splice a listening device into a cable or construct the appropriate type of receiver, it is plausible he would be able to easily recover the communications between the two parties.

Depending on the type of data being sent, this could compromise the security and integrity of the system. For instance, maybe an attacker would be able to recover the command sequence to reset a sensor array. He could then reset the sensor array at will.

To combat this type of threat, it is important to assume that any communication being done is being eavesdropped on. This then necessitates *encrypting* the data being transmitted. In this way, even if an adversary was to recover the data, he would not be able to make sense of it. Chapter 3 goes into more detail on encryption techniques.

As a final note, it is important to note that MITM are not relevant for only

signals being transmitted over wireless and wires, but also on the internal buses of the circuits themselves. An attacker might be able to attach logic probes to bus lines between the processor and memory of the system and deduce sensitive data. In this case, it is important to take measures to prevent these bus lines from being exposed, through the use of potting and other tamper-proofing methods. Another option is to incorporate the entire design (or at least the sensitive bus lines) on a single chip, such as a Field Programmable Gate Array (FPGA) or a System on a Chip (SOC).

2.3.3 Impersonation

Another issue for physical system designers to be aware of is that any party they are communicating with is actually an authorized party. This is especially relevant for deployed physical systems and peripheral physical systems. Since they require external communication as a major component of their proper operation, they are especially sensitive to these attacks.

It is plausible that an attacker could disconnect the cables used to communicate and re-attach them to his own machine. He could then issue commands and communicate with the physical system. Unless protective measures are in place, the system would then interact back with the attacker. The attacker could then issue any sorts of commands that he wished of the system.

The issue of impersonation harkens to the need for *authentication*. Chapter 3 goes into more details on authentication protocols, but essentially, all communications between the system and the other party would have to be *signed*. If the signatures do not match the expected values, the communication is rejected and dropped. In this way, an attacker would have to be able to forge the signature of the valid party, which is considered computationally difficult if a proper signature scheme is used.

2.3.4 Replay Attacks

There is a class of attacks that is related to MITM attacks. Replay attacks leverage the fact that certain protocols might consistently send the same data every execution of the protocol. For instance, consider a system that requires the sender to send an encrypted version of an ID number before every message to identify itself. If that ID is always the same, an attacker could simply capture the encrypted text and send that; he does not need to actually know the plaintext version of the ID to impersonate the sending party.

This type of attack can be remedied by ensuring that every execution of a protocol is unique. This is done through the use of time stamps or "nonce" values, which are randomly chosen, one time use values. Then, if an attacker tried to replay previous communications, the attack would fail since the time stamp or the nonce value would not match. So in the example above, the sender might encrypt his ID number concatenated with the current time. Then, an attacker would not be able to re-use any communications he captures in the future.

2.3.5 Signals Injection

Due to the nature of electronics, physical systems are susceptible to external signals being directed at them. If an electric or magnetic field is directed at certain elements of internal circuitry, it is possible to alter the behavior of those circuits. An attacker can potentially bombard a physical system in some way to elicit a response from the device.

An example of this type of signal injection was shown in the Cold War with 'The Thing'. [2] In 1945, a Soviet made Seal of the Republic was given as a gesture of friendship and installed in a sensitive office. When bombarded with radio waves, the device internals would resonate, modulate the radio waves, and it was possible to listen to conversations in the room where it was installed. This is a classical example of how physical systems can be manipulated through signal injection. In this case, the

signal injection was a desired feature, but it is important to be aware of this danger when designing physical systems.

Another example is disruption of GPS signals. This typically occurs because radio frequency signals are using the same wavelengths as GPS signals. GPS signals are usually weaker than RF signals, so the RF signals dominate and drown out the GPS signals. A report [3] was delivered in 2001 and details some of these risks and defenses associated with GPS interference, both unintentional and intentional.

To mitigate signal injection, it is important that system designers consider and plan for signal injection attacks. Defenses against this could include shielding equipment against magnetic and electrical fields or using multiple frequencies and receivers when possible [3]. These are just some techniques to defend against the signal injection threat which must be considered.

2.3.6 Signal Emissions

Adversaries may also attempt to harvest a physical systems signal emissions in an attempt to gather information. This is because during normal operation, many devices give off electromagnetic and radio signals, at least to some extent, even if unintended. There have been examples showing that it is possible to recreate what is on a user's CRT or LCD computer monitor by recording the emissions of the monitor from a far, using a process known as "Van Eck phreaking". [4] [5]

NATO created a program called TEMPEST to investigate and report on the risks associated with signal emissions and defenses against these threats. [6] Some of the easily implementable changes they suggest are to put electromagnetic shielding around devices. Suggestions presented also include signal filtering such that certain frequencies are attenuated or completely removed from emission.

2.3.7 Tampering

If a system is not protected, tampering with the system itself is one of the easiest attacks for an adversary to execute. He can attach logic probes or some device to record traffic being sent over signal buses to learn sensitive information. He could also tamper with certain portions of the system so that error handling and recovery routines were triggered, which might be easier to exploit in some way.

There are many different techniques to deal with physical tampering of a system. One of these includes potting, which involves sealing all components in a type of epoxy, so that no wires are exposed. Another technique would be to put the sensitive components in an enclosure that had some sort of alarm on it. When the enclosure was opened, an authority would be notified, who could then deal with the tampering. Physical tampering is a very large problem and these are just a few techniques to address it, but every system designer should consider how to protect his system from tampering.

3. CRYPTOGRAPHY OVERVIEW

3.1 Overview

Before delving into the details of various applications for secure physical system design, it is necessary to define and understand several different cryptographic primitives, as they form a foundation on which the applications build. The following sections present a brief introduction to the necessary cryptographic primitives that will be used in the rest of the thesis.

3.2 Encryption

It is often necessary to scramble and protect data so that only certain parties, such as those who possess a key value, can de-scramble and read the protected data. This might be necessary when sending any sort of sensitive data, such as financial records or e-mail messages. Presumably, if a person does not have the correct key value, he or she will not be able to scramble or unscramble the data properly.

The act of scrambling the data is called *encryption*. The corresponding act of descrambling encrypted data is called *decryption*.

Encryption and decryption operations and relevant parameters are denoted using the following notation below.

$$C = E_{K_E}(M) \tag{3.1}$$

$$M = D_{K_D}(C) \tag{3.2}$$

Above, C is the *ciphertext*, or encrypted text. M is the message or *plaintext*. E represents the *encryption algorithm*, of which there are several types. This algorithm takes plaintext as a parameter and returns ciphertext. D represents the *decryption algorithm*, which takes ciphertext and return plaintext. K represents the *key value*.

K_E is used with the encryption algorithm, while K_D is used with the decryption algorithm.

A sender would use his plaintext message to generate the ciphertext and transmit it. The receiver would then process the received data using the decryption algorithm and then be able to successfully recover the plaintext message.

3.2.1 Symmetric Encryption

Symmetric encryption is a fairly intuitive method of using encryption. In this style of encryption, both the sender and receiver share the same key value, K. In this scenario $K_E = K_D$. There are several, different symmetric encryption algorithms, such as AES [7], DES [8], Blowfish [9], and many others. For my purposes, I typically use AES. It is fast and considered fairly secure.

One difficulty with symmetric encryption is the establishment of a shared symmetric keys between the two communicating parties. If there is a secure channel between the sender and recipient, it is trivial to simply send the value K across the secure channel. If there is no secure channel however, there are protocols, such as the Diffie-Hellman Key Exchange [10] algorithm. This is a somewhat cumbersome step to do for every communication. Additionally, a party must maintain a different symmetric key for each other party he or she wishes to contact.

3.2.2 Asymmetric Encryption

Asymmetric encryption is an interesting cryptographic building block. In this system, $K_E \neq K_D$. Typically, K_E is a *public key*, while K_D is a *private key*. That is, K_E may be published somewhere publicly, which then allows anyone to encrypt messages. However, without K_D, these messages cannot be decrypted. As such, (typically) only one person will have K_D.

This is a useful system because it allows anyone to send a given person an encrypted message easily: Simply retrieve the public key, encrypt the message, and send

it to the recipient. Unlike symmetric encryption, there is no need for a protocol to establishing a shared symmetric key. This greatly alleviates the problems that key management systems impose.

There are several systems for asymmetric encryption schemes. One popular scheme is RSA [11]. In this scheme, a user picks a value which becomes K_D and uses that to derive K_E. It is considered unreasonably difficult to derive K_D from K_E though, which makes this scheme secure. Interested users may refer to [11] for more specific details on the RSA scheme.

3.3 Digital Signatures

An important ability when sending messages is for one party to "sign" the message. This allows a recipient to be confident that the sender is actually who he claims to be, much like a physical signature on a physical document. Due to the fact that electronic media is so easily manipulated, it is somewhat difficult to ensure that the sender of electronic materials is who he claims to be.

Several algorithms exist that can be used for digital signatures. In general terms, the signer will know some secret, or a private key, which he will use to sign messages. He will also publish some value related to the private key, known as the public key. When he wishes to send a signed message, the sender will use his private key to perform some operation on the message and send the message and result of the operation to the recipient. The recipient will then perform some operation on what he received from the sender, using the sender's public key. If the signature is authentic, the recipient will be able to determine this, or if not, this will also become more apparent. Figure 3.1 illustrates this process.

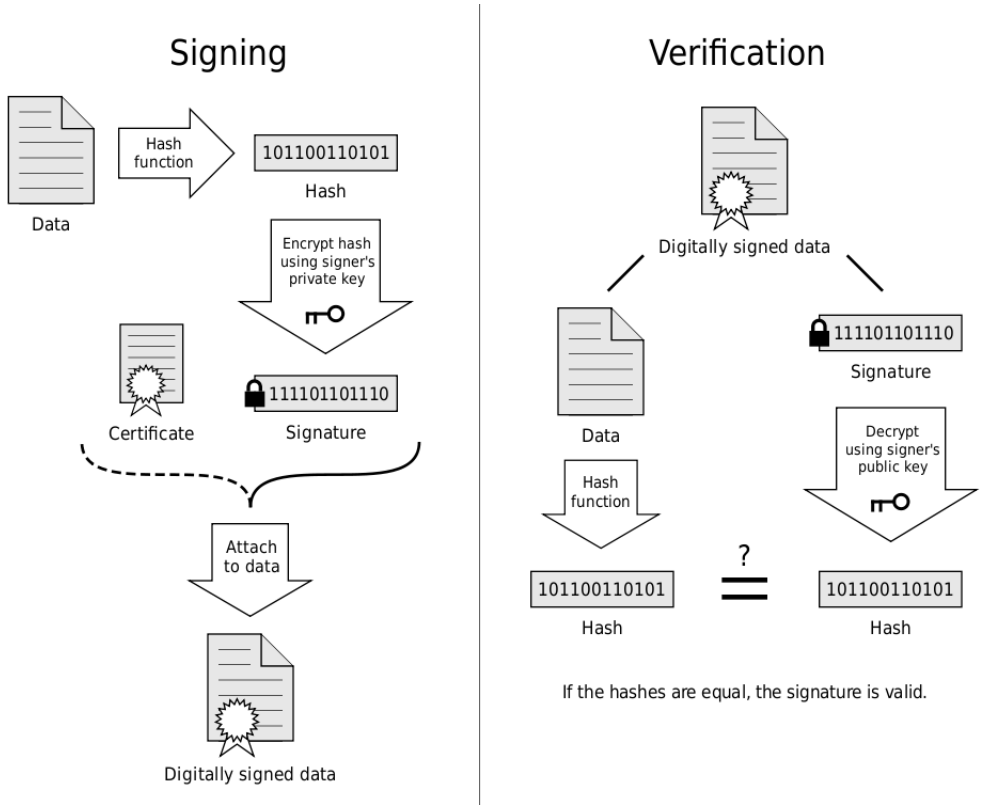

Figure 3.1. A graphical representation of the digital signature process [12]

3.4 Zero Knowledge Proof of Knowledge

A party might need to prove that it knows some value. Sometimes it is acceptable to simply reveal the value, but in other cases, this might not be acceptable. For instance, imagine if Peggy, Victor, and Eve are in the same room. Victor wants to make sure Peggy knows his phone number, but does not want Eve to learn her phone number. Peggy cannot simply repeat Victor's phone number, lest Eve overhear and write it down. In this case, Peggy will use what is called a *zero knowledge proof of knowledge* or ZKPK.

A ZKPK is a way to prove for one party to prove to another that it knows a secret without actually revealing the secret. A basic use case was just described, but there are many reasons why a ZKPK might be useful. A ZKPK can be used when there is concern that a man-in-the-middle is present and could determine the secret (as was

the case in the previous example), but to defeat this, encryption is usually a better alternative. Where a ZKPK is very useful is when the sender does not want to reveal the secret to the recipient. So in a similar example, say that Peggy wanted to prove to Victor that she knew her own phone number, but did not want to give it to Victor. This is when the use of a ZKPK is most applicable.

There are multiple different schemes that implement ZKPK behaviors. The scheme the author is most familiar with is called the Feige-Fiat-Shamir Identification Scheme (FFS). The scheme works using modular arithmetic, similar to the RSA algorithm above. Interested readers are referred to the original paper in [13] for more details.

3.5 Commitment Schemes

It is often necessary for a party to be bound to some decision it has made in the past, but without actually revealing that decision. For example, suppose Alice and Bob are talking on the phone and want to flip a coin to solve a dispute. This is clearly subject to much distrust, since whoever flips the coin could lie about the result so he or she always wins. This is where a commitment scheme would be useful.

More formally, a commitment scheme is defined as a two step procedure. In the first step, a value is chosen and *committed*. This *commitment* does not reveal the value chosen, so it may be publicly published somewhere if desired. The second step is *revealing*. In this step, the private value is revealed and then verified against the commitment previously made.

In the coin-flipping example, a commitment scheme could be employed as follows. Alice would secretly decide whether to call heads or tails. She would then *commit* this value to Bob. Recall that this commitment does not reveal Alice's choice. Bob would then tell Alice the result of the coin flip. Alice would then tell Bob what she committed previously. Bob checks this value against Alice's commitment. If the value checks out, Bob accepts that Alice actually did call that value.

Commitment schemes are often used in conjunction with zero knowledge proof

of knowledge schemes; A prover proves to a verifier that he knows the value used to generate a commitment without having to reveal to the verifier such value. The verifier only accepts the committed value. The method the author is most familiar with is known as Pedersen commitments. Readers interested in more details of this scheme are referred to the original paper [14].

4. PHYSICALLY UNCLONABLE FUNCTIONS

It is desirable for a user to be sure that the device that he is using is authentic. However, due to the sophistication of forgeries or possible communication tampering, a user might be suspicious that the system is the system it claims to be. A device called a Physically Unclonable Function, or PUF, is a technology that addresses this problem.

A PUF device provides a unique challenge-response capability. That is, when two PUFs are provided the identical challenge, they will each produce unique responses. In this way, a PUF, and the system it contains, can be identified by the response value it generates to a specific challenge. A more formalized definition of this relationship is given below.

$$PUF_1(C) = R_1$$
$$PUF_2(C) = R_2$$
$$R_1 = R_2$$

This relationship can thus be used to bind certain information to a given system by adding a PUF to it. That is, when a system produces a specific response, it is possible to unique identify that specific system from another.

4.1 Types of PUFs

A PUF device provides this sort of relationship by leveraging the physical properties of the materials in which it is instantiated. There are several different ways of doing this, from measuring the distortions of reflected light to leveraging the manufacturing inconsistencies from one chip to another.

The Ring Oscillator PUF is presented first and in somewhat greater detail than

other types of PUF since this is the type of PUF that the author worked with primarily. As such, it was incorporated in many of the different applications presented later in Chapters 5, 6, and 7.

4.1.1 Ring Oscillator PUF

A Ring Oscillator PUF is a PUF design that utilizes a circuit called a Ring Oscillator (RO). An RO is an odd number of inverter gates tied together. Because there are an odd number of gates, this will produce a continuously changing, or oscillating, signal. Because it is a combination of circuits, the RO PUF can be instantiated on a piece of silicon, such as an FPGA or ASIC device.

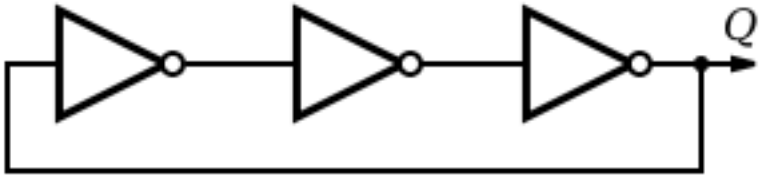

Figure 4.1. A 3-gate ring oscillator

Depending on the number of inverter gates being used as well as the propagation delay of every individual inverter, the output frequency of one RO may be different from another RO. In Figure 4.1, this output signal corresponds to the signal marked Q.

When used as part of a PUF, the unique behavior of an RO will be examined. Consider again the 3 stage RO shown in Figure 4.1. All three inverter gates are assumed to have the same propagation delay and the interconnecting wires are assumed to impose a negligible delay. However, in an actual instantiation of an RO, these assumptions are invalid. All three inverters should have the same propagation delay, but, due to uncontrollable manufacturing inconsistencies and tolerances, they do not. In a similar vein, the interconnecting wires will also impose a non-zero delay time in signal propagation. Both of these factors will combine so that even if two ROs are produced on the same manufacturing line, they will generate a slightly different

output frequency.

The slightly different output frequencies of two ring oscillators forms the basis of randomness for the Ring Oscillator PUF. Because the output frequencies of the ROs cannot be predicted, their actual frequency at run time gives a way to uniquely identify the individual PUF that contains them. In Figure 4.2, a more detailed diagram of a PUF based off of ring oscillators is presented.

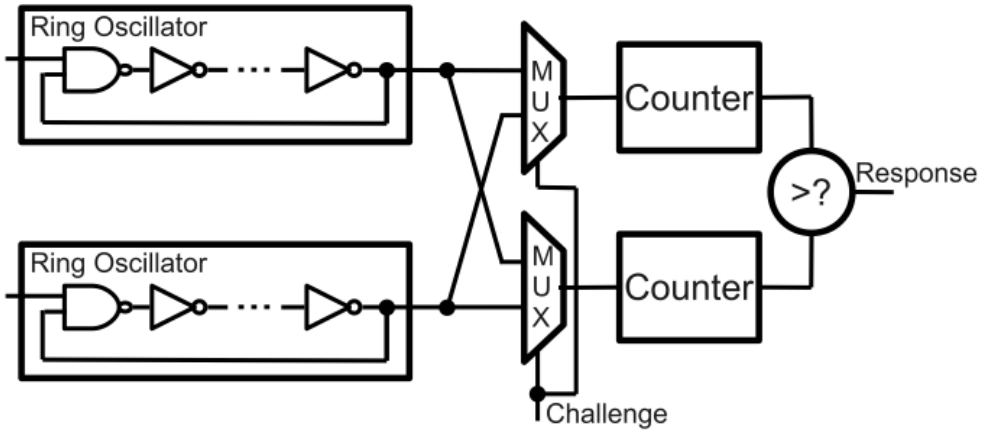

Figure 4.2. A 1-bit ring oscillator PUF

The ring oscillator PUF shown above uses a challenge bit and feeds it to a multiplexer. If the challenge bit is zero, the top ring oscillator will be fed to the top counter and the bottom ring oscillator to the bottom counter. If the challenge bit is one, the top ring oscillator will be fed to the bottom counter and the bottom ring oscillator will be fed to the top counter. The counters will then be executed for a given amount of time. At the expiration of this time duration, the values of the counters are compared. If the top counter has a larger total, a zero is output as the response. If the bottom counter has a larger total, a one is output as the response. While the diagram only displays two ring oscillators and only 1 bit of challenge and response, this diagram can be extrapolated to form arbitrarily large PUFs.

That is the most basic design of an RO PUF. In practice, this design is somewhat inefficient, since for an N-bit PUF, 2*N ring oscillators are needed, which is fairly

expensive. There has been work done for alternative designs of an RO PUF to reduce the number of ring oscillators needed [15]. Typically, this involves using a pool of ring oscillators and then using a multi-bit challenge to select some permutation of them. Details presented in [15] illustrate that 35 oscillators can be used to generate 133-bits of output by using this pool strategy.

4.1.2 Butterfly PUF

Another design of a PUF is called a Butterfly PUF. This design is similar to the previous RO design in that it can be instantiated on a piece of silicon. This allows for easy incorporation into existing FPGA designs or through the production of a custom ASIC chip. Figure 4.3 shows a circuit example of a butterfly PUF and how it might be designed.

The Butterfly PUF works by tying the output of two D flip flops to each others inputs. By applying the CLR signal to one flip flop and the PRE signal to the other flip flop, the circuit will enter an undefined state. It will eventually go to one of two defined states (0 or 1). The circuit will typically settle in the same state, which forms the basis for the PUF response.

Interested readers are referred to [16] for more information about the Butterfly PUF.

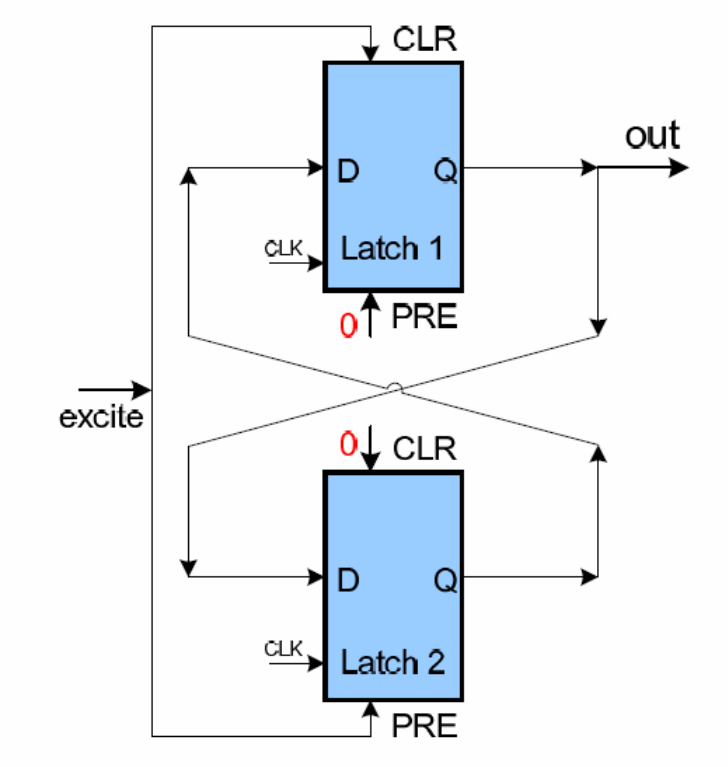

Figure 4.3. A depiction of a Butterfly PUF

4.1.3 Optical PUF

An optical PUF is a design that leverages physical randomness that is explicitly introduced during a manufacturing process. The typical optical PUF is constructed by taking a transparent material and randomly coating it with particles to disperse the light. A laser light is then shone on the material and the resulting pattern is recorded. The image is then processed and this is the response of the PUF. Interested readers are referred to the original thesis that proposed this idea in [17].

4.1.4 Coating PUF

A coating PUF works by creating a mesh of wires and then filling the cavities with some sort of dielectric material. Based on how the dielectric is applied, there will be varying levels of capacitance between the wires in the mesh. This capacitance

can then be measured as the unique response for the given PUF design. Details are presented in [18].

The coating PUF is typically used as a sort of anti-tamper device. The coating PUF is wrapped around an existing circuit and is used to enable its operation. That is, the proper PUF response is needed to unlock the device. If an adversary is to alter the coating in anyway (though reverse engineering for example), this will alter the PUF response and cause the underlying circuit to not function.

4.2 PUF Error and Error Correction

An important part to consider for any PUF device is the stability of its output for the same input. If the PUF device yields different outputs for the same challenge, the utility of a PUF is greatly reduced. As such, it is important to examine and investigate the stability and error rates of PUF. As the author worked primarily with RO PUFs, unless otherwise noted, this section refers to RO PUFs.

The basic use case that should be examined is when a PUF is executed twice in the same environment; that is, temperature, humidity, and other environmental factors are constant. For the RO PUF design in Figure 4.2, error rates can be reduced by increasing the time that the ring oscillators are executed. In this way, the faster ring oscillator's counter will clearly dominate the slower ring oscillator's counter. If the execution time is very brief, start-up times and routing delays may impose a noticeable difference and induce additional error. Note that when the author refers to increasing timer execution time, he is discussing orders of milliseconds. The ring oscillators were typically run at upwards of 100 MHz, so several milliseconds was enough time for frequency differences to become apparent.

A source of error that can be introduced is a change in temperature. Circuits will run either faster or slower as temperature changes, due to the changing resistance of the internal components. Note that this is not a behavior specific to PUFs, but electronics in general. As such, it is important to consider the effect of temperature

on PUF devices. Work in [19] presents detailed, empirical studies of temperature's effect on PUFs.

Circuit aging is another source of error that can potentially affect PUFs. Over time, certain pathways and routes of the PUF may change in their propagation delay. Since the PUF is predicated upon the same routes being used over and over, this can cause drastic problems for the PUF. At the least, aging can make a PUF more susceptible to other sources of error, but at worst case, it could cause enough bits of the PUF to change from their original values so that the PUF is no longer identifiable as the original PUF. Interested readers are referred to work in [20] which discusses PUF aging in greater detail.

4.2.1 Error Correction Schemes

As previously described, there are multiple different factors that can affect PUFs and their execution. If these are not mitigated, the functionality and utility of a PUF is greatly reduced. As such, an error correction scheme is typically needed when employing a PUF device.

Usually, the raw PUF response is not directly output, but rather, is fed into an error correction block. The error correction block processes the raw PUF output and removes any small errors that may be apparent and outputs the corrected response. This is diagrammed in Figure 4.4. While it is possible to do the error correction on a discrete chip separate from the PUF itself, it is safer to perform the error correction on the same chip as the PUF. This prevents an adversary from potentially intercepting the raw PUF output as it is transmitted to the error correction block. Note that in Figure 4.4, there is a notation that the PUF and the error correction are self contained to illustrate this.

The author has typically employed the use of Reed-Solomon (RS) error correction codes to do error correction. By adding t RS symbols, up to t bit errors can be detected, while $t/2$ bit errors can actually be corrected. This is a fairly large amount

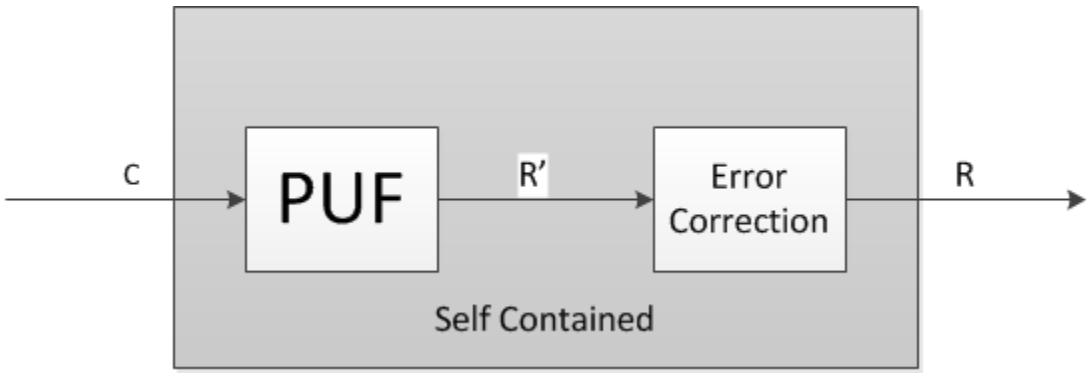

Figure 4.4. A high level concept of PUF error correction

of errors to correct. For a ring oscillator PUF, [21] found that inter-chip variation (that is, the same challenge) in response was approximately 0.86%. For a 128 bit PUF, that equates to around 1 bit of error per execution. As such, not a lot of error correction is needed, but some is indeed needed. In the author's work [22] [23] [24], he has typically used 32 bits of Reed-Solomon error codes. This allows for detection of up to 32 bit errors in the 128 bits and correct up to 16 bit errors, which is usually sufficient.

4.3 Vulnerabilities

Besides just correcting for benign errors due to the nature of PUFs, it is important to consider malicious tampering and how vulnerable the PUF is. Again, the use of PUF in this section refers to the ring oscillator design previously described unless otherwise noted.

There is a class of attacks called differential power analysis, *DPA*, that needs to be considered for PUFs. This involves monitoring the power consumption of a circuit during its execution. It is then possible to deduce what bits of data are being processed at a given time. For example, if the floating point processor takes a considerable amount of power in a micro controller, an attacker could monitor the power during execution. When he saw the power usage spike, he could infer that a floating point operation was taking place. This attack is detailed in several papers

and a fairly good overview is given in the book [25].

The ring oscillator PUF design is fairly resilient against this type of attack however. During execution of the PUF, every ring oscillator will be executing simultaneously as well as the other support circuitry. Because every execution utilizes the ring oscillators, power consumption will be constant. This thus eliminates the power differences needed for DPA.

Tampering with the PUF itself is a vulnerability that needs to be considered. An attacker may attempt to reverse engineer the PUF so that he could model and impersonate it. This would involve measuring the distances between wires, capacitance values, and propagation delays of the various elements of the PUF. However, these elements are so small (nanometers) that the invasive techniques for measuring them would likely alter their qualities, so any measurements taken would not be usable for impersonating the original PUF. As such, tampering can destroy a PUF and its usability, but the risk of being able to duplicate and impersonate the PUF is very very low.

Another important aspect that needs to be considered is if the PUF is tampered with at any point during the manufacturing or delivery process. The main concern is that an attacker could intercept the PUF and create a model of it; that is, the response for every possible challenge. It is possible to offset this risk however by using sufficiently large PUFs. If a 128 bit PUF is used, this sort of attack would require storing 2^{128} different values. There are estimated to be about 2^{80} atoms in the universe so this attack is not very realistic.

4.4 Comparison to Alternatives

Several technologies exist that fulfill roles similar to the of PUF technology. The Trusted Platform Module (TPM) and Radio Frequency Identification Tags (RFID Tags) are described below, as these are two very common technologies that are in practice today.

4.4.1 Trusted Platform Module

Trusted Platform Modules (TPM) are a technology that is similar, but different than PUFs. They are typically integrated into a computer's motherboard, though they are also present in other applications.

TPMs typically provide a large variety of services, as they are technically a cryptoprocessor. This is in contrast to a PUF device, which is just a provider of the challenge-response capability. TPMs are used to provide remote attestation, binding, and data sealing. These services leverage an *endorsement key*, which is installed in the TPM at manufacture time.

One area where TPMs and PUFs differ is that TPMs only have their one secret endorsement key to act as the key to all operations. If this is ever compromised, such as when it is being burned in at manufacturing time, all TPM security is lost. In contrast, PUFs behavior is characteristic of the challenge provided. If a sufficiently large key space is chosen, such as 128 bits, then it is unrealistic that an attacker could ever model the entire response space, which would encompass 2^{128} possible choices.

Readers interested in more about TPMs are referred to the TPM standards in [26].

4.4.2 Radio Frequency Identification Tags

Radio Frequency Identification Tags, or RFID, are devices that are exposed to radio waves, which then allow a reader device to read the information stored on the tag. They sometimes do not require power and can operate off the power from the reader itself.

The information stored on tags is typically personally identifiable information, such as ID numbers, location of origin, or related data. RFID tags are used in applications such as livestock inventory, shipping containers, or warehouse progress.

A security concern with RFID tags is that they can be read passively and silently by unintended parties. For instance, someone might carry a reader in his pocket and record the data about people walking by him. This is a security risk and there has

been some controversy over it.

RFID tags are a contrast to PUF. An RFID tag simply stores data and reads it out when queried. In contrast, a PUF is a function that requires a specific input to get the desired response. In this way, a PUF is thus more secure.

5. READ ONCE KEYS

5.1 Overview

In this section, we present the definition, design, and implementation of read-once keys (ROKs). The presentation is this chapter follows after [23].

The term read-once key (ROK) describes the abstract notion that a cryptographic key can be read and used for encryption and decryption only once. While it seems intuitive that a trusted piece of software could be designed that deletes a key right after using it, such a scheme naïvely depends on the proper execution of the program. This approach could be easily circumvented by running the code within a debugging environment that halts execution of the code before the deletion occurs. That is, the notion of a ROK entails a stronger protection method wherein the process of reading the key results in its immediate destruction.

ROKs could be applied in a number of interesting scenarios. One application could be to create one-time programs [27], which could be beneficial for protecting the intellectual property of a piece of software. A potential client could download a fully functional one-time program for evaluation before committing to a purchase. A similar application would be self-destructing email. In that case, the sender could encrypt a message with a ROK; the message would then be destroyed immediately after the recipient reads the message. More generally, there is considerable interest in self-destructing data, both commercially [28] and academically [29]. In addition, the use of trusted hardware tokens have been proposed for applications including program obfuscation [30], monotonic counters [31], oblivious transfer [32], and generalized secure computation [33]. ROKs can provide the required functionality for these applications.

Another interesting application of PUF ROKs is to defend against physical attacks

on cryptographic protocols. For example, consider fault injection attacks on RSA [34–38]. In these attacks, the algorithm is repeatedly executed with the same key, using a controlled fault injection technique that will yield detectable differences in the output. After enough such iterations, the attacker is able to recover the key in full. Similarly, "freezing" is another class of physical attack that can extract a key if it was *ever* stored in an accessible part of memory [39]. PUF ROKs offer a unique defense against all of these attacks because repeated execution with the same key cannot occur, and the key is *never* actually present in addressable physical memory.

The ability to generate ROKs in a controlled manner could also lead to an extension where keys can be generated and used a multiple, but limited, number of times. For example, consider the use of ROKs to encrypt a public key pk. If an identical ROK can be generated twice, the owner of pk could first use the key to create $e_{ROK}(pk)$ (indicating the encryption of pk under with the ROK). Later, an authorized party could create the ROK a second time to decrypt the key. Such a scheme could be used to delegate the authority to cryptographically sign documents.

In a sense, a ROK is an example of a program obfuscator. An obfuscator $\mathcal{O}$ takes a program $\mathcal{P}$ as input and returns $\mathcal{O}(\mathcal{P})$, which is functionally identical to $\mathcal{P}$ but indecipherable. A ROK, then, involves an obfuscator that makes only the key indecipherable. While ROKs are promising ideals, the disheartening fact is that program obfuscators–of which ROKs are one example–cannot be created through algorithmic processes alone [40]. Instead, trusted hardware is required to guarantee the immediate destruction of the key [27]. However, we are aware of no work that has specifically undertaken the task of designing and creating such trusted hardware for the purpose of generating a ROK.

Our insight for the design of such "PUF ROKs" is to incorporate the PUF in a feedback loop for a system-on-chip (SoC) design.[1] That is, our design is for the PUF to reside on the same chip as the processor core that performs the encryption. This

[1]Our design could also be made to work for application-specific integrated circuits (ASICs), but we limit our discussion to SoC designs for simplicity.

integration of the PUF and the processor core protects the secrecy of the key. An attempt to read the key from memory (given physical access) will fail, because the key *never exists in addressable memory*. Also, attempts to learn the key from bus communication will be difficult or impossible, as each key is used to encrypt only a single message, and the key is *never transmitted across the bus*.

The unpredictable nature of PUFs provides a high probability that each iteration of a ROK generation will produce a unique, seemingly random key. Yet, to ensure that a key can be generated to perform both encryption and decryption, the PUF must be initialized repeatedly to some state, thus providing the same sequence of keys. To accomplish this, Alice could provide an initial seed to produce a sequence of keys that are used to encrypt a set of secrets. Alice could then reset the seed value before making the device available to Bob. Bob, then, could use the PUF to recreate the keys in order, decrypting the secrets. As Bob has no knowledge of the seed value, he is unable to reset the device and cannot recreate the key just used.

Astute readers will note the similarities between our approach and using a chain of cryptographic hashes to generate keys. That is, given a seed x_0, the keys would be $\mathsf{H}(x_0)$, $\mathsf{H}(\mathsf{H}(x_0))$, etc., where H denotes a cryptographic hash function. The insight of our approach is that a PUF, as a trusted piece of hardware, can provide a hardware-based implementation that is analogous to a hash function, but is more secure than software implementations of such algorithms.

5.2 Read Once Keys (ROK)

Our formal notion of a ROK is based on an adaptation of Turing machines. Specifically, define the machine T to be

$$T =< Q, q_0, \delta, \Gamma, \iota >$$

where Q is the set of possible states, q_0 is the initial state, δ defines the transition from one state to another based on processing the symbols Γ, given input ι. Readers familiar with Turing machines will note that ι is new. In essence, we are dividing

the traditional input symbols into code (Γ) and data (ι). For the sake of simplicity, we assume that ι only consists of messages to be encrypted or decrypted and ignore other types of input data. Thus, the definition of δ is determined by the execution of instructions $\gamma_1, \gamma_2, \ldots, \gamma_i$, where consuming $\gamma_i \in \Gamma$ results in the transition from state q_i to q_{i+1}. Based on this formalism, we propose the following primitives.

- The **encrypt primitive** $\mathsf{Enc}(\gamma_i, q_i, m)$ encrypts the message $m \in \iota$ given the instruction γ_i and the state q_i. The system then transitions to q_{i+1} and produces the returned value as $e(m)$ as a side effect.

- The **decrypt primitive** $\mathsf{Dec}(\gamma_j, q_j, e)$ decrypts the ciphertext $e \in \iota$ given the instruction γ_j and the state q_j. If the decryption is successful, the primitive returns m. Otherwise, the return value is denoted $\emptyset$. The system then transitions to q_{j+1}.

Informally, γ_i and q_i describe the current instruction and the contents of memory for a single execution of a program, and capture the state of the system just before executing the encrypt or decrypt primitive. That is, if the execution of the program is suspended for a brief time, γ_i, q_i would describe a snapshot of the stack, the value stored in the instruction pointer (IP) register, the values of all dynamically allocated variables (*i.e.,* those on the heap), etc. In short, it would contain the full software image for that process for that precise moment in time. Once the program is resumed, the symbol γ_i would be consumed, and the system would transition to state q_{i+1}. Given these primitives, we present the following definition.

Definition: A **read-once key** (ROK) is a cryptographic key $\mathcal{K}$ subject to the following conditions:

- Each execution of $\mathsf{Enc}(\gamma_i, q_i, m)$ generates a new $\mathcal{K}$ and yields a transition to a unique q_{i+1}.

- The first execution of $\mathsf{Dec}(\gamma_j, q_j, e)$ returns m and transitions to q_{j+1}. All subsequent executions return $\emptyset$ and transitions to q'_{j+1}, even when executing the machine $< Q, q_0, \delta, \Gamma, \iota >$ with e, except with negligible probability.

- The probability of successfully decrypting e without the primitive $\mathsf{Dec}(\gamma_j, q_j, e)$ is less than or equal to a security parameter ϵ $(0 < \epsilon < 1)$, even when given identical initial states. ϵ must be no smaller than the probability of a successful attack on the cryptographic algorithms themselves.

What these definitions say is that the ROK Turing machine is non-deterministic. Specifically, during the first execution of a program[2] that encrypts a message m, δ will define a transition from q_i to q_{i+1} based on the primitive $\mathsf{Enc}(\gamma_i, q_i, m)$. However, the second time, the key will be different, and the state transition will be from q_i to q'_{i+1}. Similarly, the first execution of a program that decrypts $e(m)$ will traverse the states $q_0, \ldots, q_j, q_{j+1}$, where q_{j+1} is the state that results from a successful decryption. However, returning the machine to its initial state q_0, using the same instructions Γ, the state traversal will be $q_0, \ldots, q_j, q'_{j+1} = q_{j+1}$, because the decryption fails. Thus, ROKs incorporate some unpredictable element that does not exist in traditional Turing machines: the history of prior machine executions. That is, for any given machine T, only the first execution (assuming either the encrypt or decrypt primitive is executed) will use the transitions defined by δ. The second (and subsequent) executions will use δ', as the state after the primitive is invoked will differ.

Clearly, these definitions capture the intuitive notion of a ROK. The key $\mathcal{K}$ is generated in an on-demand fashion in order to encrypt a message. Later, $\mathcal{K}$ can be used to decrypt the message, but only once. After the first decryption, the key is obliterated in some manner. Specifically, even if the contents of memory are returned to match the program state γ_j, q_j as it existed before the first call to $\mathsf{Dec}(\gamma_j, q_j, e)$, the

[2]Observe that the program doing the encryption is separate from the one doing the decryption. If the encryption and decryption occurred in the same program, the decryption would succeed, as the key would have just been dynamically generated. In contrast, when the programs are distinct, only the first execution of the decryption program will succeed.

decryption will fail. The intuition here is that a special-purpose hardware structure must provide this self-destructing property.

Observe that an adversary $\mathcal{A}$ may opt to attack the cryptographic algorithms themselves. In such an attack, the number of times the key $\mathcal{K}$ can be read by an authorized party is irrelevant: $\mathcal{A}$ is never authorized. If the cryptographic scheme is sufficiently weak, $\mathcal{A}$ may succeed in recovering the message (or the key itself). The ROK property offers no additional security against such an attack. That is, we are making no special claims of cryptographic prowess. For this reason, we require that ϵ be no smaller than the probability of a successful attack on the cryptographic scheme employed.

What is unique about our technique is that we are offering a means to limit the usage of a key by an authorized party. Clearly, with sufficient motivation, this authorized party may become an adversary himself, attempting to recover the key $\mathcal{K}$ and subvert the system. The parameter ϵ offers a means to specify the system's defense against such an insider threat. For the most sensitive data, an implementation of our design could require a very low level of ϵ, making the probability of subverting the ROK property equal to the probability of a brute-force attack on the cryptographic algorithm. In applications that are less sensitive (*i.e.,* the ROK property is desirable, but not critically important), ϵ could be larger. In short, ϵ captures the flexibility to adjust the security guarantees of the ROK according to desired implementation characteristics.

5.3 PUF-based ROKs

Figure 5.3 shows a block level diagram of a basic PUF rok design. It consists of several different components. The Processor Core (PC) is what interacts with the computer itself and the internal components of the PUF ROK. It also handles the various input and output tasks required. The PC is connected to the Cryptography Core (CC), which is responsible for performing the various cryptographic operations

as well as communicating with the internal feedback loop. The internal feedback loop consists of a register wired to the PUF which is wired to an error correction unit, which is then in turn wired back to the register and the CC.

The CC is a stand-alone hardware component that provides cryptographic services to the PC. The CC provides the following service interface to the PC:

- $\mathsf{Init}(x_0)$: an initialization routine that takes an input x_0 as a seed value for the PUF. There is no return value.

- $\mathsf{Enc}(m)$: an encryption primitive that takes a message m as input and returns the encrypted value $e(m)$.

- $\mathsf{Dec}(e(m))$: a decryption primitive that takes a ciphertext as input. This service returns the plaintext m only on the first execution. Subsequent calls to this service return $\emptyset$.

Upon a call to the encryption function, the PUF is executed with the contents of the register, error correction is stored, and then the response overwrites the contents of the register. The response is also passed back to the cryptography core where it can be used to generate an encryption key. This feedback loop ensures that once a key has been used once, it cannot be used again, since it has been overwritten in the register.

When decryption is desired, the Init function must be used to re-seed the device so that the proper value is in the register. Then, when Decrypt is called, the value in the register will be used as the challenge to the PUF, any errors will be corrected by the error correction unit, and the result will overwrite the register and also be passed into the CC. The response will then be used to derive the decryption key for the message.

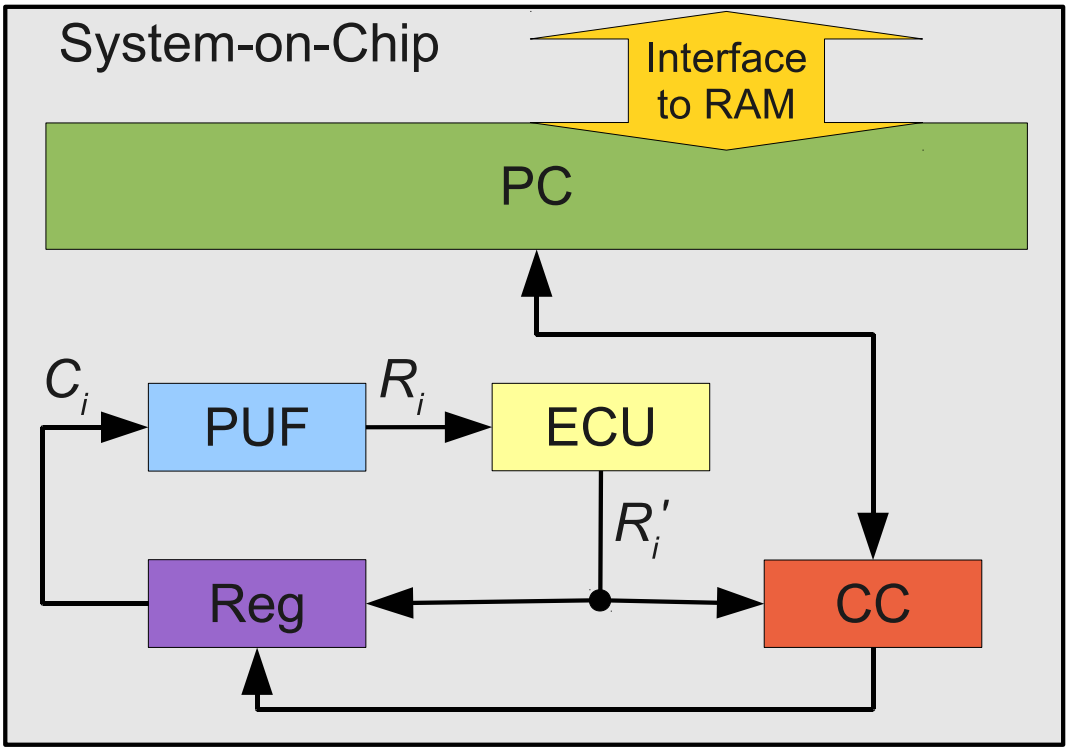

Figure 5.1. Block level view of a basic PUF ROK device

Note that the CC uses the error corrected PUF response to derive a key, but does not use it as a key directly. Typically a hash algorithm is applied to the key first. This prevents the PUF from potentially being modeled, as described in Chapter 4.

5.3.1 PUF ROKs as a Physical System

Because of their nature, PUF ROKs would be considered a peripheral physical system. This is because they rely on an interaction between themselves and a computer to augment the functionality of the computer; they do not necessarily interact with the world just by themselves.

As a peripheral physical system, the PUF ROK must be resilient against various types of environmental attacks, such as freezing and power analysis, as discussed in the next section, but they also must be very aware of the interactions they have with a host system and the dangers that can pose.

5.4 Security Considerations

For our security analysis, we consider the case of a probabilistic polynomial-time (PPT) attacker $\mathcal{A}$, with two goals. First, the goal of $\mathcal{A}$ is to recover just the key used to encrypt or decrypt a single message. The second goal considered is to model the PUF, which would enable the attacker to emulate the PUF ROK in software, thereby negating the hardware ROK guarantee. Initially, in both cases, we assume the adversary is capable of (at most) eavesdropping on bus communication. That is, the adversary is unable to observe communication between the cores in the SoC design.

Under this model, $\mathcal{A}$ is able to observe the data passing between the PC and memory, or between the PC and a network. Observe, though, that these messages consist exclusively of the plaintext m and the encrypted $e(m)$. Thus, the attack is a known-plaintext attack. However, this information offers no additional knowledge to $\mathcal{A}$. Even if $\mathcal{A}$ managed to reconstruct the key $\mathcal{K}$ (with negligible probability under the PPT model), this key is never used again.

The only use of reconstructing $\mathcal{K}$ in this manner is to attempt to reverse engineer the behavior of the PUF. However, recall that our design involved hashing the PUF output when creating the keys. Consequently, $\mathcal{K} = \mathsf{H}(R_i)$, where H is a robust cryptographic hash function. As a result, $\mathcal{A}$ again has only a negligible probability of reconstructing R_i. Yet, we can take this analysis even further, because R_i by itself is useless. That is, $\mathcal{A}$ would also need to know the corresponding C_i (or R_{i+1}) to begin to model the PUF. Thus, $\mathcal{A}$ would have to accomplish a minimum of *four* feats, each of which has only a negligible probability of occurring. Thus, we do not find such an attack to be feasible.

To continue the analysis, we loosen our assumed restrictions and grant $\mathcal{A}$ the ability to probe inside the SoC and observe all data transferred between the cores. Clearly, such an adversary would succeed, as the data passed between the PUF and the CC occurs in the open. However, this attack model is so extreme that only the most

dedicated and motivated adversaries would undertake such a task. Similarly, users who are faced with such powerful adversaries are likely to have extensive resources themselves. Thus, these users are likely to shield the processor using known tamper-resistance techniques, and we find this threat to be minimal.

Moving away from the PPT model, we can return to the discussion of fault injection [34–38] and freezing [39] attacks. Fault injection attacks fail to threaten the confidentiality of the system, because these attacks are based on repeatedly inducing the fault with the same key. However, PUF ROKs can only be used once. At best, a fault injection would become a denial-of-service, as the key would not correctly encrypt or decrypt the message. Freezing attacks are similarly unsuccessful, because they operate on the assumption that the key existed in addressable memory at some point. However, that is not the case with PUF ROKs. These keys are generated dynamically and are never explicitly stored outside the processor itself. Thus, PUF ROKs offer robust defenses against these physical attacks.

One final class of attacks to consider is power analysis [41]. Simple power analysis (SPA) involves monitoring the system's power fluctuation to differentiate between portions of cryptographic algorithms. This information leakage can reveal how long, for instance, a modular exponentiation takes, which reveals information about the key itself. Differential power analysis (DPA) observes the power fluctuations over time by repeatedly executing the cryptographic algorithm *with the targeted key.* Ironically, DPA is considered harder to defend against than SPA. And yet, PUF ROKs are immune to DPA (since repeated execution is not allowed) while vulnerable to SPA. Even though SPA is a potential threat, known techniques can prevent these attacks [42].

5.5 Out of Order Execution

One limitation of the basic PUF ROK design in 5.3 is that it does not allow out of order execution. That is, if five messages are encrypted and then the third is requested to be decrypted, the PUF ROK must be re-seeded, the PUF cycled twice,

and then the third cycle can be used to actually perform the decryption. To decrypt the first message, the PUF would again need to be re-seeded.

Supposing this had to be done many times, this process would quickly become cumbersome. As such, it is desirable to have a PUF ROK system that allows out of order execution. Figure 5.5 shows the block level design for a PUF ROK that allows this out of order execution.

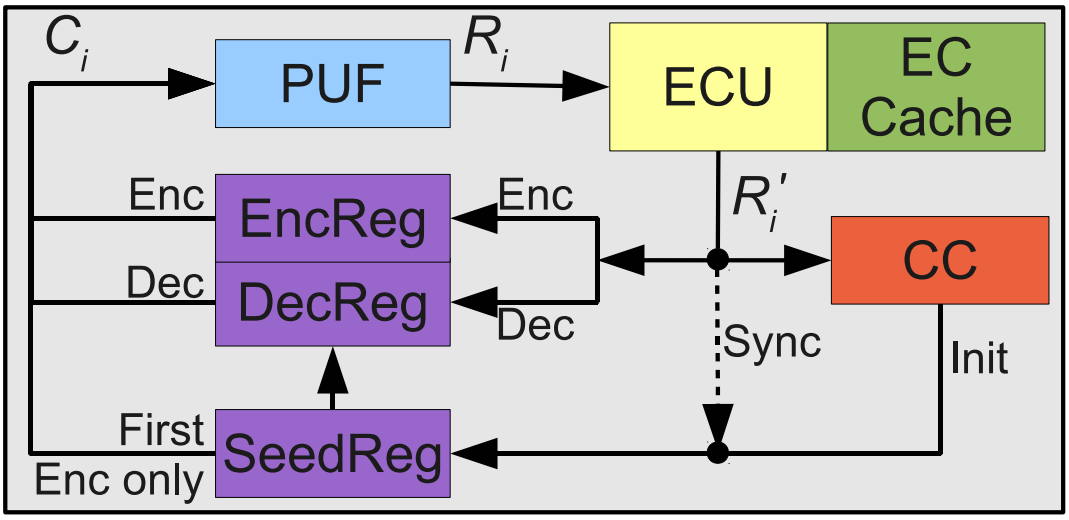

Figure 5.2. Block level view of a PUF ROK allowing out of order execution

The modified PUF ROK is able to perform out of order executions by replacing the one original register with three new registers, a seed register, an encryption register, and a decryption register. A cache is added to the error correction unit as well. Note that all these new components are still internal to the PUF ROK design, so that no buses are exposed externally. The new design requires a new parameter, N. This parameter specifies the number of keys that will be stored in the error correcting cache. The PUF ROK can then perform out of order execution on up to N different keys. The new design also introduces the Sync action, which is used to update the seed register and is further described below.

Upon the initial seeding of the PUF ROK, the seed value is stored in the seed register. When the first encryption is requested, the seed register is fed into the PUF. The response is then passed through error correction and stored in both the error

correcting cache as well as the encryption register. Note that the seed register is not updated here. Upon request for another encryption, the contents of the encryption register will be used, rather than the seed register.

Upon a request for a decryption, if the requested key is still marked as valid in the error correcting cache, the seed register is copied into the decryption register. The PUF is then cycled and writes back to the decryption register enough times for the proper response to be obtained. Note that the error correcting unit is correcting any potential errors after each cycling of the PUF. At the conclusion of this, the requested key is marked as used in the error correcting cache, meaning the PUF ROK will not use it again.

Because the error correcting cache has a finite amount of space, it will be necessary to clear the cache from time to time. This is done using the Sync action. Sync is triggered when the first key in the cache has been marked as invalid. (Note that this first key will be associated with the value currently in the seed register.) Since the values are invalid, this means that they will never be used again, so the value in the seed register is obsolete and can be updated. The error correction cache thus takes control of the feedback loop. It decides which key is the last used, and then cycles the PUF, using the contents of the seed register that many times and writing the results back into the seed register. For example, if there are 4 values stored in the cache and values 1 and 2 are invalid, the PUF will be cycled twice, with the resulting response being written into the seed register.

5.6 Implementation

For a prototype implementation, the Saxo-L board from KNJN.com was used. [43] It contains an Altera Cyclone FPGA and an NXP LPC2132 ARM processor. The two chips are connected together by a Serial Peripheral Interface (SPI). Additionally, a USB and a JTAG port are available, which makes for easy communication with the various chips.

For the ease of development, we implemented only the PUF and the register on the FPGA and then implemented the error correction unit, cryptography core, processor core, and other supporting computation on the ARM chip. When the PUF or register was needed, the ARM would issue a request over the SPI link to access the appropriate component. Figure 5.6 shows details of the implementation graphically.

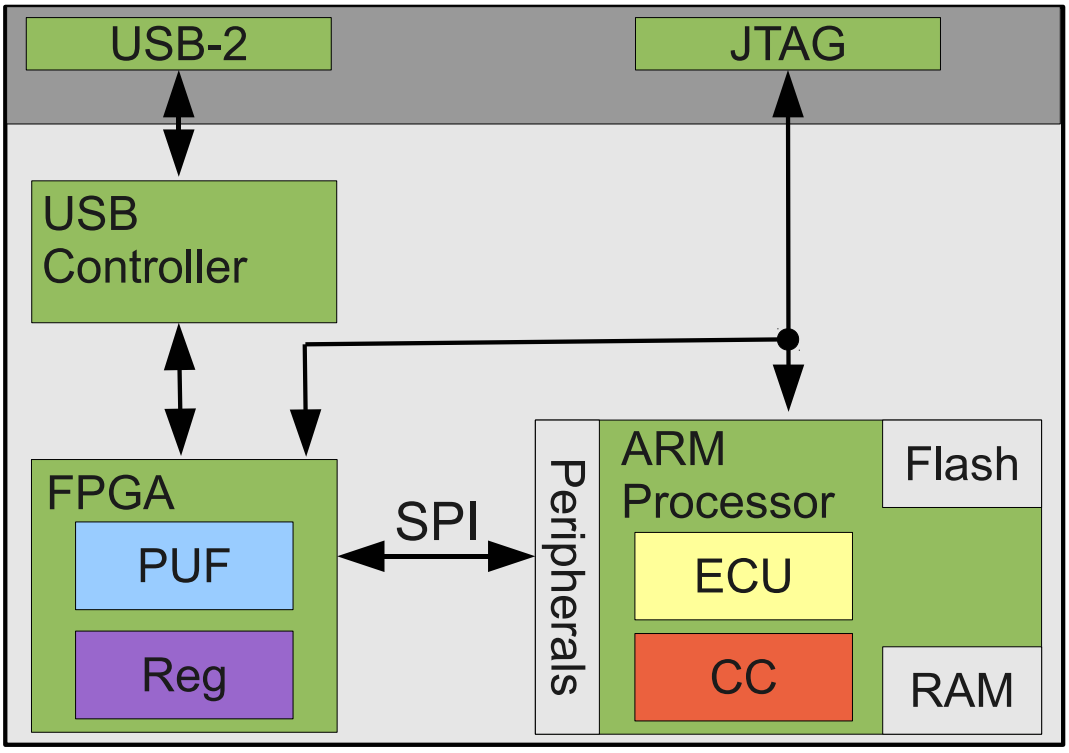

Figure 5.3. Implementation of a ROK device

The Saxo-L board is 44 x 60 mm, which makes it very portable. A production quality device would likely be smaller. This would allow the ROK to be implemented as a small dongle that could be plugged into a USB port potentially.

For the software portion of the project, the PolarSSL [44] library was used, which is an SSL library specifically optimized for small microprocessors, such as the LPC2132.

5.6.1 Limitations

There are some limitations to the implementation, since it is simply a prototype. The fact that the SPI bus is exposed is a huge problem. As it currently exists, an attacker could simply attach logic probes to the bus and intercept or modify any traffic between the FPGA and ARM chip. As such, he would be able to manipulate the values of the register as well as what the error correction unit receives. Clearly, this is not a good thing.

This vulnerability could be mitigated by using some sort of tamper proofing, such as potting, but this is a relatively expensive solution. Instead, the ideal solution would be to incorporate all the different components on one chip, as shown in the original design. There are soft core ARM processors available which can be instantiated on an FPGA already. It would be possible to simply move the entire ARM processor onto the same chip as the PUF and register. Not only would this be more secure, but it would most likely be less expensive to manufacture a device with only one chip, rather than two.

There are a variety of soft core microprocessors available, depending on the brand of FPGA selected, so there are alternatives available to the ARM architecture.

5.6.2 Results

Our PUF design consisted of 32 1-bit ring oscillator PUFs. Each of these circuits consisted of a ring oscillator constructed from 37 inverting gates. In our experiments, we found that using fewer than 37 gates yielded less consistency in the PUF behavior. That is, smaller PUFs increase the number of bit errors that must be corrected. The output from the ring oscillators was linked to 20-bit counters that were controlled by a 16-bit timer. The timer was synchronized with a 24 MHz clock, indicating that the timer would expire (as a result of an overflow) after 2.73 ms. When the timer expires, the values in the counters are compared, producing a 1 or 0 depending on which counter had the higher value. This design used 2060 of the 2910 (71%) logic

cells available on the FPGA. Each execution of the PUF produced 32 bits of output. Consequently, to generate larger keys, the ARM processor polled the PUF multiple times, caching the result until the key size was met.

To put the performance of the PUF into perspective, we compared the execution time with measurements [45] reported by NXP, the device manufacturer. Some of NXP's measurements are reported in Figure 5.1. As each PUF execution (producing 32 bits of output) requires 2.73 ms to overflow the timer, it is slower than encrypting one kB of data in AES. Observe, though, that larger PUFs would still only require 2.73 ms. Consequently, the overhead of executing the PUF can remain small, especially as large amounts of data are encrypted or decrypted.

Table 5.1 NXP cryptographic measurements

Symmetric Algorithm	Time (ms/kB)	RSA Operation	Time (s)
AES-CBC	1.21	1024-bit encrypt	0.01
AES-ECB	1.14	1024-bit decrypt	0.27
3DES-CBC	3.07	2048-bit encrypt	0.05
3DES-ECB	3.00	2048-bit decrypt	2.13

The comparison the RSA encryption and decryption is stark. Observe that the 2.73 ms required to execute the PUF is 27.3% of the time to perform a 1024-bit encryption in RSA. As the key size increases (assuming the PUF size is increased accordingly so that only one polling is needed), the PUF execution time becomes 0.13% overhead for 2048-bit RSA decryption. Thus, the performance impact of polling the PUF during key generation is minimal.[3]

[3]Obviously, there is additional work required to convert the PUF output into a usable key. However, the precise timing of this work is implementation-dependent, and the algorithms typically employed are significantly more efficient than the modular exponentiation. As such, we focus solely on the PUF measurement in our analysis.

6. PHYSICALLY ENHANCED AUTHENTICATION RING

6.1 Overview

In this section, we present the definition, design, and implementation of the PEAR system. The presentation is this chapter follows after [22].

One problem that is present when using computers is that users typically are not aware of the security of the system they are using. For instance, an attacker could have installed a key logger on a user's system to harvest every user name and password they have. Even with the best security systems on the machine in place, if the attacker is able to capture a user's keystrokes, the other security is moot.

A way to prevent this type of attack is by using an external device or an alternate channel to enter sensitive information, such as passwords or credit card numbers. In this way, if a key logger or the original system is compromised, the attacker will not be able to recover those passwords, credit cards, or other sensitive information.

PEAR, or Physically Enhanced Authentication Ring, was designed to counteract this key logger threat to a system. In addition to defending against key loggers specifically, it increases security in general because it is the second part of a "two factor authentication" system. It also is a physical system, specifically a peripheral physical system, since it incorporates its own processing and interacts with the user's normal computer system. Thirdly, the PEAR system incorporates a PUF device, so it is a good example of when PUF technology is useful.

From a high level perspective, a PEAR device is a device consisting of a PUF, a keypad, and some supporting circuitry. When a user wishes to log on to a given service, rather than using the keyboard for a password, he enters a 4 digit PIN on the PEAR device. The PEAR device then executes the PUF and then initiates a zero-knowledge proof of knowledge with the service provider. Note that no sensitive

data is actually input to the PC, which potentially has a key logger. Any data that the PC is requested to ferry between the PEAR device and the service provider is encrypted, so recording this data does not reveal any information.

The typical use case is that a user requests a PEAR device from a service provider, such as a bank. The bank then configures and mails the PEAR device to the user. The user sets his PIN number on the device and completes the enrollment protocol. Then, when he desires to use the service, he requests the authentication protocol. He enters his PIN into the PEAR device and the device then executes the rest of the authentication protocol. If successful, the service provider then allows the user to access the service. Note that if a user already has a PEAR device from another service provider, he can easily use the same device for another service; he simply must re-execute the enrollment procedure and enter a new PIN for the new service (or use the old PIN).

The system works by having every service provider associated with an ID number of some kind. Each user of the service will also have an ID number associated with it. This allows both parties to identify themselves to each other.

6.2 Protocol Details

The PEAR system consists of two parts, an enrollment step, which is executed once initially and then an authentication step, which is executed every time the user desires to use the service. Table 6.2 presents a formalized description of the protocols, while Figure 6.1 and Figure 6.2 give graphical representations of the different stages occurring.

As seen in the diagrams below, the four players in the PEAR system are the user himself, the PEAR device, the user's computer, and the service provider. Note that the user's computer and the service provider are assumed to be connected over the Internet.

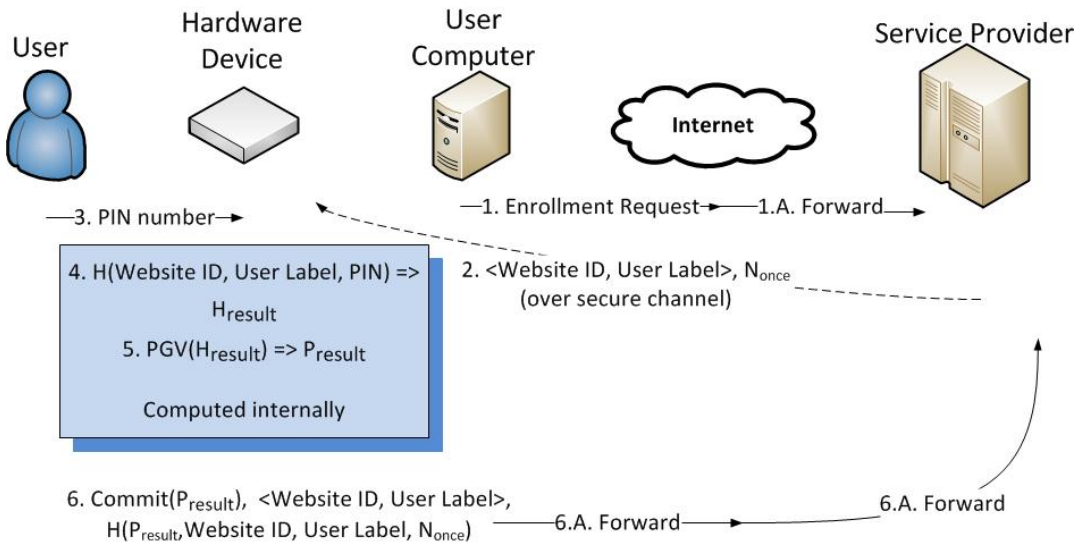

Figure 6.1. The enrollment stage of PEAR

The user initially requests an enrollment procedure from the service provider. The service provider then sends the tuple of $< WebsiteID, UserLabel >$ and a nonce to the PEAR device directly, over a secure channel. The user then enters his new PIN number to the PEAR device. Once the PEAR device has these different pieces of information, it is able to execute the steps contained inside the blue box in the diagram. All the pieces of information are hashed together. The resulting hash is then used as input to the Password Generator and Verifier (PGV). Note that a PUF is an acceptable PGV. The results from the PGV are then used in a commitment protocol. In addition to sending the results of the commitment to the service provider, the device also sends the $< WebsiteID, UserLabel >$ tuple and a hash of the tuple combined with the committed value.

An interesting point to note is that during the enrollment stage, an "out of band" communication is required to deliver the combination of the service provider's ID, the user's corresponding ID for that service, and a nonce value. This could be done by installing these values on the hardware device before it is given to an end user. For instance, if PEAR was being used with a bank, the bank might install these values before mailing the device to the user. If the user was adding a new service to a PEAR device he already had, the tuple of $< WebsiteID, UserLabel >$ and the nonce

might be delivered through post or given over the phone to the user. The key point is that they are not delivered over the same channel as will be used for the rest of the protocols (such as the Internet), since if an attacker is able to recover these values, he would be able to make a fraudulent account.

Table 6.1 Formalized version of the PEAR protocols

$\mathsf{Enroll}(U)$ — Device T (using input data from user U) computes a commitment and enrolls the results with S.
• C requests enrollment from S • S sends the tuple <Label, ID> and nonce N to T over a secure channel • U sends PIN to T • T computes H(ID, Label, PIN) as H_{result} • T executes $\mathsf{PGV}(H_{result})$ as P_{result} • T sends $\mathsf{Commit}(P_{result})$, <Label, ID>, $\mathsf{H}(\mathsf{Commit}(P_{result})$,Label,ID,$N)$ to S, via C
$\mathsf{Authenticate}(U)$ — Device T (using input data from user U) authenticates itself as a registered user of S.
• C initiates the authentication request from S • S sends the tuple <Label, ID> and $\mathsf{Chal}(P_{result})$ to T • U sends PIN to T • T computes H(ID, Label, PIN) as H_{result} • T executes $\mathsf{PGV}(H_{result})$ as P_{result} • T responds with $\mathsf{Prove}(P_{result})$, which C forwards to S

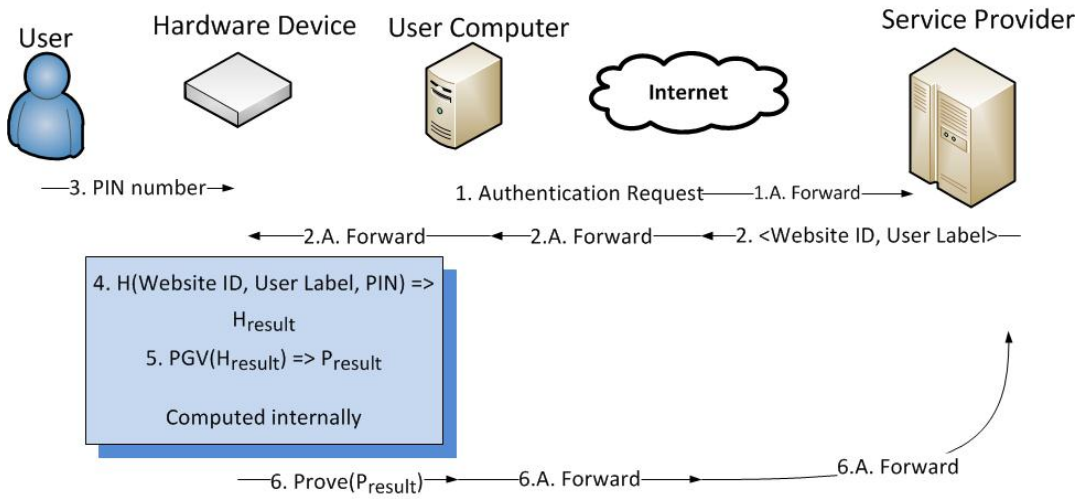

Figure 6.2. The authentication stage of PEAR

6.3 Additional Usage Scenarios

In addition to allowing authentication over an insecure channel as originally intended, PEAR also provides two other beneficial usage scenarios.

6.3.1 Unlinkability Property

Currently, users may be enrolled under several service providers for various reasons. Each service provider might hold some pieces of sensitive information about the user, but not necessarily every piece of sensitive information. However, if the two service providers were to collude, they would be able to learn a lot more about an individual user, which is not desirable.

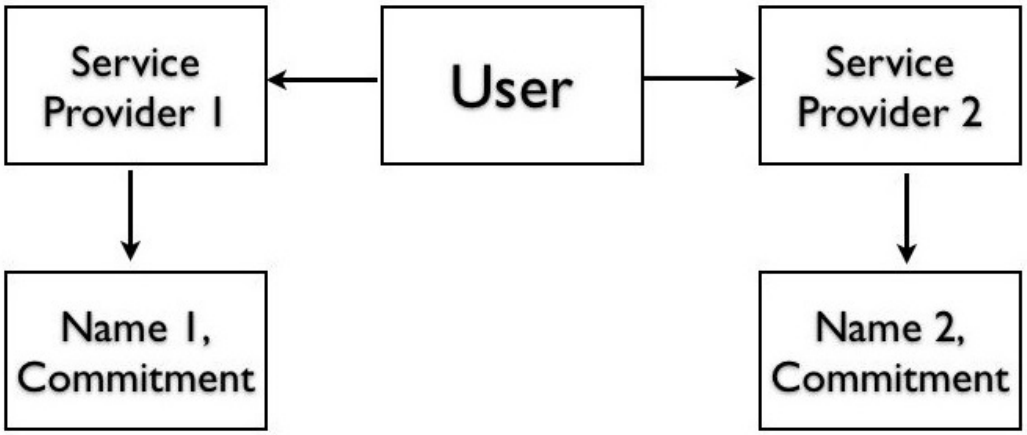

Figure 6.3. The unlinkability property of PEAR

As such, it is desirable to only give information out in a way such that service providers cannot collude to violate a user's privacy. The PEAR system provides this capability. When enrolling with two different service providers, the user would use a different screen name with each and send a commitment. Because the commitments do not leak any information (and the website IDs and user IDs are different), even if the service providers colluded, they would not be able to determine which pair of accounts belong to the same user very easily.

6.3.2 Organizational Anonymity

Another issue that frequently arises is that users may need to contact a party outside of their organization, but they do not want to be individually identified; they wanted to be recognized as a university member only. For instance, a Purdue student may wish to access the ACM digital library. She does not want to register with the library as an individual; she simply wants access granted due to her Purdue affiliation. However, the university, or whoever the authority is, still needs to maintain some sort of auditing capability.

PEAR is able to fulfill this use case as well. Users will use a PEAR device to enroll under the authority (the university in the previous example). When they desire some

service (such as digital library access), users will use PEAR to authenticate to the authority, who will then authenticate to the service provider on their behalf.

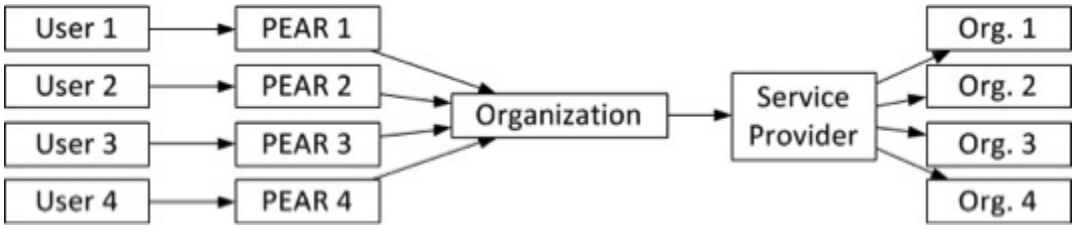

Figure 6.4. The organizational anonymity aspect of PEAR

In this way, users remain anonymous to the outside world and external service providers. However, if a problem occurs, the authority can still hold users accountable for any actions they took.

6.4 Security Considerations

Several lemmas are presented below which address various different security aspects of the PEAR system. Following the lemmas is a discussion of some of the different security issues facing physical systems that were discussed in Chapter 2 and how the lemmas relate to those concerns.

6.4.1 Lemmas

Lemma 1.

A man-in-the-middle attacker cannot recover any useful data communicated over the network between the service provider and the computer.

Proof: The only data that is transmitted between the computer and network is the tuple containing the service ID and the user's ID initially and then steps of the zero-knowledge proof (see Figures 6.1 and 6.2). The tuple will only be sent during authentication, so we can assume that users are already enrolled. An attacker gains nothing by intercepting the tuple during authentication, since it still requires both the user PIN number and the device itself to impersonate a user. Intercepting the steps

of the zero-knowledge proof also gives him no information since these zero-knowledge protocols do not reveal any information about the committed value.

Lemma 2.

A man-in-the-middle attacker cannot recover any useful data communicated between the user computer and the device.

Proof: As shown in Figures 6.1 and 6.2, the only data that is being transmitted over this channel is the tuple from the server and the zero-knowledge steps. As shown in Lemma 1, an attacker cannot gain any useful information from this. Also note that the PUF secret is never transferred outside of the device, but rather a commitment or proof is sent. As such, a MITM attack would not reveal the user's secret, but only the various steps of the zero-knowledge proofs, which are secure against MITM attacks. In addition, the service provider does not even ever know the user's PIN.

Lemma 3.

An active man-in-the-middle attacker cannot recover any useful information by modifying data between the device and computer or computer and network during the authentication stage.

Proof: An attacker who modifies the tuple being sent to the device or computer from the network would cause the device to create an incorrect zero-knowledge proof. This would disrupt the user's ability to authenticate. However, the attacker would not be able to glean any information from the proof generated from this modified tuple, due to the use of the zero-knowledge proof. Note that an attacker would be able to recover useful information if it could modify the tuple during enrollment. It could substitute a malicious tuple for the valid tuple, which would cause users to be authenticating to the MITM, rather than the service provider. We avoid this problem by requiring that the tuple be sent securely during enrollment.

Lemma 4.

A PPT adversary can impersonate a legitimate user to the server with only negligible probability.

Proof: As the final authentication step is to complete a zero-knowledge proof, an

attacker would have to be able to defeat a zero-knowledge proof, which happens only with a negligible probability if an attacker does not know the user's secret.

Lemma 5.

Given physical access to the device, an attacker could impersonate the legitimate user with only negligible probability.

Proof: If an attacker had access to the device, it would not be able to compute the proper hash value unless it supplied the correct PIN to the device. If the attacker attempted a brute force attack on the user's PIN, it would be trivial for a server to detect and disable the user's account temporarily. As long as the key space for the PIN is sufficient, this attack is not realistic.

Lemma 6.

A legitimate user can authenticate to a legitimate S, except with negligible probability.

Proof: As a legitimate user would have access to the user PIN and a valid tuple from the server, he would be able to successfully complete the zero-knowledge proof, thus authenticating.

Lemma 7.

An attacker cannot enroll using an existing or past user's credentials, except with negligible probability.

Proof: An attacker would be able to capture a user's tuple during authentication. It is plausible that he could attempt to enroll using this tuple. To prevent this, when the service provider issues the tuple initially, it also provides a nonce. During the enrollment protocol, the user submits the committed value, the tuple, and a hash of the tuple, nonce, and committed value. The service provider will verify that this tuple is valid. If the tuple is not valid or has already been enrolled, the service provider denies the enrollment request.

6.4.2 Man in the Middle

One of the considerations for system design is the presence of a man in the middle (MITM) attacker. If the MITM attacker is able to capture sufficient sensitive data, he could potentially compromise the security and integrity of the system.

Lemmas 1, 2, and 3 address the capabilities of an MITM attacker. Notably, since the attacker would only capture public information or interim steps of the zero knowledge proof, he is unable to recover anything of interest. However, an active MITM attacker would be able to drop all traffic between the various parties and commit a denial of service attack, though this case is not of great interest.

6.4.3 Replay Attacks

It is important to ensure that an attacker cannot record communication and then replay these communications later on.

Lemma 7 addresses this type of threat. Since the enrollment stage requires a nonce, it is not possible to record a user's tuple from the authentication step and use that to enroll. Lemma 1, 2, and 3 make mention of the fact that only interim steps of the zero knowledge proof would be recoverable, which are not usable in a replay attack.

6.4.4 Impersonation

Impersonation is another threat that must be considered; it is unacceptable if an adversary is able to pose as the service provider or as a user.

Lemmas 4 and 5 discuss the threat of impersonation. Simple brute force impersonation of a user to a service provider would require defeating a zero knowledge proof, which is computationally unfeasible. If the attacker was able to steal the physical device, he would be able to execute the zero knowledge proof, assuming he could guess the user's PIN. If a brute force attack was used, the service provider would

easily be able to detect it.

6.5 Implementation

From a high level view, Figure 6.5 describes the initial implementation of a PEAR enabled device. The Saxo-L board from KNJN.com [43] was used to do this work. The board contained both an FPGA and an ARM micro controller. The FPGA was used to implement the PUF device and supporting circuitry. The ARM micro controller was used to do the general purpose computation, including all the different cryptographic processing. The two were connected to each other with a Serial Peripheral Interface. A keypad for entering PINs was connected using a breadboard and directly wiring it to I/O pins of the ARM. The entire system was then connected to the PC over a USB connection.

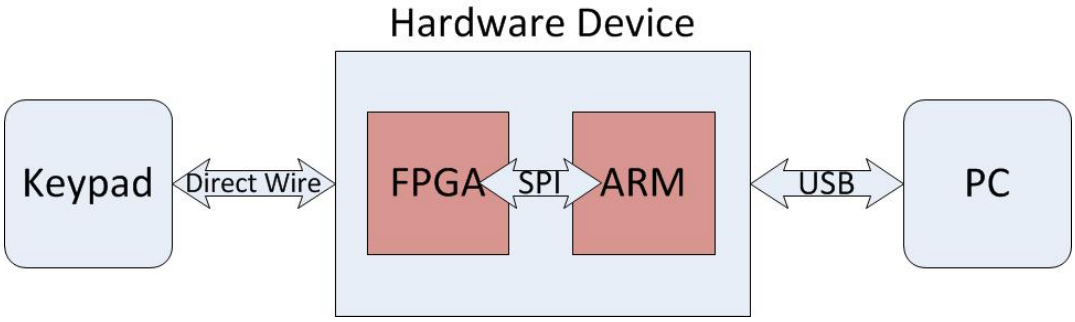

Figure 6.5. Implementation of a PEAR device

The entire Saxo-L board is 44mmx60mm, which is smaller than a typical credit. As such, a production level PEAR device could be implemented in a variety of user friendly sizes. A dongle or a smart card are two possible form factors that would be usable and convenient. The keypad for PINs could be overlayed as an array of touch-capacitive buttons on top of either form factor as well.

6.5.1 Limitations

Because the initial implementation was proof of concept, there are several ways it could be improved. Several vulnerabilities are also present that were discussed in Chapter 2.

One glaring shortcoming is the fact that the PUF and the ARM are connected over an SPI interface which is completely open on the Saxo-L board. As such, an adversary could easily put logic probes on the SPI bus and record the PUF challenge/response values as they were transmitted to the ARM, since they are sent unencrypted. There are a few ways to remedy this.

One solution is to put sorts of tamper proofing in place, such as potting, power filtering, among others. This would be effective, but would most likely be expensive and not completely effective. Another solution would be to encrypt the traffic being sent over the SPI bus. This would be fairly easy to do, but would impose some computational overhead on the entire device, which may or may not be acceptable.

The ideal solution to this problem is to incorporate the micro controller onto the same chip as the PUF itself. This is possible since there are "soft core" versions of ARM available which can be instantiated on the FPGA. Additionally, Altera (who makes the FPGA on the Saxo-L) provides a soft core micro controller for their FPGAs, as do many other FPGA vendors. This is probably the best solution since it eliminates the need for two discrete chips, since both portions can be put in one FPGA. It is still possible to use the same SPI bus concept, but the SPI bus will be internal to the FPGA, so it will not be possible to put logic probes on it. If an attempt is made to strip portions of the FPGA to insert probes, this will likely disturb the PUF and corrupt it, so any results will be unhelpful. Tampering with the FPGA and its relation to the PUF inside was discussed in Chapter 4 in more detail.

If the entire device was incorporated on to an FPGA, there would need to also be some sort of power filtering done. This would prevent differential power analysis attacks from being done on the cryptographic portions of the FPGA.

While it is necessary to prevent tampering to some degree, Lemma 5 means that a large amount of tamper proofing is not necessarily needed. The device is in its most vulnerable state after the PIN has been entered and the device is executing the zero knowledge proof. If an attacker has attached logic probes and other tools necessary to conduct many kinds of attacks, it would be very obvious to the legitimate users, who would then refuse to enter their PINs.

6.6 Future Work

The PEAR system as it exists is quite functional. However, some extensions could be done as future work to improve the system.

One such improvement is the concept of "transaction levels". Typically, service providers grant a user all services after he has logged in, regardless of the sensitivity of the requested action. For instance, a web application will require a user to log in before starting a new session, after which, all requests are granted. Transaction levels are the use of different credentials or the requiring of re-authentication upon requests for certain actions.

In a banking website example, one transaction level may allow a user to perform non-destructive operations, such as checking an account balance. However, for destructive operations, such as making an online bill payment, a different transaction level would be needed.

Transaction levels would be possible to implement so that a user only needed one physical PEAR device. This is desirable, because requiring a user to carry around multiple PEAR devices is not necessarily realistic.

Properly implementing transaction levels would require addressing a number of technical challenges. First, we would need an appropriate policy language to capture the behavior. Next, we would need to explore how to present the user with a usable interface that abstracts the behavior. Finally, we would need to consider the trade-offs in usability and security that would result from applying these approaches.

7. SMART GRID AND SMART METERS

7.1 Overview

There has been a move recently towards the design and implementation of what is called a "smart grid." A smart grid is an electrical power grid which can gather information about the meters, houses, and consumers of electricity that are attached to it.

By gathering information from a smart grid, a power company can more efficiently manage its production and delivery of electricity, thus reducing cost. Not only does the smart grid help the power company, but by providing analytics to its customers, customers can adjust their consumption habits to lower their bills as well. With the increasing cost of fossil fuels and the increasing energy consumption of today's consumer, reducing wasted electricity will be very useful.

Naturally, with this increase in functionality, there is an increase of risk as well. While the power company can use the smart grid to smartly redirect power to where it is needed, an attacker might try to direct power away from an area or direct so much power to an area that the system becomes overloaded and fails.

As a response to the increased dangers that a smart grid provides, we describe a system that is designed to allow for secure, yet convenient, communication and management between consumers and the utility company. Existing systems are leveraged, such as the ZigBee mesh network [46], which is further described below. One of the main things that is critical in this new system is that individual meters are able to be uniquely identified so billing can be done correctly, but also so no one can impersonate them. This is a perfect opportunity for PUF technology.

7.2 Actors

For the rest of the chapter, it will be useful to discuss the different players in the smart grid scenario.

7.2.1 Utility Company

One of the major players in the smart grid scenario is the utility company. The utility company is responsible for generation of the electricity using techniques such as coal power, hydroelectric, wind, among others. The utility company is also responsible for managing the dynamic delivery of power to areas that need it.

Finally, the utility company is also responsible for maintaining information on all of its customers and billing them appropriately.

7.2.2 Substations

Substations receive large amounts of high voltage power from the utility company. However, this power is not ready for end user consumption. The substation thus takes the power from and converts it to more reasonable levels that is suitable for being to delivered to end users.

The substation will communicate with the utility company to tell the utility how much load it is under. Depending on the load, it will then receive more or less power. It also acts as a sort of "collector" node for different smart meters. Smart meters will communicate with the substation which will then relay the data back to the utility company.

7.2.3 Smart Meters

A smart meter is much like a regular power meter but with some added features. Smart meters measure power as a normal meter, but they can typically be configured so that they can also "rewind" if a user is pumping power back into the grid. Addi-

tionally, smart meters have many different features which allow real time information and analytics about power consumption to be obtained.

One of the large benefits of smart meters is that the utility company can communicate and read them without necessarily sending a worker out to read the values. This is much faster and more cost effective than traditional power meters. This is accomplished using some sort of wireless interface. Typically, the ZigBee [46] [47] wireless protocol is used and is set up so that all meters form a mesh network with each other. This allows meters that are not within wireless range of a substation (or some other communication center) to still talk to the substation. This is accomplished since each smart meter not only transmits its own data, but it also acts as a relay for other nodes in the mesh network.

Note that smart meters have some amount of general purpose computation, but they are not very powerful. As such, it is necessary to optimize programs so that they can be run on the smart meters.

The smart meters used in this project are produced by Landis+Gyr. They consist of two components of interest. One board is called the "metering board" and is responsible for measuring and recording information about the actual power consumption of attached devices. The other board is called the "communication board". It is responsible for performing the different types of communication as necessary. The two are connected over an event-based interface, but this interface is not encrypted.

There are 5 security levels on the smart meter, each giving a different amount of privilege to different meter controls, with level 0 being read only access and level 5 giving control of everything. There is a table in the smart meter which stores each of the corresponding keys for the levels; These keys are referred to as access level keys. Communication between the utility and smart meter is encrypted using a random session key, which is derived from an encryption key.

7.3 Physical Systems

The actors above are clearly examples of physical systems. All three heavily interact with the environment around them to fulfill the power needs of consumers.

The utility company as a whole might not be considered a physical system, but parts of it will definitely be considered examples of deployed physical systems. The utility must manage the production of electricity by its generators as well as well as manage the communication of commands to the various smart meters, via the mesh network.

Substations could also be considered deployed physical systems, since they perform a large amount of interaction with the physical world to transform the power from the utility company into power that is usable by consumers. They additionally must take a large amount of input from both consumer smart meters and the utility to decide how to manage the power resources in a dynamic way.

The smart meters are deployed physical systems. A large part of the smart meters job is to record and analyze the power usage of the home or building they are attached to. This requires interaction with the physical world. However, smart meters are also clearly networked, since they can have so much interaction across them (in the concept of the mesh network) and the utility company.

7.4 Protocol

Our goal is to provide a protocol that allows for secure, authenticated communication between the utility company and smart meters, in the presence of and despite different types of adversaries. We leverage the use of a PUF device in conjunction with the smart meter to uniquely identify the smart meter to the utility company.

Before describing the protocol itself, it will be helpful to discuss the information each party is expected to maintain. The utility company will maintain a database containing two tables, one for zero knowledge commitments and the other for access level keys. The first table, referred to as the authentication table, will correlate a

specific meter with a zero knowledge proof. The second table, referred to as the access level key table, will correlate a smart meter with the five keys for the five different security levels of the meter. This table also stores the encryption key.

The smart meter needs to store the public key of the utility company and also maintain a "master key" that is generated internally using the PUF.

Both parties also need enough information about the mesh network to ensure that they can communicate with each other. Both parties also need to share the encryption key, which is used to derive temporary session keys.

The first step that must be done is that the utility company must record a zero-knowledge commitment from the smart meter for a specific challenge. This is illustrated in Figure 7.1. This step is required so that later in the protocol, the smart meter can execute a zero-knowledge proof to prove its identity. Note that this step must be done out-of-band from the mesh network, lest an adversary enter an invalid commitment. This could be done at manufacture time or at installation time by a technician, as long as it is executed through a secure channel.

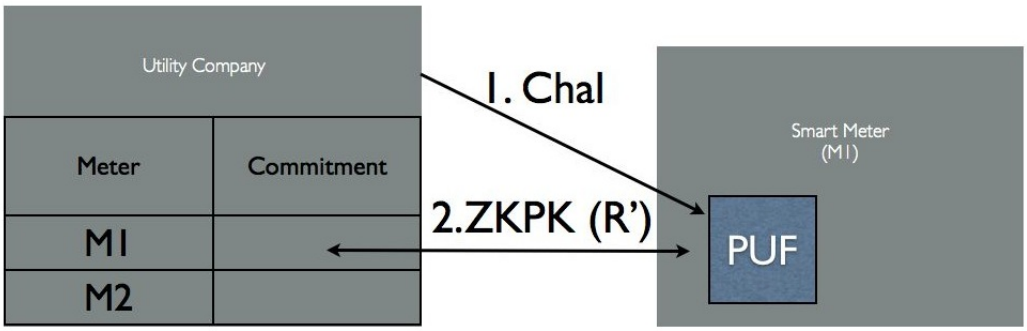

Figure 7.1. Enrollment of the commitment

After the commitment phase has been completed and the meter has been deployed, the *keying* operation can be performed. This operation starts by executing a key derivation protocol. Recall that the smart meter maintains a "master key" which is generated by the PUF. This master key is passed through some sort of one-way function, such as a hash in the style of $K_1 = H(MK|1), K_2 = H(MK|2)...$ so that six

new keys are derived, but the original master key is never revealed. These six keys are then encrypted under the public key of the utility company. All the encrypted keys as well as a zero knowledge proof are then sent to the utility company. The ZKPK allows the utility company to authenticate the encrypted keys before updating its database with the six new entries. This transmission step is detailed graphically in 7.2. Figure 7.3 graphically describes the key derivation process.

Note that the PUF uses an internal feedback loop, so that every invocation of the keying procedure will generate a different PUF response, which will in turn generate different derived keys.

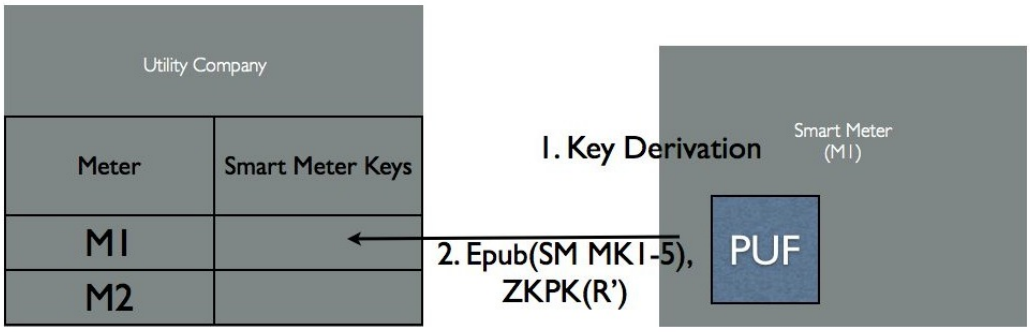

Figure 7.2. Storage of the derived keys

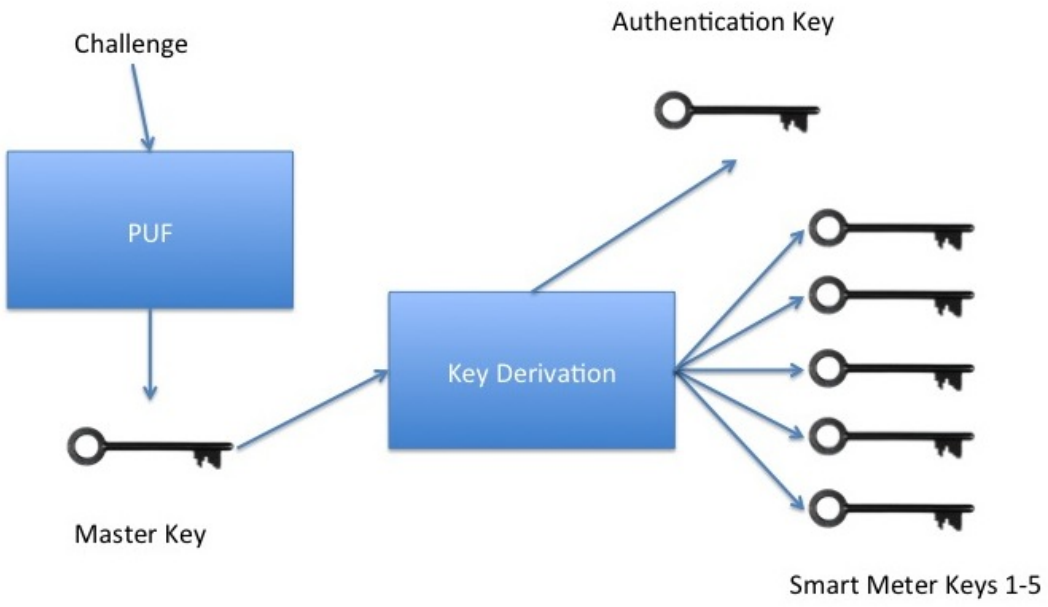

Figure 7.3. The key derivation process

After the utility company has received these five access level keys and the encryption key, interactions between the smart meter and the utility company will then proceed as it currently does. That is, the smart meter and utility company will use the shared encryption key to compute a temporary, symmetric session key which will be used to encrypt data that is sent between the two.

Note that since the smart meter has limited computation powers, the operations above are fairly expensive. This is acceptable though, since the enrollment and keying operations will be done very infrequently. If they need to be re-run, this can be planned ahead for off peak times, such as in early morning and consumers can be notified of this. The symmetric session key is frequently used, but symmetric encryption is much less computationally expensive than asymmetric encryption.

Additionally, note that the utility company signs any correspondence during the enrollment or keying stages. This allows the smart meter to validate the signature and thus ensure that commands are actually coming from the utility company.

7.5 Security Considerations

Because there are several different ways in which the system could be compromised and the gravity that such a compromise could have, it is necessary to consider and discuss the different security issues.

Note that a distinction is made between the set of initial enrollment and keying operations that we defined versus the existing system of encryption using session keys.

7.5.1 Meshnet Transmission Security

One potential problem is that when data is being transmitted across the meshnet, rather than relaying, a node may attempt to read the data in transit. This would be considered a man in the middle attack. There are encryption layers imposed by the ZigBee protocol and other transmission protocols, but it stands to reason that if a node were malicious, it might be able to crack these layers.

As such, we add another layer of security at the application layer. Data is encrypted under the public key of the utility company during the keying operation. As such, an adversary would be required to compromise the public key algorithm, which is considered computationally unfeasible. A more likely attack would be to compromise the private key of the utility company. This possibility is considered separately below.

Data being transmitted using the session key between the utility company and smart meter is encrypted under a symmetric key algorithm, EAX', which is a variant of AES that is defined under ANSI standard C12.22 [48] [49]. As such, an adversary would have to be able to defeat this algorithm to make any progress. An attack on EAX' has been reported [50], but this attack is new and may not be effective enough to compromise the total integrity of the communication.

The system is also resilient against replay attacks in this situation, since both time stamps and nonces are incorporated. As such, any replay will have invalid time stamps and nonces, so will not be valid.

There is a risk of a denial of service if an active adversary simply drops all traffic, but this is considered an acceptable risk.

7.5.2 Compromised Smart Meter Key or Encryption Key Compromise

It is possible that somehow, one of the five smart meter keys or the encryption key used to derive session keys could be compromised. This might happen if an employee copies a key from the database or for any other number of reasons. Even if this happens, security is only temporarily effected.

Recall that the master key generated from the PUF is passed through a one way, key derivation process. As such, even if one of the derived keys (the smart meter keys or the authentication key) is compromised, the master key is still secure. To re-secure the system, the keying procedure will then be re-run, which will provide a brand new set of smart meter keys and authentication key.

This procedure is essentially analogous to a re-key request, even if there is no compromise.

Note that this case is separate from the databases containing this information, which is considered below.

7.5.3 Utility Database Compromise - Commitment Table

If the commitment table in the utility database is compromised, the results will not be catastrophic. Since the committed values for the ZKPK do not leak any information, the adversary will not be able to impersonate any users or garner any new information about the underlying authentication key.

The worst case is when the adversary changes the committed values in the database. This would essentially create a denial of service attack. However, he could also update the database with his own commitment value and place himself between the smart meter and the utility company as an active MITM and then impersonate the smart meter.

Since he would be able to authenticate to the utility server, he could thus send false data about consumption or whatever he wanted. This threat could be dealt with by making sure that the database is very secure and that the entries are carefully monitored and updates are only allowed during the keying operation.

Note that in this case, the adversary would not be able to impersonate the utility company to the smart meter, since he would not know the private key of the utility, which is necessary to sign the various messages.

This compromise would have no effect on the messages being sent under the temporary session key, since there is no need for the authentication key during that stage.

7.5.4 Utility Database Compromise - Key Tables

It is possible that the database storing the smart meter access level keys or encryption key could be compromised. If these keys were compromised, the adversary could then access the sensitive commands of the smart meter or read the interim traffic between the two parties.

However, this is not necessarily a real danger, since the keys are encrypted under the utility's private key before being stored into the database. As such, the adversary would only recover ciphertext. Assuming he cannot break the public key algorithm, the data will remain secure.

There is a possibility that if the database was compromised, the adversary might have been able to recover a key as it is being decrypted in memory and record that. As such, it is wise to execute a re-keying operation for any meters whose information was stored in that database.

7.5.5 Utility Company Private Key Compromise

One of the worst case scenarios is that the private key of the utility would become compromised. As previously described, the smart meter's PUF-derived master key would remain secure, since it is never transmitted.

However, all smart meter access level keys and encryption keys could potentially be compromised, since the adversary would record all traffic during the keying operation and use the private key to recover the keys.

Once those keys were recovered, it would be trivial for the adversary to communicate with the smart meter to execute sensitive commands, request re-key operations, or whatever the adversary desired.

To recover from this attack, the old private key would first need to be discarded. Following that, the new public key would need to be installed in every single meter in the grid. This would have to be done by hand, since any network transmissions could not be trusted. Additionally, every smart meter would then need to execute the keying operation so that the utility company could then update its database with the new values.

These recovery procedures would be very expensive and time consuming. As such, it is critical that the private key never becomes compromised. Key management is a difficult, open problem that needs to be solved.

7.5.6 Smart Meter Physical Security

Since smart meters are physical systems, it is necessary to discuss and consider some of the physical threats they face.

The smart meters are actually designed to be fairly tamper resistant. If the external glass shell is removed without the use of a technician's key, a signal is sent over the network to alert the utility company. After that happens, the utility company can then flag any future traffic as suspicious and send a technician out to investigate and fix the meter. Regardless, improving the tamper resistance of smart meters would be a good step to take.

Because the smart meter is doing cryptographic operations, it could potentially be vulnerable to power analysis attacks as well as monitoring of emitted signals. To resolve this, some sort of potting mechanism, power filtering, and shielding should be

implemented.

As previously described, the PUF is extremely resilient against tampering, so it will likely not be a problem. Incorporating all the different components on a system on chip will be a good step towards security, since this will hide all the internal buses as well as protect the areas with sensitive information stored on them.

7.6 Implementation

For the proposed scenario and protocol, we developed a proof of concept implementation. In the interest of time and difficulty, rather than creating an entire smart meter from scratch, we opted to implement the PUF inside an FPGA and perform some of the network communication and computations on a PC, the smart meter PC, which was connected to another PC, which is modeling the utility company. The smart meter PC is connected to an actual smart meter via an optical port. The smart meter is also connected to the utility PC via a ZigBee interface, as would be used in the actual field. Figure 7.4 represents the prototype architecture.

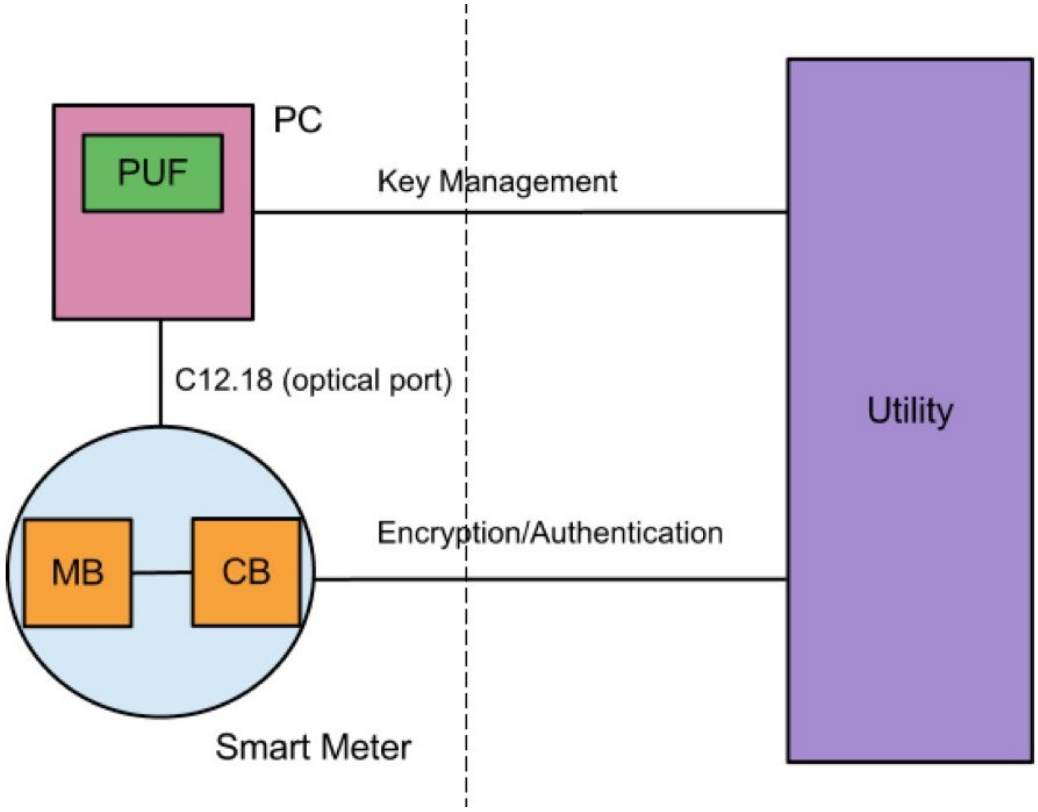

Figure 7.4. Prototype implementation of smart grid project

We used the Xilinx Spartan-6 Embedded Development board for our FPGA platform. This board was useful since we were able to incorporate an existing PUF design on it. Additionally, we were able to use the Microblaze soft core microprocessor which Xilinx provides. As mentioned in Chapter 6 and Chapter 5, it is a good idea to include the microprocessor on the same chip as the PUF device. This allows buses sending sensitive information, such as PUF responses, to be better protected against adversaries. Again, inserting logic probes into the inner parts of the chip would most likely disrupt the PUF, so any results obtained about the PUF would thus be useless.

For the smart meter PC, we created an application in C++ which uses standard networking libraries to communicate with the utility PC. The smart meter PC interacts with the PUF board via a serial port connection. The PC also contains drivers and code to communicate with the actual smart meter over an optical port interface.

7.6.1 Limitations

Our approach has many different components that are not necessarily going to be present in an actual, deployed system. For instance, the fact that the PUF board and the smart meter PC are separate from the smart meter is a major issue that would not work in the field.

In the future, it would be better to integrate all the control circuitry, PUF device, and metering circuitry into a single chip, such as a large FPGA or even an ASIC design. In this way, only one chip would need to be installed into the smart meter, rather than the large amount of devices currently needed.

8. SUMMARY

Systems that interact with the physical world are not only becoming more pervasive, but also more powerful from a computing perspective. It is helpful to classify these systems into groups of standalone, deployed, and peripheral physical systems. The distinction lies in how they communicate with other systems, such as the ones connected over the internet.

There are many different security issues facing physical systems. Not only must they compensate for threats similar to those faced in software, such as impersonations, man-in-the-middles, and replay attacks, they must also contend with threats specific to a system that exists in the physical world. These include power analysis attacks, which can glean information from power consumption, signal injection attacks, which can alter system behavior by bombarding the system with wireless signals, and simple tampering, such as breaking components or attaching logic probes.

The technology of Physically Unclonable Function (PUF) was presented, which allows strong guarantees of device authenticity to be made by leveraging the challenge-response properties of PUF. These devices are able to offer these guarantees since they cannot be duplicated using a manufacturing process, so the responses they give to given challenges are always distinct from other PUFs. PUFs require a certain amount of support circuitry to deal with bit errors that occasionally occur, but this is acceptable and solvable using existing error correction techniques.

It is possible to utilize PUF devices in conjunction with certain cryptographic protocols, such as zero knowledge proof of knowledge (ZKPK) proofs, to implement interesting applications. Several different applications were presented, each of which demonstrated the use of PUF in a different context.

The PUF ROK application leveraged a PUF device to create keys that, once used, are unrecoverable. This was done by giving the PUF an initial "seed" value and then

creating a feedback loop with the PUF. When the PUF generated a response, it would overwrite the previously stored value. Since PUFs are one-way functions, there is no way to go backwards and recover the value that was previously used. These ROKs could then be used to create "self-destructing" documents or for an authority to give a delegate a limited number of a higher privilege level.

The PEAR application used PUFs in a way to uniquely and securely identify devices, despite insecure communication channels. In addition to a PUF, a ZKPK protocol was used to provide this benefit. The PUF as well as an external keypad make up a PEAR device so that data can be entered securely. The initial goal was to allow users to be able to log in to websites securely, even if a hardware key logger was attached to the keyboard. This is possible, but PEAR also allows this capability in the presence of software threats on the PC, since all traffic is encrypted.

Finally, the smart grid project utilized PUFs to provide strong guarantees of smart meters' identity. This is critical because if the utility company was not sure of meters it was communicating with, catastrophic attacks would be possible, such as overloading of power handling circuits. The PUF was again used in conjunction with ZKPK proofs to protect information in transit between the two parties. Additionally, since the utility company is required to maintain a large database, storing ZKPK commitments, rather than secrets directly, is a much more secure approach. Another interesting point of this application was that the PUF generated and maintained a master key internally, but derived keys were used in all steps of the protocol. This protects the long term security of the device, even in the event of certain security compromises.

These different applications show the versatility of PUF technology in helping to secure physical systems. Depending on which cryptographic tools and protocols they are used in conjunction with, PUFs can be used in a variety of different ways, as was demonstrated.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Mainichi Daily News. From broken temp sensors to leaky pipes, Fukushima nuke plant plagued with problems. `http://mdn.mainichi.jp/mdnnews/news/20120303p2a00m0na008000c.html`, March 2012.

[2] P. Wright. *Spycatcher: The candid autobiography of a senior intelligence officer*. Viking, 1987.

[3] J. Volpe. Vulnerability assessment of the transportation infrastructure relying on the global positioning system. Technical report, National Transportation Systems Center, 2001.

[4] W. Van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? *Computers & Security*, 4, 1985.

[5] Markus G. Kuhn. Electromagnetic eavesdropping risks of flat-panel displays. In *Proceedings of the 4th International Conference on Privacy Enhancing Technologies*, PET'04, pages 88—107, Berlin, Heidelberg, 2005. Springer-Verlag.

[6] H. Rosenblum. Tempest fundamentals. Letter of promulgation, National Security Agency, 1982.

[7] Joan Daemen and Vincent Rijmen. *The design of Rijndael*. Springer-Verlag, New York, Inc., Secaucus, NJ, USA, 2002.

[8] United States. National Bureau of Standards. Data encryption standard. FIPS Pub 46-3, National Bureau of Standards, Washington, DC, USA, 1977.

[9] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (Blowfish). In *Fast Software Encryption, Cambridge Security Workshop*, pages 191—204, London, UK, 1994. Springer-Verlag.

[10] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644—654, 1976.

[11] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 26:96—99, January 1983.

[12] Wikipedia Acdx. Diagram illustrating how a simple digital signature is applied and verified. `http://en.wikipedia.org/wiki/File:Digital_Signature_diagram.svg`, 2012. [Online; accessed 23-February-2012].

[13] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 210—217, New York, NY, USA, 1987. ACM.

[14] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 129—140, London, UK, 1992. Springer-Verlag.

[15] G. Suh, C. O'Donnell, and S. Devadas. Aegis: A single-chip secure processor. *IEEE Design and Test of Computers*, pages 570—580, 2007.

[16] J. Merchan, S. Kumar, P. Tuyls, and G. Schrijen. Identification of devices using physically unclonable functions. US Patent 7062320, September 2011. `http://www.patentlens.net/patentlens/patent/US_7062320/`.

[17] P. Ravikanth. *Physical one-way functions*. PhD thesis, Massachusetts Institute of Technology, 2001. `http://alumni.media.mit.edu/~pappu/pdfs/Pappu-PhD-POWF-2001.pdf`.

[18] B. Skoric, T. Maubach, S.and Kevenaarl, and P. Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied Physics*, 2006.

[19] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 148—160, New York, NY, USA, 2002. ACM.

[20] A. Maiti, L. McDougall, and P. Schaumont. The impact of aging on an FPGA-based physical unclonable function. In *2011 International Conference on Field Programmable Logic and Applications (FPL)*, pages 151—156, September 2011.

[21] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. A large scale characterization of RO-PUF. In *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 94—99, June 2010.

[22] Sam Kerr, Michael S. Kirkpatrick, and Elisa Bertino. PEAR: A hardware based protocol authentication system. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, SPRINGL '10, pages 18—25, New York, NY, USA, 2010. ACM.

[23] Michael S. Kirkpatrick, Sam Kerr, and Elisa Bertino. PUF ROKs: A hardware approach to read-once keys. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 155—164, 2011.

[24] Michael S. Kirkpatrick and Sam Kerr. Enforcing physically restricted access control for remote data. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, pages 203—212, New York, NY, USA, 2011. ACM.

[25] Ross Anderson. *Security Engineering: A guide to building dependable distributed systems*. Wiley, 2001.

[26] TPM main specification. Technical report, Trusted Computing Group, 2011. `http://www.trustedcomputinggroup.org/resources/tpm\_main\_specification`.

[27] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO 2008: Proceedings of the 28th Annual Conference on Cryptology*, pages 39—56, Berlin, Heidelberg, 2008. Springer-Verlag.

[28] Ironkey military strength flash drives. `http://www.ironkey.com`, 2010.

[29] Roxana Geambasu, Tadayoshi Kohno, Amit Levy, and Henry M. Levy. Vanish: Increasing data privacy with self-destructing data. In *Proceedings of the 18th USENIX Security Symposium*, 2009.

[30] Srivatsan Narayanan, Ananth Raghunathan, and Ramarathnam Venkatesan. Obfuscating straight line arithmetic programs. In *DRM '09: Proceedings of the Nineth ACM Workshop on Digital Rights Management*, pages 47—58, New York, NY, USA, 2009. ACM.

[31] Luis F. G. Sarmenta, Marten van Dijk, Charles W. O'Donnell, Jonathan Rhodes, and Srinivas Devadas. Virtual monotonic counters and count-limited objects using a TPM without a trusted OS. In *STC '06: Proceedings of the First ACM Workshop on Scalable Trusted Computing*, pages 27—42, New York, NY, USA, 2006. ACM.

[32] Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In *Proceedings of the 7th International Conference on Theory of Cryptography*, TCC'10, pages 327—342, Berlin, Heidelberg, 2010. Springer-Verlag.

[33] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *Theory of Cryptography*, volume 5978 of *Lecture Notes in Computer Science*, pages 308—326. Springer Berlin / Heidelberg, 2010.

[34] Eric Brier, Benoît Chevallier-Mames, Mathieu Ciet, Christophe Clavier, and École Normale Supérieure. Why one should also secure RSA public key elements. In *Cryptographic Hardware and Embedded Systems (CHES 2006)*, volume 4249 of *Lecture Notes in Computer Science*, pages 324–338. Springer-Verlag, 2006.

[35] Alexandre Berzati, Cécile Canovas, and Louis Goubin. Perturbating RSA public keys: An improved attack. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems (CHES 2008)*, volume 5154 of *Lecture Notes in Computer Science*, pages 380—395. Springer Berlin / Heidelberg, 2008.

[36] Alexandre Berzati, Cécile Canovas, and Louis Goubin. In(security) against fault injection attacks for CRT-RSA implementations. *Workshop on Fault Diagnosis and Tolerance in Cryptography*, 0:101—107, 2008.

[37] Alexandre Berzati, Cécile Canovas, Jean-Guillaume Dumas, and Louis Goubin. Fault attacks on RSA public keys: Left-to-right implementations are also vulnerable. In *CT-RSA '09: Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, pages 414—428, Berlin, Heidelberg, 2009. Springer-Verlag.

[38] Andrea Pellegrini, Valeria Bertacco, and Todd Austin. Fault-based attack of RSA authentication. In *Design Automation and Test in Europe (DATE)*, March 2010.

[39] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC '09: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, pages 474—495, Berlin, Heidelberg, 2009. Springer-Verlag.

[40] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Lecture Notes in Computer Science*, pages 1—18. Springer-Verlag, 2001.

[41] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Introduction to differential power analysis and related attacks. Technical report, Cryptography Research, 1998.

[42] V. Sundaresan, S. Rammohan, and R. Vemuri. Defense against side-channel power analysis attacks on microelectronic systems. In *Aerospace and Electronics Conference, 2008. NAECON 2008. IEEE National*, pages 144—150, July 2008.

[43] KNJN. Fpga-fx2 boards. `http://www.knjn.com/FPGA-FX2.html`.

[44] Offspark. Light-weight, modular cryptographic and SSL/TLS library in C - PolarSSL. `http://www.polarssl.org`.

[45] Encryption for ARM MCUs. `http://ics.nxp.com/literature/presentations/microcontrollers/pdf/nxp.security.innovation.encryption.pdf`, 2010.

[46] Yichi Zhang, Weiqing Sun, Lingfeng Wang, Hong Wang, R.C. Green, and M. Alam. A multi-level communication architecture of smart grid based on congestion aware wireless mesh network. In *North American Power Symposium (NAPS), 2011*, pages 1—6, August 2011.

[47] C. Bennett and D. Highfill. Networking AMI smart meters. In *Energy 2030 Conference, 2008. ENERGY 2008. IEEE*, pages 1—8, November 2008.

[48] ANSI. American national standard protocol specification for interfacing to data communication networks. Technical report, American National Standards Institute, 2008.

[49] M. Bellare, P. Rogaway, and D. Wagner. EAX: A conventional authenticated-encryption mode. Cryptology ePrint Archive, Report 2003/069, 2003.

[50] Kazuhiko Minematsu, Stefan Lucks, Hiraku Morita, and Tetsu Iwata. Cryptanalysis of EAXprime. Cryptology ePrint Archive, Report 2012/018, 2012.