SECURITY ANALYSIS OF STRONG PHYSICAL UNCLONABLE FUNCTIONS

TUDOR ALEXIS ANDREI SOROCEANU

On Arbiter PUFs and Bent Functions

Institut für Informatik
Fachbereich Mathematik und Informatik
Freie Universität Berlin
11th May 2017

Tudor Alexis Andrei Soroceanu: *Security Analysis of Strong Physical Unclonable Functions,* On Arbiter PUFs and Bent Functions, © 11th May 2017

SUPERVISORS:
Prof. Dr. Marian Margraf

Nils Wisiol

LOCATION:
Berlin

TIME FRAME:
11th May 2017

Für A. und A.
In tiefer Dankbarkeit und Liebe.

ABSTRACT

Modern Cryptography is heavily based on the ability to securely store secret information. In the last decade *Physical Unclonable Functions* (PUFs) emerged as a possible alternative to non-volatile memory. PUFs promise a lightweight and lower-priced option compared to non-volatile memory, which has to be additionally secured and is known to be prone to reverse-engineering attacks [Qua+16]. PUFs are traditionally divided into *Weak PUFs* and *Strong PUFs*, depending on the number of possible challenges. One of the more popular Strong PUFs on silicon integrated circuits is that of the *Arbiter PUF,* in which two signals run through n stages, influenced by a challenge, and an arbiter that decides the output of the PUF depending on where a signal arrives first. As one single Arbiter PUF is easily to model and learn (cf. [Gas+04; Rüh+10; Rüh+13]), Suh and Devadas [SD07] proposed to combine the output of multiple Arbiter PUFs with an XOR, however this construction also turned out to be learnable [Rüh+10; Rüh+13; Bec15]. In this thesis we will investigate the use of different combining functions for Arbiter PUFs. As combined Arbiter PUFs show structural similarity to *linear feedback shift registers* (LFSR) and *nonlinear combination generators* (the parallel use and combination of multiple LFSRs), we will carry out known attacks targeting the combining function on Arbiter PUFs. We will show that in order to prevent these attacks more sophisticated combining functions than XOR are needed. We propose a new class of Strong PUFs called *Bent Arbiter PUFs*, using Boolean bent functions as combiner. It turns out that Bent Arbiter PUFs are resistant against such kind of attacks. Future work must contain the analysis of the feasibility of Bent Arbiter PUFs against machine learning attacks.

ZUSAMMENFASSUNG

Das Gebiet der modernen Kryptographie basiert auf der Möglichkeit, geheime Informationen sicher zu speichern. In den letzten Jahren haben sich *Physical Unclonable Functions* (PUFs) als Alternative zu nicht-flüchtigem Speicher hervorgetan. PUFs versprechen eine einfache und kostengünstigere Alternative zu nicht-flüchtigem Speicher, der oft zusätzlich abgesichert werden muss und anfällig gegenüber Reverse-Engineering-Angriffen ist [Qua+16]. PUFs werden üblicherweise in Abhängigkeit von der Anzahl an möglichen Anfragen in zwei Klassen eingeteilt*: Weak PUFs* und *Strong PUFs*. Eine der am meisten verbreiteten Art von Strong PUFs ist die *Arbiter PUF*, in der zwei Signale durch einen Schaltkreis mit n Stufen laufen. Die Pfade

der Signale werden durch die Anfrage beeinflusst, bis am Ende des Schaltkreises ein Schiedsrichter die Ausgabe der PUF festlegt. Dies geschieht in Abhängigkeit davon, an welcher Stelle ein Signal zuerst ankommt. Eine Arbiter PUF ist leicht zu modellieren und zu lernen (vgl. [Gas+04; Rüh+10; Rüh+13]). Aus diesem Grund haben Suh and Devadas [SD07] vorgeschlagen, die Ausgaben mehrerer Arbiter PUFs mit einem XOR zu verknüpfen. Allerdings hat sich diese Konstruktion auch als unsicher herausgestellt [Rüh+10; Rüh+13; Bec15]. In der vorliegenden Arbeit haben wir nach einer geeigneteren Möglichkeit gesucht, mehrere Arbiter PUFs zu kombinieren. Kombinierte Arbiter PUFs zeigen strukturelle Ähnlichkeiten zu linear rückgekoppelten Schieberegistern (LFSR) auf, die durch eine Combinerfunktion verknüpft werden. Wir werden Angriffe auf die Combinerfunktion ausführen und zeigen, dass komplexere Funkionen als XOR notwendig sind, um diese Angriffe zu verhindern. Wir stellen mit *Bent Arbiter PUFs* eine neue Klasse von Strong PUFs vor, die Bentfunktionen als Combinerfunktion nutzen. Wir können zeigen, dass Bent Arbiter PUFs gegenüber Angriffen auf die Combinerfunktion sicher sind. Zukünftig muss die Resistenz von Bent Arbiter PUFs gegenüber Algorithmen, die maschinelles Lernen benutzen, untersucht werden.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

Teil I

INTRODUCTION AND PRELIMINARIES

1

INTRODUCTION

Cryptography is the science of developing and analyzing schemes that handle the en- and decryption of messages. The desire of human kind to hide communication and make it only available to certain groups has led to a race between designers and breakers of cryptographic systems. As throughout the history the breaking of one system lead to the necessity of a new, safe, and more complex system, cryptographic algorithms became more and more sophisticated throughout the centuries. With the emergence of computers, modern cryptography does not anymore operate on letters but on bitstrings of zeros and ones. To a certain degree, this enables the mathematical modeling, computation, and verification of cryptographic systems. Mathematically speaking, a cryptographic system is a fivetupel $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ based on an alphabet $\mathcal{A}$. In the case of computers the alphabet corresponds to $\mathcal{A} = \{0, 1\}$. The finite set $\mathcal{P} \subseteq \mathcal{A}$ is the set of all possible messages, called the *plaintexts*. The finite set $\mathcal{C} \subseteq \mathcal{A}$ is the set of all possible encryptions of plaintexts, called the *ciphertexts*. The finite set $\mathcal{K} \subseteq \mathcal{A}$ is the set of all possible keys used to en- and decrypt plaintexts to ciphertexts, called the *keyspace*. For every key $k \in \mathcal{K}$ there is a *encryption function* $e_k : \mathcal{P} \to \mathcal{C} \in \mathcal{E}$ and a corresponding *decryption function* $d_k : \mathcal{C} \to \mathcal{P} \in \mathcal{D}$ such that

$$d_k\left(e_k(m)\right) = m$$

for every plaintext $m \in \mathcal{P}$.

Security properties

The task of a cryptographic system is to enable secure, accurate, and authenticated communication between two or more parties over insecure communication channels. To achieve this, there are three key concepts defined in cryptography and information security:

1. *Confidentiality*: The goal of confidentiality is guaranteed by a cryptographic system, if no unauthorized individuals or entities can retrieve information from the system or encrypted messages.

2. *Integrity*: A cryptographic system guarantees the integrity of given information, if no one can manipulate or change this information.

3. *Authenticity*: The authenticity of the attribute of a piece of data is given, if its truth can be unambiguously be reviewed and checked.

The three key concepts are mostly accomplished by *cryptographic primitives*. Examples of primitives include encryption ciphers, hash functions, and digital signature algorithms. A cryptographic system can consist of one or more primitives. *Encryption ciphers* are mainly used to ensure the confidentiality and authenticity. *Hash functions* are used to check the integrity of transmitted data. *Digital signature algorithms* are used to assure the authenticity.

The security of cryptographic systems and primitives is measured with the help of a security level, that is usually expressed in bits. If a system has a security level of n bits, it means that an attacker would need approximately 2^n operations to break it. We will state the number of operations in the Landau notation to have the option to compare runtimes with other computational algorithms.

Attack models

To model and analyze attacks on cryptographic primitives we use an oracle access model. An attacker interacts with an oracle which is in possession of the attacked entity. To do so, the attacker sends a request (*challenge*) to the oracle, the oracle processes the request and generates an answer (*response*) which is then send back to the attacker. The request can for example be a plaintext which the attacker ought to be encrypted, or a ciphertext which shall be decrypted. We divide the knowledge of the attacker and the authorized operations of the oracle into different attack classes:

- *Ciphertext-only attack*: The attacker has knowledge only of one or more ciphertexts. There is no communication with the oracle allowed.
- *Known-plaintext attack*: The attacker knows one or more plaintexts and their corresponding ciphertexts. There is no communication with the oracle allowed.
- *Chosen-plaintext attack*: The attacker has the possibility to choose one or more plaintexts and send them to the oracle. The oracle encrypts the plaintexts to their corresponding ciphertexts and forwards these to the attacker. A subcategory of this attack is the *adaptive chosen-plaintext attack*. Here, the attacker has the opportunity to send the plaintexts one by one to the oracle and to analyze the received ciphertexts between each step. If desired, the attacker can adapt future plaintexts according to the analysis of the previous ciphertexts.
- *Chosen-ciphertext attack*: The attacker can choose arbitrary ciphertexts and send them to the oracle. The oracle decrypts the ciphertext to their corresponding plaintexts and sends them back to the attacker.

Kerckhoff's principle

Depending on the attack model the attacker can have different goals that she wants to achieve with the attack. But the structure of the cipher should never be kept a secret. In 1883 Auguste Kerckhoffs

drafted six principles for military ciphers [Ker83]. The most important and notable principle is that the security of a cipher should not depend on the secrecy of the cipher itself, but only on the used secret key. So that the cipher "can conveniently fall into the hands of the enemy".

Attack goals

Possible goals of attacks on cryptographic primitives are:

- *Secret key*: The goal is to figure out the deployed secret key for one or more ciphertexts.
- *Plaintext*: The goal is to figure out the plaintext to one or more ciphertexts. Here, the secret key is not the primary target, but an attack on the secret key can be used to decrypt the ciphertexts.
- *Complete breaking of the cipher*: The goal is to find a structural deficit in the used cipher to allow the en- and decryption without the knowledge of the used keys. This can be an attack on the theoretical foundation of the cipher or on one specific implementation (on hard- or software) of the cipher.

Brute force attack

The simplest attack on the secret key of a cryptographic system is the *brute force attack*. Here, the attacker iterates over all keys in the keyspace $\mathcal{K}$ and tries for each key if it decrypts the given ciphertexts. For ciphers that operate on the alphabet $\{0,1\}$ and a key length of $n \in \mathbb{N}$, a brute force attack would require $\mathcal{O}(2^n)$ steps. Hence, the above definition of the security level of n bits.

Asymmetric encryption

Encryption ciphers can be divided into two main groups. The first group are the *asymmetric* or *public key encryption ciphers*. Here, keys arise only in pairs and are assigned to a specific owner. The *public key* of the pair can be disseminated wildly, whereas the *private key* must only be known to the owner of the key pair. Each receiver of information needs a separate key pair. Asymmetric cryptography can be used to achieve the confidentiality of information, where the data is encrypted by the sender with the public key and decrypted by the receiver with his private key. But asymmetric cryptography can also be used to authenticate the sender of information. Hereby, the public key of the sender is used to verify her identity. Public key cyptosystems were first proposed by Diffie and Hellman [DH76] in 1976. The first practical and to date most used public key cryptosystem is the *RSA* system proposed by Rivest, Shamir and Adleman [RSA78] in 1978.

Symmetric encryption

The second group of ciphers is that of *symmetric encryption ciphers*. Here, the same cryptographic key is used to encrypt the plaintext and decrypt the ciphertext. The key represents a shared secret between the sender and receiver of the messages. A challenge arises in the distribution of the secret keys. To distribute keys over insecure information channels additional protocols are needed, like the *Diffie–Hellman*

key exchange [DH76]. Another possibility is to share the keys at an initialization phase; for example the initialization phase of smart cards from the distributor.

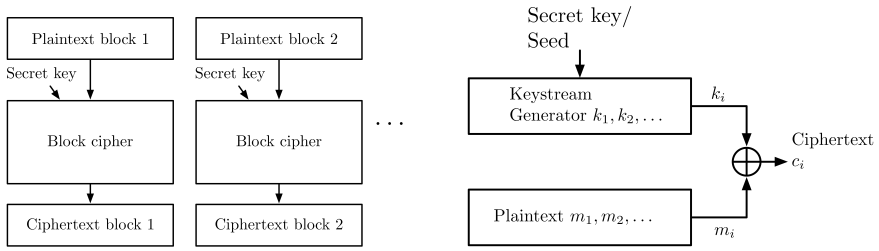
Block ciphers

Symmetric ciphers can be further divided into two subclasses, *block ciphers* and *stream ciphers*. As the name suggests, block ciphers operate on blocks of a fixed length (see Figure 1.0.1a). For this, the plaintext is divided into given blocks and each block is encrypted with the same secret key to its corresponding ciphertext block. The concatenation of the ciphertext blocks makes up the ciphertext. To decrypt a ciphertext the reverse operation is applied. The first and most significant modern block cipher is the *Data Encryption Standard* (DES) from 1977 (see *Federal Information Processing Standard* 46). The DES was published by the United States' National Bureau of Standards and was mainly developed at IBM with consultation from the Nation Security Agency (this led amongst other things to a decrease of the key size). As the key length of 56 bits became unsafe with time, the United States' National Institute of Standards (NIST) announced in 1997 an open call for a new symmetric block cipher to replace the DES as the new standard. The winner of the contest was the *Rijndael block cipher* proposed by Daemen and Rijmen [DR99], which became later in 2001 the *Advanced Encryption Standard* (AES). The AES can operate on key sizes of 128, 192, and 256 bits.

Stream ciphers

The other subclass of symmetric ciphers consists of stream ciphers (see Figure 1.0.1b). A stream cipher is an algorithm with internal memory that receives bits of a plaintext and combines them one-by-one with bits of a keystream to output the ciphertext. Over the alphabet of $\{0,1\}$ the combination consists of the bitwise addition modulo 2, also known as XOR. This operation is denoted as $\oplus$. The most important part of the stream cipher is the keystream generator that produces the individual key bits. To an attacker, the keystream should look like a complete random sequence. However, a truly random sequence would cause a lot of trouble, since the receiver of a ciphertext would not be able to reproduce the needed keystream to decrypt the ciphertext. Therefore, a *pseudo-random number generator* (PRNG) is used as keystream generator. A PRNG is a deterministic algorithm that produces an output sequence that looks statistically independent and identically distributed. Using a *seed* as initialization parameter (sometimes also referred to as the secret key of a stream cipher) a PRNG can reproduce the same output sequence. One of the most used stream cipher construction is the *linear feedback shift register* (LFSR) which uses a shift register as PRNG. As a single LFSR is linear an thus unsafe, multiple LFSRs are combined to a *nonlinear combination generator* using a combining function. LFSRs, nonlinear combination generators, and stream ciphers are well studied and used, especially in the field of information theory.

Boolean functions



(a) Scheme of a block cipher (b) Scheme of a stream cipher

Figure 1.0.1: The two main types of ciphers

As most ciphers operate on the alphabet $\mathcal{A} = \{0, 1\}$, or mathematically speaking the finite field $\mathbb{F}_2$ of two elements, encryption ciphers can be modeled as *Boolean functions*. A Boolean function takes as input a bitstream or vector of length n from $\mathbb{F}_2^n$ and outputs a bitstream of length k from $\mathbb{F}_2^k$, where usually $k = 1$. Therefore, the understanding and analysis of Boolean functions play an important part in the analysis of cryptographic systems and also in this thesis.

Physical Unclonable Functions

Referring to Kerckhoff's principle, the security of all ciphers depends in practice heavily on the secrecy of the secret key. This goal is usually achieved through non-volatile memory which is included on-chip, but this approach remains vulnerable to hardware attacks [Qua+16; Sko05]. The effort to build circuits that detect an invasive attacker are more successful, but can be expensive, complex, unwieldy, battery draining, and/or unfitting for lightweight use. With this problems in mind, the notion of *Physical Unclonable Functions* (PUFs) was proposed. In general, a PUF P is a (partly) disordered system that can be challenged with a challenge c, upon which P responds with a corresponding response r. In the evaluating process, r depends on imperfections on a nanoscale level and environmental noise; hence, the partly disorder of P. There is no designated key storage on the PUF and due to the challenge/response protocol there is also no need for it. They were first introduced in 2002 as physical one-way functions by Pappu et al. [Pap+02] with the property to be easy to evaluate but hard to invert. Their PUF was an optical device using transparent tokens, which are randomly filled up with scattering particles. Hence, yielding a practically *unclonable* identifier, *e. g.*, as a alternative to smart cards. In this setup, a laser beam is used to create an unpredictable sparkle pattern, depending on the random arrangement of the scattering particles. Although optical PUFs are tamper-resistant, *i. e.* they are secure against invasive attacks, and hard to model and analyze, they have practical drawbacks. The orientation of the laser beam and the other mechanical components are highly sensitive and prevent practical use.

Arbiter PUF

After the optical PUF, many attempts to construct more practical PUFs were introduced. The next important step in PUF design was

the proposition to integrate PUFs on silicon *integrated circuits* (IC). This attempt has the advantage that the response of the PUF can be used directly on the chip.

Gassend et al. [Gas+04] proposed the *Arbiter PUF*, which exploits the intrinsic randomness of electronic components on ICs. This randomness is introduced during the fabrication process of the chip. Although multiple ICs, fabricated from the same template, produce the same output, each IC takes a different amount of time. Even the manufacturer of a chip has no influence on this imperfection, therefore, Arbiter PUFs can not be copied. In an Arbiter PUF two signals run through n stages on two different paths. The paths are determined by the challenge. Behind the stages an arbiter determines the response of the PUF depending on which input a signal arrived first. Gassend et al. [Gas+04] first showed and exploited the fact that Arbiter PUFs can be modeled via a linear additive model. In the following, more elaborated schemes using Arbiter PUFs were proposed to harden the circuit and model of the Arbiter PUF.

XOR Arbiter PUF

Gassend et al. [Gas+04] themselves suggested in their initial Arbiter PUF paper *Feed-Forward Arbiter PUFs*, which additional arbiters in intermediate points in the circuit, and delay buffers for the paths throughout the PUF circuit. Suh and Devadas [SD07] proposed to XOR multiple outputs of different challenges fed into one Arbiter PUF to create the final output of the Arbiter PUF. Thereafter, Devadas [Dev09] proposed to use multiple *individual* Arbiter PUFs in parallel and XOR their output to obtain the final response. This Arbiter PUF scheme is called *XOR Arbiter PUF*. Unfortunately, XOR Arbiter PUFs can be modeled by *machine learning* (ML) algorithms. Two different approaches make up the current state-of-the-art attacks on XOR Arbiter PUFs. Rührmair et al. [Rüh+10; Rüh+13] presented a *logistic regression* (LR) algorithm that uses the linear additive model of the Arbiter PUF to learn even its combined structure. The other approach was presented by Becker [Bec15] and uses the reliability of a response, *i. e.* the probability that a PUF answers with the same response to one challenge. This is possible due to environmental noise, that affects the outcome of Arbiter PUFs. Hence, at the current state-of-the-art XOR Arbiter PUFs are not secure.

PUF classification

Since PUFs are also wildly used for security purposes, there are many attempts to classify PUFs and define a formal security model. The two most important subtypes of PUFs are *Weak PUFs* and *Strong PUFs*. The meaning of the two terms changed throughout the short history of PUFs, but we will stick to the notions that prevail nowadays [Arm+11; Arm+16; RH14; RSS09]. Hereby, the classification depends mainly on the number of different challenges that a PUF can receive.

Weak PUFs are PUFs with just a few and fixed challenges. In the most applications of Weak PUFs the access to the response—or more generally to the challenge/response interface—is restricted. The most

notable implementation of Weak PUFs is the *SRAM PUF,* first presented by Guajardo et al. [Gua+07]. A SRAM PUF uses the intrinsic structure of the CMOS memory to create a unpredictable response.

Strong PUFs are PUFs with—in most cases—exponentially many challenges in the number of used stages. Thereby, a Strong PUF mostly responds with only one bit. Arbiter PUFs [Gas+04] and their extended versions [Dev09; SD07; MKP08] are the most widespread Strong PUFs.

The definitions and security models of PUFs started with Pappu's et al. notion of "easy to evaluate but difficult to invert" [Pap+02]. As PUFs were further developed, more and more demands were added to the definition like "physical unclonability" and "tamper resistance" [MV10]. Armknecht et al. [Arm+11] recognized that many implementations of PUF don't fulfill the proposed models anymore. They therefore dropped the 'unclonable' from the PUF notion and proposed the term of *Physical Functions*. However, the community does still use PUFs and even Arbiter PUFs, although they are basically broken and do not meet the definitions of PUFs anymore. Therefore, research still tries to find a suitable model and definition for PUFs. The most recent and universal attempt comes from Armknecht et al. [Arm+16] in 2016. They omit different classes of PUFs and define a general PUF security model with the following properties:

- *Output distribution*: Describes how the output of a PUF is distributed. This property is in regard not only to the probability that one specific output occurs, but also to the distance between the different outputs, *i. e.* after how many outputs a repetition occurs.
- *One-Wayness*: Indicates the probability that one can imply from a response to the corresponding challenge.
- *Unforgeability*: The potential of an attacker to predict a response without the possession of the actual PUF device. This property refers to mainly to the ability to create two PUF instances with the same imperfections.
- *Unclonability*: The ability of an attacker to come up with a model or copy of a PUF that behaves similar to the original. This property refers to modeling and machine learning attacks.
- *Indistinguishability*: The property of a PUF, that no one can deduce from given outputs which PUF instance was used.
- *Pseudorandomness*: The notion of how random the output of a PUF appears.
- *Tamper-Resilience*: The attribute of protection against invasive attacks.

Depending on which PUF one uses some of the properties do not fulfill any purpose. For example, for any Strong PUF with a one bit output the property of One-Wayness does not make any sense due to the fact that the probability to get one specific output is about 0.5 (provided the PUF has an approximately even output distribution). This thesis studies only a specific type of Strong PUFs—Arbiter PUFs—using the oracle access model. Therefore, we will ask only for the properties of *unclonability* and *output distribution*. The other properties are not applicable to Strong PUFs. Regarding the tamper-resilience: Tajik et al. [Taj+17] demonstrated that it is possible to completely characterize an Arbiter PUF and even its extended versions (see Section 6.3) via photonic emission analysis. Therefore, any adversary can break an Arbiter PUF while in possession of the PUF device.

We now look at modeling attacks on PUFs, the most successful type of attacks known. Any attacker can be understood as a probabilistic learning algorithm that returns a model of the PUF, as first used in a study by Ganji et al. [GTS16]. The probabilistic algorithm has two parameters $0 \leq \delta \leq 1$ and $0 \leq \varepsilon \leq 1$. We call δ the confidence parameter and ε the accuracy parameter of the algorithm. It outputs with probability $1 - \delta$ a model of the PUF that has error rate ε. We call an attack successful on a PUF if $\delta < \frac{1}{2} - c$ for a constant c and if $\varepsilon < \frac{1}{2} - c'$ for a constant c'. It is sufficient for δ to have a constant distance from $\frac{1}{2}$ to obtain a reasonable result, as we can improve the value of δ with s repetitions of the algorithm to $\delta' < 1 - \frac{1}{2^{cs}}$. The accuracy ε is less easy to be improved. Schapire [Sch90] presented the first provable polynomial-time algorithm that boosts the accuracy. If an attacker can successfully create a model of a PUF we call that PUF *learnable*.

Use of PUFs

PUFs can be used to fulfill mainly two goals in cryptography, *key extraction* and *authentication*. While Weak PUFs are primarily used as a key storage device, Strong PUFs can be used as key extraction circuits [Lim+05] and for the authentication of a device [Gas+04]. Classical authentication is mostly realized by challenge/response protocols [Dun01] with the help of secret keys. To avoid man-in-the-middle attacks the device that shall be authenticated needs two components: a keyed cryptographic module (such as a block cipher or a hash function) and an obfuscated secret key. Normally, a secret key is securely stored on the device, using non-volatile and obfuscated memory with restricted access. This process is, as stated above, expensive and non-volatile memory is known to be subject to tamper-attacks. Hu and Sunar [TW11, Sec. 13] are giving a state-of-the-art overview on tamper resistant memory. PUFs arise in hope to increase the robustness against physical attacks. Although many are known, e.g. side channel [DV14; XB14; Mer+11], invasive [Sko05], and fault injection attacks [DV14; Taj+17], Arbiter PUFs and their extended versions re-

main a cheap and lightweight alternative to obfuscated non-volatile memory and hence attract a lot of attention.

PUF derivatives

There are different ways to increase the hardness of the Arbiter PUF circuit to make them secure again against modeling attacks.

Increasing the circuit size: The idea herby is to use more individual Arbiter PUFs in parallel to get a larger circuit. At the current state of the PUF research this seems to be hard to realize due to the noisy responses of the individual Arbiter PUFs. Using 16 parallel individual Arbiter PUFs for an XOR Arbiter PUF yields a probability of about 76% to obtain the same response to a given challenge [Bec15]. This fact makes bigger XOR Arbiter PUFs unusable in practice.

Controlled PUF: Gassed et al. [Gas+02] proposed to restrict the access to the challenge/response interface of (Strong) PUFs via an algorithm. Any challenge would be preprocessed before given as input to the PUF, and any response would be post processed before returned to the enquiring entity. They called their suggestion *Controlled PUF*. Their paper doesn't contain any specific algorithms controlling the interface, as any such algorithm is specific to the use case.

Input transformation: Majzoobi et al. [MKP08] proposed an Arbiter PUF structure named *Lightweight Arbiter PUF*. They use a transformation of the challenge before feeding it to an XOR Arbiter PUF. Apparently, Lightweight Arbiter PUFs are more difficult to learn compared to XOR Arbiter PUFs [Rüh+13; RS14]. It is still not completely understood why the input transformation yields a more complex Arbiter PUF model. Unfortunately, the Lightweight Arbiter PUF is the only known advanced Arbiter PUF with an input transformation.

Combining function: Another opportunity to possibly harden the circuit of Arbiter PUFs could be to use different functions to combine the output of multiple Arbiter PUFs. The only function used at the current state-of-the-art is the XOR function [Dev09]. The change of the combining function from the known XOR to an arbitrary combining function shows similarities to Controlled PUFs, as one can interpret the combining of the outputs of the individual Arbiter PUFs as post processing.

This thesis attempts to gain a growing understanding for the use of alternative and arbitrary combining functions for multiple Arbiter PUFs used in parallel. For a combining function F, we call multiple individual Arbiter PUFs that are combined with that function *F-Combined Arbiter PUFs*. Because of the similarity of the structure of nonlinear combination generators to the structure of F-Combined Arbiter PUFs, we will first describe the composition of LFSRs and nonlinear combination generators and afterwards cover attacks on these structures. Thereafter, we will carry out these attacks on F-Combined Arbiter PUFs. It is possible to show that multiple attacks are applicable if the function F does not fulfill certain security properties. We therefore suggest *bent functions* as combining functions for multiple

Arbiter PUFs. The resulting *Bent Arbiter PUF* is not vulnerable to the investigated attacks.

1.1 RELATED WORK

Excellent overviews and handbooks of modern cryptography were published by Menezes et al. [MVV96], Goldreich [Golo1], and Katz [KL14]. As modern cryptography is tied to Boolean functions the study of O'Donnell's "Analysis of Boolean functions" [ODo14] is almost mandatory. Alternatively, a summary of relevant accomplishments and applications of Boolean functions in cryptography was published by Cusick and Stănică [CSo9]. The cryptographically important class of *bent functions* was first introduced by Rothaus [Rot76] in 1976 and was rediscovered for cryptographic use in 1989 by Meier and Staffelbach [MS89].

Physical unclonable functions (PUFs) that use manufacturing imperfections were first proposed by Pappu et al. [Pap+02] as *physical one-way functions*. As the physical one-way functions proposed by Pappu et al. are optical devices, Gassend et al. [Gas+04] introduced PUFs on silicon *integrated circuits* (ICs), so called *Arbiter PUFs* that use imperfections of ICs on a nanoscale level. Lee et al. [Lee+04], Lim et al. [Lim+05], and Suh et al. [SDo7] developed the idea of Arbiter PUFs further resulting in *XOR Arbiter PUFs*, a PUF using multiple individual Arbiter PUFs in parallel. Other types of PUFs on ICs are *Feed Forward Arbiter PUFs* [Gas+04; Lee+04], *Lightweight Secure PUFs* [MKPo8], and *Ring Oscillator PUFs* [SDo7].

With the introduction of Arbiter PUFs by Gassend et al. [Gas+04] they themselves discovered that Arbiter PUFs are learnable using *machine learning* algorithms, as Arbiter PUFs can be modeled as *linear threshold functions* (LTFs). The *perceptron algorithm*, one of the first ML algorithms for LTFs, was presented by Rosenblatt [Ros57] in 1957. The learning of LTFs was revived in the 1990s and led to new results on how fast LTFs can be learned [MT94] and how LTFs with noisy samples can be learned [Blu+96; BK13]. The first major attack on the XOR Arbiter PUF was executed by Rührmair et al. [Rüh+10; Rüh+13] using an advanced version of the perceptron algorithm. Becker [Bec15] introduced an attack on XOR Arbiter PUFs that exploits the noise of the used sample set. These two attacks are the current state-of-the-art regarding attacks on XOR Arbiter PUFs.

The emergence of PUFs in cryptography led to the need for definitions of security properties for PUFs, addressed for example in [RSSo9; Arm+11; RD13; PM15; Arm+16].

1.2 OUTLINE OF THE THESIS

The rest of the thesis will be structured as follows. The mathematical preliminaries—especially regarding Boolean functions and their different representations—are introduced in Chapter 2. LFSRs and nonlinear combination generators are presented in Chapter 3. In Chapter 4 we execute attacks against the combination function of nonlinear combination generators. The resulting security properties for nonlinear combination generators are presented in Chapter 5. Arbiter PUFs and their extended versions are introduced in Chapter 6 and similar attacks are carried out against $F-$Combined Arbiter PUFs in Chapter 7. Finally, Chapter 8 introduces a new extended Arbiter PUF structure, the *Bent Arbiter PUF*, and Chapter 9 draws the conclusions from our work.

2

PRELIMINARIES

In this chapter we introduce the mathematical preliminaries and definitions we will use throughout this work. We will start with general definitions before we define Boolean functions, their different representations, and tools for their use.

Let $\mathbb{F}_2$ be the finite group with two elements 0 and 1. With the addition modulo 2 (denoted by $\oplus$) and the usual multiplication (denoted by $\cdot$) the group $\mathbb{F}_2$ forms the finite field $(\mathbb{F}_2, \oplus, \cdot)$. Sometimes we omit the multiplication sign for better readability. Let $n \in \mathbb{N}$, then $\mathbb{F}_2^n$ denotes the vector spaces over $\mathbb{F}_2$ with component-wise addition and multiplication. Let $a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n$ be a vector, then the *Hamming weight* $w(a)$ is defined as the number of ones in the vector a, *i. e.*

$$w(a) = |\{i \in \{1, \ldots n\}; a_i = 1\}| \,.$$

For two vectors $a, b \in \mathbb{F}_2^n$ the *Hamming distance* $w_d(a, b)$ is defined as the number of positions in which a and b are different, *i. e.*

$$w_d(a, b) = |\{i \in \{1, \ldots, n\}; a_i \neq b_i\}| = w(a \oplus b) \,,$$

where $a \oplus b$ denotes the component-wise addition modulo 2 of two vectors. Note that x will denote an entire vector, and x_i his ith coordinate.

The *scalar product* or *inner product* $\langle a, b \rangle$ between two vectors $a, b \in \mathbb{F}_2^n$ is defined as

$$\langle a, b \rangle = \sum_{i=1}^{n} a_i b_i \mod 2 \,.$$

The scalar product can also be interpreted as a *linear function* $l_a(x) = a_1 x_1 + \ldots + a_n x_n = \langle a, x \rangle$ and $l_a(x) \oplus 1$ defines an *affine function*.

Let $l \in \mathbb{N}$, then $\mathbb{F}_2[X_1, \ldots, X_l]$ denotes the *polynomial ring* in l variables $X_1, \ldots, X_l$ over the field $\mathbb{F}_2$. Each polynomial $f \in \mathbb{F}_2[X_1, \ldots, X_l]$ has the form

$$f(X_1, \ldots, X_l) = \sum_{I \in \mathcal{I}} a_I X^I \,,$$

where $\mathcal{I} \subseteq \mathbb{N}^l$, $a_I \in \mathbb{F}_2$ are called the *coefficients* of f, and $X^I := X_1^{i_1} \cdot X_2^{i_2} \cdot \ldots \cdot X_l^{i_l}$ for all $I = (i_1, \ldots, i_l) \in \mathbb{N}^l$. A polynomial of the form $g = a_I X^I$ is called a *monomial*. If $I \in \mathbb{F}_2^l$ then g is called a *reduced monomial*, *i. e.* the exponents of the variables are either 1 or 0. If a polynomial consists only of reduced monomials it is called a *reduced polynomial*. The *degree* of a polynomial f is the maximal sum of $i \in I$ for which the corresponding coefficient a_I is not zero, *i. e.*

$$\deg(f) = \max\{\sum_{i \in I} i; I \in \mathcal{I}, a_I \neq 0\} \,.$$

We will always assume that the monomials are ordered in lexicographical order, so we can also identify a polynomial of the form $\sum_{I \in \mathcal{I}} a_I X^I$ with the ordered list of coefficients, *i. e.* $(a_{(0,\ldots,0)}, a_{(0,\ldots,1)}, \ldots, a_{(1,\ldots,1)})$. Note that this representation can be interpreted as a 2^n-dimensional vector.

The *character* of a group $(G, +)$ is a group homomorphism from G to the non-zero complex numbers, *i. e.* $\chi : G \to \mathbb{C}^*$, such that $\chi(x + y) = \chi(x)\chi(y)$. For $G = \mathbb{F}_n$ the characters are given by $\chi_\alpha(x) := e^{\frac{2\pi i}{|G|}\alpha x}$ where $\alpha \in G$. For $G = \mathbb{F}_{n_1} \times \ldots \times \mathbb{F}_{n_l}$ the character is defined by $\chi_\alpha(x) := \chi_{\alpha_1}(x_1) \cdot \ldots \cdot \chi_{\alpha_l}(x_l)$. If we consider the case that $G = \mathbb{F}_2$, then the only non-trivial automorphism from $\mathbb{F}_2$ to $\mathbb{C}^*$ is in the case $\alpha = 1$, *i. e.*

$$\chi_1(x) = e^{\frac{2\pi i}{|\mathbb{F}_2|}1x} = e^{\pi i x} = (-1)^x .$$

This yields a character function $\chi_f : \mathbb{F}_2^n \to \mathbb{C}^*; x \mapsto (-1)^{f(x)}$ for a function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$.

2.1 BOOLEAN FUNCTIONS

The most important concept throughout this work are Boolean functions as they allow to handle cryptographic primitives in more mathematical way:

Boolean functions and their truth table

Definition 2.1. A *Boolean function* f in n variables is a mapping from $\mathbb{F}_2^n$ to $\mathbb{F}_2$, *i. e.*

$$f : \mathbb{F}_2^n \to \mathbb{F}_2 .$$

The input of f is a n-dimensional vector, interpreted as a binary string. We will sometimes interpret input and output bits as -1 and 1 and not always as the elements of $\mathbb{F}_2$, 0 and 1. Hence we write Boolean Function as

$$f : \{-1, 1\}^n \to \{-1, 1\} .$$

To transform a input string from $\mathbb{F}_2$-notation to $\{-1, 1\}$-notation, the string is bitwise transformed by the function

$$\begin{aligned} b(x) : \mathbb{F}_2 &\to \{-1, 1\} \\ x &\mapsto (-1)^x . \end{aligned}$$

The *truth table* of f is the 2^n-dimensional vector defined by

$$(f(0,\ldots,0,0), f(0,\ldots,0,1), \ldots, f(1,\ldots,1,0), f(1,\ldots,1,1)) \in \mathbb{F}_2^{2^n} .$$

Note that the inputs of f are arranged in lexicographical order. For readability we will abbreviate the input tuple $(x_1, \ldots x_n)$ of f to just x. The term $f(x)$ will refer to the value of the evaluation of f on input x, whereas f will refer to the truth table or polynomial representation of the function.

Besides the truth table there are many different ways to represent a Boolean function. Each representation that we present can be displayed as a 2^n-dimensional vector and has its advantages that help with the analysis of various features of Boolean functions. The first alternative representation is that of the algebraic normal form.

Algebraic normal form

Definition 2.2. The *algebraic normal form* (ANF) of a Boolean function f is the unique representation of f as a reduced polynomial in n variables. We follow the explanations of MacWilliams and Sloane [MS77, Chap. 13] on how to retrieve the ANF from the truth table of a Boolean function.

Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. Then the ANF $\mathbf{f}(x)$ of f is a polynomial in

$$\mathbb{F}_2[X_1, \ldots, X_n]/(X_1^2 - X_1, \ldots, X_n^2 - X_n)$$

with the form

$$\mathbf{f}(x) = \sum_{i \in \mathbb{F}_2^n} c_i \cdot X_1^{i_1} \cdot \ldots \cdot X_n^{i_n},$$

where $c_i \in \mathbb{F}_2$ are called the coefficients of the ANF. The coefficient can be processed with the help of the truth table as follows

$$c_i = \sum_{\substack{x \in \mathbb{F}_2^n \\ x \leq i}} f(x),$$

where $x \leq i$ means that $x_j \leq i_j$, for all $1 \leq j \leq n$.

From the definition and computation of the ANF of $\mathbf{f}$ follows the definition of the *degree* of a Boolean function, it corresponds to the degree of the ANF, *i. e.*

$$\deg(f) = \deg(\mathbf{f}).$$

The notion of the truth table, the actual mapping, and the polynomial representation of a Boolean function are all interchangeable and will be most of the time simply denoted as f. From the context it is clear which representation of the Boolean function is meant.

An important property of a Boolean function in cryptography is the distribution of the output. Knowing the distribution and avoiding functions with a high bias allows a cryptographic designer to avoid statistical attacks on cryptographic systems.

Bias of Boolean functions

Definition 2.3. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function and let v_f be the corresponding truth table. The function f is called *unbiased* or *balanced* if the number of ones equals the number of zeroes in v_f, *i. e.* $w(v_f) = 2^{n-1}$. If $w(v_f) \neq 2^{n-1}$ then f is called *biased* or *unbalanced*. In other words, for a balanced function f the probability to obtain a one as a result corresponds to

$$\Pr_{x \sim \mathbb{F}_2^n}(f(x) = 1) = \Pr_{x \sim \mathbb{F}_2^n}(f(x) = 0) = \frac{1}{2}.$$

If f is unbalanced the distance ε_f of the probability that $f(x) = 1$ from $\frac{1}{2}$ is called the *bias* of f, *i.e.*

$$\Pr_{x \sim \mathbb{F}_2^n}(f(x) = 1) = \frac{1}{2} + \varepsilon_f \,.$$

Nonlinearity

The next definition expands the notions of Hamming weight and Hamming distance for Boolean functions. Building on the Hamming distance, the nonlinearity of a Boolean function is introduced. The nonlinearity describes the "distance" of a Boolean function from all linear and affine Boolean functions, *i.e.* how good an arbitrary Boolean function can be approximated by a linear or affine Boolean function.

Definition 2.4. Let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ be Boolean functions. As Boolean functions can be represented as vectors, the *Hamming weight* of a Boolean function is defined as the Hamming weight $w(f)$ of its truth table. Analogous, the *Hamming distance* between two functions f, g is defined as the Hamming distance $w_d(f, g)$ between the corresponding truth tables.

The nonlinearity of a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, denoted by $nl(f)$, is defined as

$$nl(f) = \min_{l_a \in \mathcal{A}_n} w_d(f, l_a) \,,$$

where $\mathcal{A}_n$ describes the class of all linear and affine functions in n variables.

Algebraic immunity

Another concept to define the approximability of Boolean function is the algebraic immunity. This concept was introduced by Meier et al. [MPC04] as a response to algebraic attacks.

Definition 2.5. Let $f, g : \mathbb{F}_2^n \to \mathbb{F}_2$ be Boolean functions with $\deg(g) < \deg(f)$ and let g be not the constant zero function. The function g is called an *annihilator* of f if

$$f \cdot g = 0 \text{ or}$$
$$(f \oplus 1) \cdot g = 0 \,.$$

Let

$$\mathrm{An}(f) := \{g : \mathbb{F}_2^n \to \mathbb{F}_2; g \text{ is an annihilator of } f\}$$

denote the set of all annihilators of f. Then the *algebraic immunity* (AI) of f is defined as

$$\mathrm{AI}(f) := \min\left(\{\deg(g); g \in \mathrm{An}(f)\}\right) .$$

The computation of the algebraic immunity is not known to be easy, but the following fact by Courtois and Meier [CM03] provide an upper bound for the algebraic immunity.

Fact 2.1. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. Then the algebraic immunity of f is bounded by*

$$AI(f) \leq \left\lceil \frac{n}{2} \right\rceil .$$

2.2 FOURIER AND WALSH TRANSFORM

After introducing Boolean functions and their basic properties we describe in this section two more representations of Boolean functions.

Definition 2.6. The *(discrete) Fourier transform* $\hat{f}$ of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is a mapping $\mathbb{F}_2^n \to \mathbb{R}$, defined by *Fourier transform*

$$\hat{f}(\alpha) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle} \cdot f(x) \,.$$

We call $\hat{f}(\alpha)$ the *Fourier coefficient* of f on α. The ordered vector of all Fourier coefficients on $\alpha \in \mathbb{F}_2^n$ is called the *Fourier spectrum* of f. The Fourier transform defines the coefficients of f with respect to the orthonormal basis of the group characters $(-1)^{\langle \alpha, x \rangle}$. The value $f(x)$ can be recovered by the *inverse Fourier transform*

$$f(x) = 2^{-n} \sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle} \cdot \hat{f}(\alpha) \,.$$

The Fourier spectrum of the two constant Boolean functions—mapping always to zero respectively to one—has an unusual form that will be useful later.

Lemma 2.1. *For the two constant Boolean functions,* $f_0(x) = 0$ *and* $f_1(x) = 1$ *for all* $x \in \mathbb{F}_2^n$*, the Fourier coefficients are*

$$\hat{f}_0(\alpha) = 0$$

for all $\alpha \in \mathbb{F}_2^n$ *for the constant zero-function and*

$$\hat{f}_1(\alpha) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle} = \begin{cases} 2^n & \text{if } \alpha = (0, \ldots, 0) \\ 0 & \text{otherwise} \,. \end{cases}$$

for the constant one-function.

Proof. Let $\alpha = (0, \ldots, 0)$. Then

$$(-1)^{\langle \alpha, x \rangle} = (-1)^0 = 1$$

for all $x \in \mathbb{F}_2^n$ and hence

$$\hat{f}_1(\alpha) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle} = \sum_{x \in \mathbb{F}_2^n} 1 = 2^n \,.$$

Let $\alpha \neq (0, \ldots, 0)$. Then we define two sets which contain the values $x \in \mathbb{F}_2^n$ for which $\langle \alpha, x \rangle$ evaluates to zero, respectively to one, *i. e.*

$$F_0 = \{x; \langle \alpha, x \rangle = 0\}$$
$$F_1 = \{x; \langle \alpha, x \rangle = 1\} \,.$$

As the Fourier coefficients on α to evaluate to zero, it is sufficient to show that $|F_0| = |F_1|$.

We fix $c \neq (0, \ldots, 0) \in \mathbb{F}_2^n$ with $\langle \alpha, c \rangle = 1$. Consider the function

$$\begin{aligned} g : F_0 &\to F_1 \\ x &\mapsto x \oplus c\,. \end{aligned}$$

Since for all $x \in F_0$ it holds that

$$\begin{aligned} &\langle \alpha, x \rangle = 0 \\ \Leftrightarrow\ &\langle \alpha, x \rangle = \langle \alpha, c \rangle \oplus 1 \\ \Leftrightarrow\ &\langle \alpha, x \oplus c \rangle = 1 \end{aligned}$$

the function g is well defined, hence $x \oplus c \in F_1$. Therefore, the inverse function of g is

$$\begin{aligned} g^{-1} : F_1 &\to F_0 \\ x &\mapsto x \oplus c\,, \end{aligned}$$

g is also bijective. Thus, it holds that $|F_0| = |F_1|$. Therefore the sum of the Fourier coefficients of f on $\alpha \neq (0, \ldots, 0)$ equals zero. □

The Fourier transform of the character function

$$\begin{aligned} \chi_f : \mathbb{F}_2^n &\to \{-1, 1\} \\ x &\mapsto (-1)^{f(x)} \end{aligned}$$

is called the Walsh transform:

Walsh transform

Definition 2.7. The *Walsh transform* $\hat{\chi}_f$ of a function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is a mapping $\mathbb{F}_2^n \to \mathbb{R}$, defined by

$$\hat{\chi}_f(\alpha) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle} \cdot (-1)^{f(x)} = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle \oplus f(x)}\,.$$

We call $\hat{\chi}_f(\alpha)$ the *Walsh coefficient* of f on α. The ordered vector of all Walsh coefficients on $\alpha \in \mathbb{F}_2^n$ is called the *Walsh spectrum* of f. As for the Fourier transform, the value of $f(x)$ can be recovered by the *inverse Walsh transform*

$$(-1)^{f(x)} = 2^{-n} \sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle} \cdot \hat{\chi}_f(\alpha)\,.$$

The Fourier and Walsh transform are tools to change the representation of a function and therefore allow us the simple computation of many cryptographic properties of Boolean functions. In particular, the Walsh spectrum indicates if an arbitrary Boolean function can be approximated by a linear Boolean function.

Lemma 2.2. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. If $\hat{\chi}_f(\alpha) = 2^n$ for any $\alpha \in \mathbb{F}_2^n$ then $nl(f) = 0$, that means that f is the linear function l_α.*

Proof. Let $\alpha \in \mathbb{F}_2^n$ be fixed with $\hat{\chi}_f(\alpha) = 2^n$. In that case the term $(-1)^{\langle \alpha, x \rangle \oplus f(x)}$ in the sum of the Walsh transform has always to come to 1. This is possible only if $\langle \alpha, x \rangle \oplus f(x) = 0$ and therefore the functions $l_\alpha = \langle \alpha, x \rangle$ and f are equal. Hence, the Hamming distance $w_d(l_\alpha, f) = 0$ concluding that $nl(f) = 0$. □

In the other case, if $\hat{\chi}_f(\alpha) = -2^n$, the values of f and l_α contradict each other in every entry of the truth table and hence f can be described by the affine function $l_\alpha \oplus 1$.

Corollary 2.1. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. If $\hat{\chi}_f(\alpha) = -2^n$ for any $\alpha \in \mathbb{F}_2^n$ then $w_d(l_\alpha, f) = 2^n$ and $w_d(l_\alpha \oplus 1, f) = 0$.*

In addition to Lemma 2.2 and Corollary 2.1, the Walsh coefficient on α also indicates if a Boolean function can be approximated by the linear function l_α or the affine function $l_\alpha \oplus 1$.

Lemma 2.3. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function and $\alpha \in \mathbb{F}_2^n$. If $\hat{\chi}_f(\alpha) = 0$ then f can not be approximated by l_α, i. e.*

$$\Pr_{x \sim \mathbb{F}_2^n}(f(x) = l_\alpha(x)) = \frac{1}{2}.$$

Proof. If $\hat{\chi}_f(\alpha) = 0$ then the term $(-1)^{\langle \alpha, x \rangle \oplus f(x)}$ in the sum of the Walsh transform evaluates to 1 for half of the inputs $x \in \mathbb{F}_2^n$ and to 0 for the other half. This yields the proposition. □

Walsh coefficient on zero

Another example for the usefulness of the Walsh transform is the Walsh coefficient on 0. It displays if a function is unbiased or not by "counting" the occurrences of zeros and ones in the truth table of a Boolean function:

Lemma 2.4. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. Then the function f is unbiased if and only if $\hat{\chi}_f(0) = 0$.*

Proof. Let f be unbiased, then the Walsh coefficient on 0 yields

$$\hat{\chi}_f(0) = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle 0, x \rangle \oplus f(x)} = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} = 0,$$

as half of the values of f are equal to zero and the other half equal one. Now let $\hat{\chi}_f(0) = 0$, as $\hat{\chi}_f$ always has 2^n summands exactly half of the summands must be zero and the other half must be one. This is only possible if f is unbiased. □

Balanced output of linear Boolean functions

The Walsh transform can also be used to survey properties of linear Boolean functions. For example it is possible to show that each linear function that is not the constant null-function ($f_0(x) = 0, \forall x \in \mathbb{F}_2^n$) is unbiased as the following lemma shows.

Lemma 2.5. *Let $\beta \in \mathbb{F}_2^n$ characterize the linear function*

$$\begin{aligned} l_\beta : \mathbb{F}_2^n &\to \mathbb{F}_2 \\ x &\mapsto \langle \beta, x \rangle \ . \end{aligned}$$

Then one has for all $\beta \in \mathbb{F}_2^n$

$$\hat{\chi}_{l_\beta}(0) = \sum_{x \in \mathbb{F}_2^n} (-1)^{l_\beta} = \sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \beta, x \rangle} = \begin{cases} 2^n & \text{if } \beta = (0, \ldots, 0) \\ 0 & \text{if } \beta \neq (0, \ldots, 0) \, . \end{cases}$$

Proof. The proof follows directly from Lemma 2.1. □

As one can see, the Walsh spectrum can reveal many properties of Boolean functions, such as the ability to approximate a Boolean function by a linear or affine Boolean function, and bias.

2.3 AVALANCHE EFFECTS AND CORRELATION IMMUNITY

In order to show how a change in the input of a Boolean function affects the probability that a change in the output occurs, we use the following property of the the strict avalanche criterion.

Strict avalanche criterion

Definition 2.8. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function and let $x = (x_1, \ldots, x_n)$ and $x_{\bar{i}} = (x_1, \ldots, x_{i-1}, \bar{x}_i, x_{i+1}, \ldots, x_n)$ denote two n-bit input vectors for f which differ only in one bit at position i, for $1 \leq i \leq n$. Let $v_i(x) = f(x) \oplus f(x_{\bar{i}})$ indicate the difference of the outputs of $f(x)$ and $f(x_{\bar{i}})$. The function f meets the *strict avalanche criterion* (SAC) if the probability that the bit $v_i(x)$ equals 1 is $\frac{1}{2}$ over the set of all possible input vectors x and $x_{\bar{i}}$, *i. e.*

$$\Pr_{x \sim \mathbb{F}_2^n}(v_i(x) = 1) = \frac{1}{2} \, .$$

The SAC was first defined by Webster and Tavares [WT85] in a study about the design of S-Boxes for block ciphers. The following Lemma 2.6 connects the SAC of a Boolean function f with its Walsh transform and follows directly from Definition 2.8 of the SAC and Lemma 2.4.

Lemma 2.6. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function, then f satisfies the SAC if and only if the function v_i is balanced for every $i, 1 \leq i \leq n$, i. e. $\hat{\chi}_{v_i}(0) = 0$.*

Correlation Immunity

Next, we introduce the *correlation immunity* of Boolean functions. This property was first defined by Siegenthaler [Sie84] and describes the immunity against the extraction of information one is able to obtain from a subset of input variables. We use a definition of Cusick and Stănică present a more tangible definition of the correlation immunity.

Definition 2.9. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function in n variables. Suppose that the inputs variables $x_1, \ldots, x_n$ are independent identically distributed binary random variables. Let $\mathscr{K} = \{x_{i_1}, \ldots, x_{i_k}\}$ be a fixed subset of the the input variables of order $k, 1 \leq k \leq n$. The function f is said to be *correlation immune of order k* if the probability that the k input variables $x_{i_1}, \ldots, x_{i_k}$ have a fixed set of values is always 2^{-k}, if the value of $f(x)$ is known, for any choice of the variables in $\mathscr{K}$, *i.e.*

$$\Pr_{(x_{i_1},\ldots,x_{i_k})\sim\mathbb{F}_2^k}(\text{fixed values for } x_{i_1}, \ldots, x_{i_k} \mid f(x)) = \frac{1}{2^k}.$$

That means that any subset $\mathscr{K}$ of input variables, with $\mid \mathscr{K} \mid \leq k$, is statistically independent from, and does not reveal any information about the output of f, if f is correlation immune of order k. Note that, although Siegenthaler defines it, a function in n variables can not be correlation immune of order n, since the output is always unambiguously determined by n input variables, independently from f.

Correlation immunity and Walsh transform

There are many conditions equivalent to that of the correlation immunity, for a comprehensive list see [CS09, Lemma 4.2]. Within this work we will only present an equivalent condition to correlation immunity based on the Walsh transform. The first proof for this condition was presented by Xiao and Massey [XM88]. They refer to the following Lemma 2.7 as the *spectral characterization* of a correlation immune function, because it restricts the Walsh spectrum.

Lemma 2.7. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. Then f is correlation immune of order k if and only if*

$$\hat{\chi}_f(\alpha) = 0$$

for all $\alpha \in \mathbb{F}_2^n$ with $1 \leq w(\alpha) \leq k$.

Proof. Let f be a correlation immune Boolean function of order k. The scalar product $\langle \alpha, x \rangle$ considers only at most k input variables and describes therefore the subset $\mathscr{K}$. Now w.l.o.g look at $f(x) = 0$. It follows from Lemma 2.5 and the Definition 2.9 of correlation immunity that, for a fixed output of f, the result of $\langle \alpha, x \rangle$ is balanced. So, the expression

$$\langle \alpha, x \rangle \oplus f(x) = \langle \alpha, x \rangle \oplus 0 = \langle \alpha, x \rangle$$

is balanced over all possible inputs x. This yields a equal number of zeros and ones in the sum of the Walsh transform for $f(x) = 0$. This is analogous true for $f(x) = 1$ and therefore $\hat{\chi}_f(\alpha) = 0$. □

One of the more important properties of correlation immune functions that Siegenthaler [Sie84, Thm. 1] discovered, covers their degree. As Fact 2.2 shows there is a tradeoff between the correlation immunity of a Boolean function and its degree.

Fact 2.2. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function that is correlation immune of order $k, 1 \leq k < n$. Then the degree $\deg(f)$ of f is less or equal to $n - k + 1$. Moreover, if f is additionally balanced, then the degree $\deg(f)$ of f is less or equal to $n - k$.*

To avoid attacks on cryptographic systems that exploit the correlation of Boolean functions, one is lead to believe that the highest correlation immunity is necessary. But, as Fact 2.2 shows, if a balanced Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is correlation immune of order $n - 1$ the degree of f is less or equal to 1. This means that f is a linear or affine function (leaving out constant functions). Therefore, avoiding correlation attacks can open the door for algebraic attacks that exploit the low degree of Boolean functions.

2.4 BENT FUNCTIONS

From the perspective of a cryptographic designer one desired property of Boolean functions would be a high nonlinearity to increase the potential effort of an attacker regarding especially algebraic attacks. In the terms of the Walsh transform this would translate to the requirement that each Walsh coefficient of a Boolean function would evaluate to zero. Unfortunately, this demand is shattered by Parseval's theorem.

Theorem 2.1 (Parseval's Theorem)**.** *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function. Then one has*

$$\sum_{\alpha \in \mathbb{F}_2^n} \hat{\chi}_f(\alpha)^2 = 2^{2n}\,.$$

Proof. It holds that

$$\begin{aligned}
\sum_{\alpha \in \mathbb{F}_2^n} \hat{\chi}_f(\alpha)^2 &= \sum_{\alpha \in \mathbb{F}_2^n} \hat{\chi}_f(\alpha) \cdot \hat{\chi}_f(\alpha) \\
&= \sum_{\alpha \in \mathbb{F}_2^n} \left(\sum_{x \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \rangle \oplus f(x)} \cdot \sum_{y \in \mathbb{F}_2^n} (-1)^{\langle \alpha, y \rangle \oplus f(y)} \right) \\
&= \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)} \left(\sum_{y \in \mathbb{F}_2^n} (-1)^{f(y)} \left(\sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \oplus y \rangle} \right) \right)\,.
\end{aligned}$$

Using Lemma 2.5 for the last term yields

$$\sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\langle \alpha, x \oplus y \rangle} = \begin{cases} 0 & \text{if } x \oplus y \neq (0, \ldots, 0) \\ 2^n & \text{if } x \oplus y = (0, \ldots, 0)\,. \end{cases}$$

This means that only the cases for $y = x$ evaluate to non-zero in the second sum, so

$$\begin{aligned}\sum_{\alpha\in\mathbb{F}_2^n}\hat{\chi}_f(\alpha)^2 &= \sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)}\left(\sum_{\substack{y\in\mathbb{F}_2^n\\ y=x}}(-1)^{f(y)}\cdot 2^n\right)\\ &= 2^n\cdot\sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)}\left(\sum_{\substack{y\in\mathbb{F}_2^n\\ y=x}}(-1)^{f(y)}\right)\\ &= 2^n\cdot\sum_{x\in\mathbb{F}_2^n}(-1)^{f(x)\oplus f(x)}\\ &= 2^n\cdot 2^n = 2^{2n}\,.\end{aligned}$$

□

Hence, from Theorem 2.1 follows that not all Walsh coefficients of a Boolean function can be zero. This also means that if for a Boolean function f the square of the Walsh coefficient on one $\alpha \in \mathbb{F}_2^n$ is zero, *i. e.* $\hat{\chi}_f(\alpha)^2 = 0$, another Walsh coefficient α' of f must have a higher value; that means f can be better approximated by $l_{\alpha'}$. A more preferable way to reduce the approximation by linear Boolean functions is to reduce the value of *all* Walsh coefficients of a Boolean function f to the same absolute value. This leads to the definition of bent functions.

Definition 2.10. Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function and let n be even. Then f is called *bent* if

$$\hat{\chi}_f(\alpha) = \pm 2^{\frac{n}{2}}\,,$$

for all $\alpha \in \mathbb{F}_2^n$.

Bias of bent functions

Bent functions were first introduced by Rothaus [Rot76]. The benefit of the fact that bent functions are hard to approximate by all linear and affine functions is since then wildly used in cryptography. But this advantage has also his drawbacks. For example, bent functions are not unbiased as the following lemma shows.

Lemma 2.8. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a bent function. Then f is biased, or more precisely the bias ε_f of f is*

$$\varepsilon_f = \frac{1}{2^{\frac{n}{2}+1}}\,.$$

Proof. Recall that the Walsh coefficient on 0 "counts" the occurrences of zeros and ones in the truth table of f (see Lemma 2.4). That

means we can rewrite $\hat{\chi}_f(0)$ using the usual definition of the probability—favorable events over all possible events—as

$$\begin{aligned}\hat{\chi}_f(0) &= \mid \{x; f(x) = 0\} \mid - \mid \{x; f(x) = 1\} \mid \\ &= 2^n \left(\Pr_{x \sim \mathbb{F}_2^n} (f(x) = 0) - \Pr_{x \sim \mathbb{F}_2^n} (f(x) = 1) \right) \\ &= 2^n \left(\left(1 - \Pr_{x \sim \mathbb{F}_2^n} (f(x) = 1) \right) - \Pr_{x \sim \mathbb{F}_2^n} (f(x) = 1) \right) \\ &= 2^n - 2^{n+1} \Pr_{x \sim \mathbb{F}_2^n} (f(x) = 1) \ .\end{aligned}$$

While f is bent the Walsh coefficient on 0 equals to $\pm 2^{\frac{n}{2}}$ and the above equation can be simplified to

$$\begin{aligned}\pm 2^{\frac{n}{2}} &= 2^n - 2^{n+1} \Pr_{x \sim \mathbb{F}_2^n} (f(x) = 1) \\ \Leftrightarrow \Pr_{x \sim \mathbb{F}_2^n} (f(x) = 1) &= \frac{2^n}{2^{n+1}} \pm \frac{2^{\frac{n}{2}}}{2^{n+1}} \\ &= \frac{1}{2} \pm \frac{1}{2^{\frac{n}{2}+1}} .\end{aligned}$$

Hence, the bias equals to $\varepsilon_f = 1/2^{n/2+1}$. □

Another important factor to have in mind is that bent functions have a relatively "low" degree compared to the maximum degree of 2^n that is possible with the number of used variables, as Rothaus pointed out [Rot76, p. 301 f.].

Fact 2.3. *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be a bent Boolean function. Then f has an algebraic degree of most $n/2 + 1$.*

2.5 LINEAR THRESHOLD FUNCTIONS

Besides bent functions, another important class of Boolean functions are linear threshold functions.

Definition 2.11. A Boolean function $f : \{-1, 1\}^n \to \{-1, 1\}$ is called a *linear threshold function* (LTF) if f can be represented as

$$f(x) = \operatorname{sgn}(a_0 + a_1 x_1 + \cdots + a_n x_n) \ ,$$

where $a_0, a_1, \ldots, a_n \in \mathbb{R}$ are constants. Note that here the addition is over $\mathbb{R}$ and the sign function is defined as $\operatorname{sgn}(x) = 1$ for $x \geq 0$, $\operatorname{sgn}(x) = -1$ for $x < 0$.

The inner part of LTF is a affine function $l_a(x) = a_0 + a_1 x_1 + \cdots + a_n x_n$ which maps from $\mathbb{R}^n$ to $\mathbb{R}$. Then $\operatorname{sgn}(l_a(x))$ is the ± 1-indicator of a halfspace in $\mathbb{R}^n$. A Boolean $f : \{-1, 1\}^n \to \{-1, 1\}$ is therefore a LTF if it has a "linear separator", *i.e.* a hyperplane in $\mathbb{R}^n$ that separates the points with label -1 from the points with label 1. We

will see in section 6.1 that a class of Physical Unclonable Functions can be modeled as LTFs and thereafter attacked by machine learning algorithms that search for the separating hyperplane.

There are many attempts to count LTFs in n variables and give bounds for their numbers (cf. [Win60; Ojh; YI65]).

Fact 2.4. *A lower and upper bound for the number of LTFs in n variables $N(n)$ are given by*

$$2^{(n(n-1)/2+8)} < N(n) \leq 2 \cdot \binom{2^n}{n}.$$

The given lower bound is from Yajima and Ibaraki [YI65], and the upper bound is from Winder [Win60]. This bounds are mainly obtained with help of determining half-spaces in a n–dimensional space.

Part II

ATTACKS ON NONLINEAR COMBINATION GENERATORS

This part is dedicated to *linear feedback shift registers* (LFSRs) and *nonlinear combination generators*. The scope of nonlinear combination generators is to disturb the inherent linearity of LFSRs. This is accomplished by nonlinear functions that combine the output of multiple in parallel used LFSRs. From known attacks on combining functions we will derive security properties and introduce thereafter *bent functions* as secure combining functions.

3

LINEAR FEEDBACK SHIFT REGISTER AND NONLINEAR COMBINATION GENERATORS

In this chapter we introduce *linear feedback shift registers* (LFSRs) and *nonlinear combination generators*. These structures are used as cryptographic primitives in stream ciphers to produce a pseudorandom key stream. We present LFSRs and nonlinear combination generator and attacks against them due to the similarity of their structure to Arbiter PUFs and XOR Arbiter PUFs (see Section 6.1). As LFSRs and nonlinear combination generators are a well studied subject in cryptography, we will try to deduce useful result for the design of Arbiter PUF schemes, a relatively new subfield in cryptography.

3.1 LINEAR FEEDBACK SHIFT REGISTER

A *shift register* is a logic circuit consisting of n stages containing one bit information, in which the output of one stage is connected to the input of the next stage. Thus, the data is shifted down the line through the circuit. A *linear feedback shift register* (LFSR) is a shift register of length n consisting of n stages $S_0, S_1, \ldots, S_{n-1}$. The content of a stage S_i is s_i. The input bit of the LFSR is determined by a linear function, that takes as input the output bits of the n stages. The data exchange is controlled by a clock. During each unit of the time the following operations are performed [MVV96, Section 6.2.1]:

1. the content s_0 of stage S_0 is output and forms part of the *output sequence*;
2. the content s_i of stage S_i is moved to stage S_{i-1} for each $i, 1 \leq i \leq n-1$;
3. the new content of stage S_{n-1} (the *feedback bit*) is computed by XOR-ing the content of a fixed subset of the stages $S_0, \ldots, S_{n-1}$ (compare Figure 3.1.1 for a example of a LFSR with 6 stages).

The initial state of the n stages is denoted by $(k_0, \ldots, k_{n-1})$ and is called the *seed (key)* of the LFSR. We refer to a LFSR by the ordered pair $(n, c(x))$, where

$$c(x) = 1 + c_1 x^1 + \ldots + c_n x^n \in \mathbb{F}_2[X]$$

is called the *connection polynomial* and $c_1, \ldots, c_n$ are called the *feedback coefficients*. The feedback coefficients establish the subset of stages

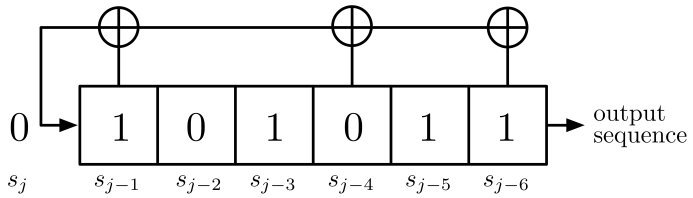


Figure 3.1.1: A LFSR with length $n = 6$.

used to compute the new content of the stage S_{n-1}. By the definition of the LFSR, the output sequence is determined by the following recursion equation:

$$s_j = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \ldots \oplus c_n s_{j-n}\,, \qquad j \geq n. \tag{3.1.1}$$

The first n bits of the output sequence correspond to the seed of the LFSR.

Example 3.1. Figure 3.1.1 illustrates a LFSR with 6 stages and the current state $1, 0, 1, 0, 1, 1$. As only the stages $1, 4$ and 6 are considered in the computation of the next input, the connection polynomial is $c(x) = 1 + 1x^1 + 0x^2 + 0x^3 + 1x^4 + 0x^5 + 1x^6$ and the next input s_j is $s_j = 1 \oplus 0 \oplus 1 = 0$.

The main usage for LFSRs in cryptography is as *pseudorandom number generator* (PRNG) in stream ciphers. In a stream cipher the plaintext $m = (m_0, \ldots, m_{t-1})$ is XOR-ed with a pseudorandom key stream $k = (k_0, \ldots, k_{-1})$ to obtain a ciphertext $c = (c_0, \ldots, c_{-1})$, i.e. $c_i = m_i \oplus k_i, 0 \leq i \leq t-1$. The only element of the LFSR that is kept secret and uses as secret key is the seed, neither the secrecy of the connection polynomial nor of the length benefit the security of the LFSR, as shown below.

There are two main weaknesses regarding standalone LRSRs due to the inherent linearity, that prevent their use in cryptography:

1. *Known-plaintext attack*: Let $L = (n, c(x))$ be a LFSR used as PRNG for a stream cipher. Let L be of length n with the connection polynomial $c(x)$ and the seed $(k_0, \ldots, k_{n-1})$. The output sequence $s_0, s_1, \ldots$ of the LFSR is used as key stream for a stream cipher, *i.e.* $c_i = m_i \oplus s_i, i \geq 0$. Assume an attacker A has possession of a substring of the plaintext and ciphertext of length $l \geq n$, that is, she knows $m_r, m_{r+1}, \ldots, m_{r+(l-1)}$ and $c_r, c_{r+1}, \ldots, c_{r+(l-1)}$. The goal of the attack is to determine the seed of the used LFSR. Here, A can compute the substring $s_r, s_{r+1}, \ldots, s_{r+(l-1)}$ of the output sequence by XOR-ing the plaintext and ciphertext bitwise. As the connection polynomial is not kept a secret, A can figure out every successor of $s_{r+(l-1)}$ in the output sequence with the recursion in Equation 3.1.1 and therefore decrypt the remaining message and future messages. Due

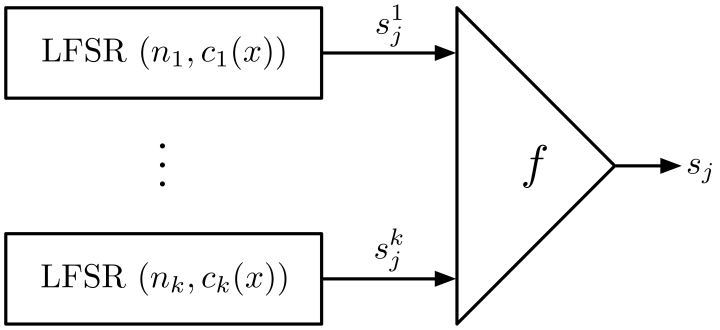


Figure 3.2.1: Scheme of a nonlinear combination generator.

to the linearity of the recursion in Equation 3.1.1 A is also able to compute gradually the predecessors of s_r and hence decrypt even the previous part of the message and past messages. The attacker can carry out the computation of the predecessors until A is in possession of the seed and hence the attack is successful.

2. *Berlekamp–Massey algorithm*: The Berlekamp–Massey algorithm takes as input a finite binary sequence and is a able to efficiently compute the shortest LFSR that will produce this sequence (for further details, see [MVV96, Section 6.2.3]). Therefore, an attacker can figure out—given an output sequence of a LFSR—the corresponding connection polynomial in a efficient amount of time. That means that the length of the LFSR and the privacy of the connection polynomial or the feedback coefficients does not boost the security of a LFSR.

To be able to still utilize LFSRs in cryptography, one can use multiple LFSRs in parallel, as shown in the next section.

3.2 NONLINEAR COMBINATION GENERATOR

One possibility to disturb the linearity of LFSRs is to use multiple LFSRs in parallel. The key stream is generated out of every of the k in parallel used LFSRs. In a clock cycle the bit s_j^i of the output stream of the ith LFSR is used as the ith input bit for a Boolean function $f : \mathbb{F}_2^k \to \mathbb{F}_2$ to compute the jth bit s_j of the key stream, for $1 \leq i \leq k$. Such a key stream generator is called a *nonlinear combination generator*, and f is called the *combining function*. Figure 3.2.1 shows the scheme for a nonlinear combination generator.

4

ATTACKS ON NONLINEAR COMBINATION GENERATORS

In the last chapter we presented LFSRs and basic attack against them. Due to this attacks, nonlinear combination generators were introduced with the goal to use the simplicity of LFSRs in a slightly complex setting to gain a secure stream cipher. Nonlinear combination generators use multiple LFSRs in parallel and combine their output with a combination function. In this chapter we will carry out two types of attack against the combination function. The *correlation attack* exploits a possible existing correlation between one or more inputs of the combining function and its output. The *algebraic attack* uses a low degree of the combining function to create a system of equations that needs to be solved.

4.1 KNOWN-PLAINTEXT ATTACK

First, we describe a brute force attack on nonlinear combination generators to have a runtime to compare the effort of the other attacks. Suppose that a nonlinear combination generator consists of k individual LFSRs $(n_1, c_1(x)), \ldots, (n_k, c_k(x))$. The bits s_j^i of the output sequences of the k LFSRs are used as inputs for the combining function $f : \mathbb{F}_2^k \to \mathbb{F}_2$ to form the output sequence s_j of the nonlinear combination generator (see Figure 3.2.1). Following Kerckhoffs' principle the security of the nonlinear combination generator should only depend on the seeds of the individual LFSRs and not on the connection polynomials or the combing function. This means we can assume that the connection polynomials and the combining function are public knowledge.

An attacker A can carry out a known-plaintext attack on a nonlinear combination generator as follows: Let A have knowledge of the output sequence $s_j, \ldots, s_{j+t-1}$ of the nonlinear combination generator. The goal of A is to find out the secret key, *i.e.* the seed, of the nonlinear combination generator. As the connection polynomials and the combining function are public knowledge, A can try out every possible seed for each LFSRs and simulate the output sequence of the nonlinear combination generator. If the simulated output sequence equals the given output sequence the attack is successful. Since a single LFSR of length n has $2^n - 1$ usable seeds (the seed consisting only

of nulls yields a null output sequence) the number of different seeds of the nonlinear combination generator is

$$\prod_{i=1}^{k}(2^{n_i}-1)\,.$$

So A needs about this amount of time to find out the seed used for the initialization the nonlinear combination generator.

4.2 CORRELATION ATTACKS

The main idea behind the correlation attack on nonlinear combination generators is to use the correlation between one or more inputs of the combining function and its output. Such a correlation allows, in combination with the knowledge of the output sequence of a nonlinear combination generator, to draw conclusions about the individual output of one or more LFSRs. This leads to a significant decrease of the time needed to break a nonlinear combination generator compared to a simple known-plaintext attack.

Correlation between one input and the output

Let the nonlinear combination generator consist of k LFSRs and a combining function $f : \mathbb{F}_2^k \to \mathbb{F}_2$ as in Section 4.1. Now suppose that there is a correlation between the output sequence of the LFSR $(n_1, c_1(x))$ and the output of the nonlinear combination generator, *i. e.*

$$\Pr_{s_j^i \sim \mathbb{F}_2}\left(f(s_j^1,\ldots,s_j^k) = s_j\right) = \frac{1}{2} \pm \varepsilon =: p\,,$$

where $0 < \varepsilon \leq \frac{1}{2}$. Hence, the above known-plaintext attack can improved: A computes for each of the $2^{n_1}-1$ usable seeds of the first LFSR its output sequence $s_j^1, s_{j+1}^1, \ldots, s_{j+t-1}^1$ and calculates the probability p' for coincidences between this output sequence and the given output sequence $s_j, \ldots, s_{j+t-1}$, *i. e.*

$$p' = \frac{\mid \{m \in \{j,\ldots,j+t-1\} \mid s_m^1 = s_m\} \mid}{2^t}\,.$$

If the probability p' is equal to the probability of the correlation p, then A has found the used seed for $(n_1, c_1(x))$ in $(2^{n_1}-1)\cdot t$ steps. The rest of the attack can be carried out as usual, with the difference that the seed for $(n_1, c_1(x))$ is already known. This yields a number of needed steps of

$$\prod_{i=2}^{k}(2^{n_i}-1)$$

to break the remaining LFSRs. Therefore, the total amount of steps needed for the correlation attack on k LFSRs decreases to

$$(2^{n_1}-1)\cdot t + \prod_{i=2}^{k}(2^{n_i}-1)\,.$$

Correlation between k inputs and the output

If a correlation between each of the output sequences of the LFSRs $(n_1, c_1(x)), \ldots, (n_k, c_k(x))$ and the output sequence $s_j, \ldots, s_{j+t-1}$ exists, the number of trials needed to break the nonlinear combination generator comes down to

$$t \cdot \sum_{i=1}^{k} (2^{n_i} - 1) \,.$$

Similarly, the correlation between a subset $\left\{ s_m^{i_1}, \ldots, s_m^{i_l} \right\}$ of inputs of order $l < k$ and the output of f can be exploited to decrease the runtime of the known-plaintext attack by a significant amount, *i. e.*

$$t \cdot \left(\prod_{i \in \{i_1, \ldots, i_l\}} (2^{n_i} - 1) \right) + \left(\prod_{i \notin \{i_1, \ldots, i_l\}} (2^{n_i} - 1) \right) .$$

4.3 ALGEBRAIC ATTACKS

The main idea behind algebraic attacks on cryptographic systems is to find a system of equations and thereafter so solve the found system. The challenge with solving systems of equations in the case of multivariate polynomials lies in its complexity. Garey and Johnson [GJ79] were the first to prove that the search for the root of a multivariate polynomial equation system is NP-complete. The first modern approach to use systems of equations for the cryptanalysis of cryptographic systems was executed by Kipnis and Shamir [KS99]. Their cryptanalysis of the HFE public key crypto-system uses a method called relinearization to reduce the degree of the polynomials in the equation system. The principle of relinearization is to substitute monomials with a degree higher than one with new, linear variables. Thereafter, the new linear equation system can be solved in time $\mathcal{O}(n^3)$ with the Gaussian elimination algorithm where n is the number of new and linearized variables. In the worst case the number of new variables is exponential in the degree of the equation system.

The algebraic attack is a known-plaintext attack. Suppose that a nonlinear combination generator consists of k individual LFSRs $(n_1, c_1(x)), \ldots, (n_k, c_k(x))$. The output bits $s_j^1, \ldots s_j^k$ of the output sequences of the k LFSRs are used as inputs for the combining function $f : \mathbb{F}_2^k \to \mathbb{F}_2$ to form the output sequence s_j of the nonlinear combination generator (see Figure 3.2.1). Suppose an attacker A wants to attack the given nonlinear combination generator, *i. e.* find out the individual seeds of the k LFSRs. As this is a known-plaintext attack, A has knowledge of the output sequence $s_j, \ldots, s_{j+t}, j \geq \max(n_1, \ldots, n_k)$. As each output bit of the nonlinear combination generator is a output of the combining function and each Boolean function $f : \mathbb{F}_2^k \to \mathbb{F}_2$

has unambiguous representation as a reduced polynomial (see Definition 2.2), we can construct the following equation system:

$$\begin{aligned} f(s_j^1, \ldots, s_j^k) &= s_j \\ &\ldots \\ f(s_{j+t}^1, \ldots, s_{j+t}^k) &= s_{j+t}\,. \end{aligned}$$

Let $d = \deg(f)$ be the degree of the polynomial f. With the Gaussian elimination and relinearization the equation system can be solved in time $\mathcal{O}\left(l^3\right)$ where l is the approximate number of linearized variables, respectively the number of monomials as the linearization replaces each monomial with a new variable. This number varies depending on the degree d of f. The number l of monomials can be approximated by

$$l \approx \left(\sum_{i=1}^{k} n_i\right)^d .$$

That means that the number of monomial grows exponentially with the degree of f. So, if the degree of f is high, the Gaussian elimination proves intractable. But, if f has a high degree one can search for a polynomial g with $\deg(g) < \deg(f)$ and $f \cdot g = 0$ or $(f \oplus 1) \cdot g = 0$. Recall that such a polynomial is called an annihilator of f and approximates f by a polynomial with a lower degree (see Definition 2.5). As Fact 2.1 shows, f can be always approximated by a polynomial with degree less than half of the degree of f, if f has an annihilator. Therefore, the Gaussian elimination can be applied even if f has a higher degree after finding a lower degree annihilator. The search for an annihilator has to be only executed once, since the annihilators of f do not change and even an annihilator that has not the degree $\mathrm{AI}(f)$ can turn out to be useful, if its degree is small enough for the Gaussian elimination.

Now suppose f has a low degree or an annihilator with low degree. The remaining question is how to model the output sequences of the k LFSRs to obtain a system of equations that can be linearized.

To model the output sequence of k LFSRs as an algebraic equation we first take a look on one LFSR. Recall the notation from Section 3.1. Let $c_i(x)$ be the connection polynomial of the ith LFSR $(n_i, c_i(x))$ with the feedback coefficients $c_1^i, \ldots, c_{n_i}^i$. The corresponding recursion equation is

$$s_j^i = c_1^i s_{j-1}^i \oplus c_2^i s_{j-2}^i \oplus \ldots \oplus c_{n_i}^i s_{j-n_i}^i\,.$$

Using the connection coefficients, the next bit of the output sequence of the LFSR $(n_i, c_i(x))$ can be produced by a matrix multiplication

with the knowledge of the previous n_i bits of the output sequence (similar to the recursion equation of the LFSR):

$$\begin{pmatrix} s^i_{j-n_i+1} \\ s^i_{j-n_i+2} \\ \vdots \\ s^i_{j-2} \\ s^i_j \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ c^i_{n_i} & c^i_{n_i-1} & c^i_{n_i-2} & \cdots & c^i_1 \end{pmatrix} \cdot \begin{pmatrix} s^i_{j-n_i} \\ s^i_{j-n_i+1} \\ \vdots \\ s^i_{j-2} \\ s^i_{j-1} \end{pmatrix},$$

where the matrix is from $\mathbb{F}_2^{n_i \times n_i}$ and the vector of previous elements from $\mathbb{F}_2^{n_i \times 1}$. Hence, from the bits $s^i_{j-n_i}, \ldots, s^i_{j-1}$ of the output sequence the bits $s^i_{j-n_i+1}, \ldots, s^i_j$ can be derived. Let L_i denote the matrix from above with the connection coefficients of the ith LFSR $(n_i, c_i(x))$, *i. e.*

$$L_i = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ c^i_{n_i} & c^i_{n_i-1} & c^i_{n_i-2} & \cdots & c^i_1 \end{pmatrix},$$

and let π_i denote the projection on the ith component of the input of π, *i. e.*

$$\pi_i(x_1, \ldots, x_n) := x_i\,.$$

Then, using multiple powers of L_i, the output sequence of the ith LFSR $s^i_j, s^i_{j+1}, \ldots$ for $j \geq n_i$ can be described as

$$\begin{aligned} s^i_j &= \pi_{n_i}\left(L_i \cdot \begin{pmatrix} s^i_{j-n_i} & \cdots & s^i_{j-1} \end{pmatrix}^T\right) \\ s^i_{j+1} &= \pi_{n_i}\left(L_i \cdot \left(L_i \cdot \begin{pmatrix} s^i_{j-n_i} & \cdots & s^i_{j-1} \end{pmatrix}^T\right)^T\right) \\ &= \pi_{n_i}\left(L_i^2 \cdot \begin{pmatrix} s^i_{j-n_i} & \cdots & s^i_{j-1} \end{pmatrix}^T\right) \\ &\ldots \end{aligned}$$

with knowledge of the previous n_i bits of the output sequence. The output sequence $s^i_j, s^i_{j+1}, \ldots$ for $j \geq n_i$ can also be described depending on the seed $s_0, \ldots, s_{n_i-1}$ of the ith LFSR as the following lemma shows.

Lemma 4.1. *Let $(n, c(n))$ be a LFSR with feedback coefficients $c_1, \ldots, c_n$ and let $s_0, \ldots, s_{n-1}$ denote its seed. Let the matrix L and the projection function be as described above. Then the jth bit s_j of the output sequence of $(n, c(n))$ can be described as*

$$s_j = \pi_n\left(L^{j-n+1} \cdot \begin{pmatrix} s_o & \cdots & s_{n-1} \end{pmatrix}^T\right), \tag{4.3.1}$$

for for all $j \in \mathbb{N}, j \geq n$.

Proof. We will verify the statement with an induction proof.

BASE CASE: Let $j = n$. Then

$$\begin{aligned} s_n &= \pi_n \left(L \cdot \begin{pmatrix} s_o & \cdots & s_{n-1} \end{pmatrix}^T \right) \\ &= \pi_n \left(\begin{pmatrix} s_1 & s_2 & \cdots & (c_n s_0 \oplus \ldots \oplus c_1 s_{n-1}) \end{pmatrix}^T \right) \\ &= c_n s_0 \oplus \ldots \oplus c_1 s_{n-1} \\ &= s_n \,. \end{aligned}$$

INDUCTION STEP: Let $k \in \mathbb{N}, k \geq n$ be given and suppose Equation 4.3.1 is true for $j = k$. Then

$$\begin{aligned} s_{j+1} &= \pi_n \left(L^{j+1-n+1} \cdot \begin{pmatrix} s_o & \cdots & s_{n-1} \end{pmatrix}^T \right) \\ &= \pi_n \left(L \cdot \left(L^{j-n+1} \cdot \begin{pmatrix} s_o & \cdots & s_{n-1} \end{pmatrix}^T \right)^T \right) \\ &= \pi_n \left(L \cdot \begin{pmatrix} s_{j-n} & \cdots & s_{j-1} \end{pmatrix}^T \right) \\ &= s_{j+1} \,. \end{aligned}$$

CONCLUSION: By the principle of induction, Equation 4.3.1 is true for all $j \in \mathbb{N}, j \geq n$.

□

Applying Lemma 4.1 the output sequence $s^i_j, s^i_{j+1}, \ldots$ for $j \geq n_i$ can be processed by

$$\begin{aligned} s^i_j &= \pi_{n_i} \left(L_i^{j-n_i+1} \cdot \begin{pmatrix} s^i_0 & \cdots & s^i_{n_i-1} \end{pmatrix}^T \right) \\ s^i_{j+1} &= \pi_{n_i} \left(L_i^{j-n_i+2} \cdot \begin{pmatrix} s^i_0 & \cdots & s^i_{n_i-1} \end{pmatrix}^T \right) \\ &\ldots \end{aligned}$$

Therewith, the bit s_j of the nonlinear combination generator can be modeled as

$$\begin{aligned} s_j = f \Bigg(\pi_{n_1} \left(L_1^{j-n_1+1} \cdot \begin{pmatrix} s^1_0 & \cdots & s^1_{n_1-1} \end{pmatrix}^T \right), \\ \ldots, \pi_{n_k} \left(L_k^{j-n_k+1} \cdot \begin{pmatrix} s^k_0 & \cdots & s^k_{n_k-1} \end{pmatrix}^T \right) \Bigg) \,. \end{aligned}$$

Recall, that this is a known plaintext attack and the attacker A has knowledge of the bits $s_j, \ldots, s_{j+t}$ of the output sequence. Then the following equation system can be established:

$$\begin{aligned} f\left(\pi_{n_1}\left(L_1^{j-n_1+1}\cdot\begin{pmatrix}s_0^1 & \cdots & s_{n_1-1}^1\end{pmatrix}^T\right),\right. \\ \left.\ldots,\pi_{n_k}\left(L_k^{j-n_k+1}\cdot\begin{pmatrix}s_0^k & \cdots & s_{n_k-1}^k\end{pmatrix}^T\right)\right) = s_j \\ \ldots \\ f\left(\pi_{n_1}\left(L_1^{j-n_1+1+t}\cdot\begin{pmatrix}s_0^1 & \cdots & s_{n_1-1}^1\end{pmatrix}^T\right),\right. \\ \left.\ldots,\pi_{n_k}\left(L_k^{j-n_k+1+t}\cdot\begin{pmatrix}s_0^k & \cdots & s_{n_k-1}^k\end{pmatrix}^T\right)\right) = s_{j+t} \end{aligned}$$

Expanding the matrix multiplications yields a polynomial equation system that can be solved with Gaussian elimination and relinearization in time $\mathcal{O}\left(l^3\right)$ as described above if f has a low degree. If f has a high degree one has to find the annihilator of f first. Note that the search for the annihilator has to be completed only once for each combining function f.

Correlation and algebraic attack are the most common attacks on nonlinear combination generator. After presenting these attacks on nonlinear combination generators with arbitrary combination functions we will discuss in the next section how to take precautions against these kind of attacks and what class of functions to use as combining functions.

5

SECURITY PROPERTIES OF COMBINING FUNCTIONS

In the previous chapter we carried out attacks on nonlinear combination generators targeting the combination function. To prevent these attack different security properties of combining functions are desired. We will introduce the desired properties and explain why they prevent the discussed attacks. In the following let a nonlinear combination generator consist of k in parallel used LFSRs

$$(n_1, c_1(x)), \ldots, (n_k, c_k(x))$$

and let $f : \mathbb{F}_2^k \to \mathbb{F}_2$ be the corresponding combining function. In the spirit of Kerckhoffs, assume that the connection polynomials and the combining function are public knowledge.

- *Degree of the combining function*: The degree $d = \deg(f)$ of the combining function corresponds to the degree of its ANF representation (recall Definition 2.2) and can be computed in time $k \cdot 2^k$ using the fast Fourier transform [MS77, Chap. 13]. The degree of f is an indicator for the ability to attack a nonlinear combination generator with an algebraic attack. A low degree of f allows for a easy relinearization and solving of the equation system as explained in Section 4.3. However, a high degree does not guarantee directly a resistance against the algebraic attack due to the existence of annihilators with relatively low degree (see Fact 2.1).

- *Nonlinearity of the combining function*: As the computation of the AI turns out to be rather difficult, another useful indicator for the ability to approximate a higher degree Boolean function is to compute its Walsh spectrum (recall Lemma 2.2, Corollary 2.1 and Lemma 2.3). While the AI is an indicator for the existence of an annihilator of f with low degree, the AI and the Definition 2.4 of nonlinearity do not provide a meaningful reference for the possibility to approximate f by all linear and affine functions. One can examine the distance from all linear and affine functions by computing the Walsh spectrum of f. This can be done in 2^{2k} steps. Together, the AI and the Walsh spectrum allow a relatively good evidence for the ability to approximate f by a function with a lower algebraic degree. Knowing this property of a combining function can also prevent especially algebraic attacks on nonlinear combination generators.

- *Distribution of the combining function*: The need for an unbiased distribution of the combining function does not follow directly from the aftermath of the attacks on nonlinear combination generators. Nevertheless, the distribution of f plays an important role, as an attacker could use the bias of the combination function to carry out for example statistical attacks that target the used protocol. The distribution can be verified by the computation of the Walsh coefficient on zero, *i. e.* $\hat{\chi}_f(0)$, in 2^k steps.

- *Correlation immunity*: The conclusion from Section 4.2 is the demand for a high correlation immunity of the combining function f. To prevent correlation attacks entirely f should be correlation immune of order $k-1$. Unfortunately, as Fact 2.2 shows, a correlation immunity of order $k-1$ leads to a degree of $\deg(f) \leq 2$. Combined with the demand for a balanced output of f the degree is even $\deg(f) \leq 1$. Therefore, one has to weigh the desired properties in this case, as a correlation immunity of high order and a high algebraic degree can not be accomplished at the same time. Recall from Lemma 2.7 that the correlation immunity can also be computed with the help of the Walsh spectrum of f in time $\mathcal{O}(2^k)$.

- *Strict avalanche criterion*: The SAC (see Definition 2.8) also follows not directly from the presented attacks on nonlinear combination generators. But the SAC interrelates with the correlation immunity, as the SAC requires the output of $f(x)$ to change with a probability of 0.5 if one input bit x_i is changed. If x_i has no impact on the output, f can not be correlation immune of order $n-1$. Therefore, with a high correlation immunity comes also the SAC. The SAC can also be computed in time $\mathcal{O}(2^k)$ using the Walsh transform (see Lemma 2.6).

As one can see from the tradeoff between the correlation immunity and the algebraic degree (Fact 2.2) not all desired security properties can be fulfilled at the same time. During the search for suitable cryptographic functions, Meier and Staffelbach came up with the notion of *perfect nonlinearity* [MS89] in 1989. As linear and affine functions are considered cryptographically weak, they looked for functions that have a maximum distance from all linear and affine functions. Meier and Staffelbach discovered that their perfect nonlinear function correspond to bent functions (see Section 2.4) introduced by Rothaus [Rot76] in 1976. Meier and Staffelbach showed that bent functions "have practical applications for block ciphers as well as stream ciphers", as they fulfill the above formulated security properties up to a satisfactory degree.

Recall that bent functions are defined as Boolean functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$ where each Walsh coefficient has the same absolute value, *i. e.*

$$\hat{\chi}_f(\alpha) = \pm 2^{\frac{n}{2}},$$

for all $\alpha \in \mathbb{F}_2^n$. Bent functions have not the highest possible degree for functions in n variables, but they can have a degree up to $n/2 + 1$ (see Fact 2.3). The most important property is the maximum distance from all linear and affine functions, that prevents a approximation through this class of functions. This property follows directly from the definition of bent functions. Although bent functions are not perfectly balanced, their bias approaches 0 very fast. Consider a nonlinear combination generator with $k = 12$ LFSRs and a bent function as a combining function. Then the bias of the output is

$$\frac{1}{2^{\frac{12}{2}+1}} = \frac{1}{2^7} = 0.0078125,$$

assuming a uniformly distributed input of the combining function. The bias is so close to 0 that no useful attack is derivable if the bent function has enough variables. Bent function are not correlation immune of order $n - 1$, in fact, bent functions are not correlation immune at all due to Lemma 2.7, since none of their Walsh coefficients is zero. But bent functions have on every subset of their input set the same correlation and it is therefore extremely small. That means, that an adversary can not practically exploit the existing correlation of bent functions. Similarly, the SAC is not fulfilled on all possible inputs of a bent function, but the deviation is to small to exploit.

In the last chapters we presented LFSR and their extended version, nonlinear combination generators. We investigated correlation and algebraic attacks on nonlinear combination generators and introduced security properties for combining functions to restrain these attacks. As it turns out, the reasons mentioned above cause bent functions to be the most suitable combining functions for nonlinear combination generators. In the following we will apply the same methodology to Arbiter PUFs and their extended version, due to the structural similarity with nonlinear combination generators.

Part III

ATTACKS ON COMBINING FUNCTIONS FOR PHYSICAL UNCLONABLE FUNCTIONS

Physical Unclonable Functions (PUFs) promise a lightweight alternative to non-volatile memory to safely store information on a chip. The most common type of PUFs is the *Arbiter PUF,* which exploits the intrinsic randomness of electronic components on integrated circuits. As single Arbiter PUFs can be modeled by machine learning (ML) algorithms, multiple Arbiter PUFs are used in parallel to combine their output by a combining function to prevent modeling. The *XOR Arbiter PUF,* which uses the XOR function to combine the outputs, turned out to be vulnerable against ML attacks, too. We will investigate the use of different combining functions. Therefore, we will compare combined Arbiter PUFs with nonlinear combination generators due to the structural similarities. We propose *Bent Arbiter PUFs* as a new Arbiter PUF class, which uses bent functions as combining functions.

6 PHYSICAL UNCLONABLE FUNCTIONS

As stated in the introduction, *Physical Unclonable Functions* (PUFs) emerged from the effort to bypass the drawbacks of non-volatile memory to store secret keys. In the last part of this thesis we introduced LFSRs and nonlinear combination generators and evaluated attacks on their combining function. Nonlinear combination generators have a very similar structure to XOR Arbiter PUFs. Each one of them uses multiple simple building blocks–LFSRs respectively Arbiter PUFs–in parallel to join their output with the help of a combining function. As nonlinear combination generators are vulnerable to attacks targeting their combining function, we will evaluate these attacks also for Arbiter PUFs combined with arbitrary Boolean functions.

Note, that from this chapter on we will sometimes use the $\{-1, 1\}$-notation instead of the $\mathbb{F}_2$-notation, since some of the functions have a more comfortable representation over this notation.

6.1 ARBITER PUFS

Integrated circuits (ICs) can be mass-produced to have identical logic functionality. Nevertheless, each IC has its own delay characterization due to manufacturing imperfections and process variations. That means that two ICs with the same logic functionality produce the same output on the same input, but they need a slightly different time to accomplish it. An *Arbiter PUF* on a silicon IC uses this delay characterization to allow a unique authentication of the underlying IC.

Gassend et al. [Gas+04] presented the first Arbiter PUF design on silicon ICs. Their PUF structure is depicted in Figure 6.1.1. A challenge $c \in \{0,1\}^n$ of n bits is given as input and one bit $r \in \{0,1\}$ is retrieved as output of the PUF. A total of n switches are connected in series and are characterizing two symmetrical paths over all the switches. The bits c_i of the challenge c determine if the paths through ith switch pass each other crossed or uncrossed. If $c_i = 1$ the paths cross each other. If $c_i = 0$ the paths don't cross each other. In Figure 6.1.1 the paths in the first switch run parallel as $c_1 = 0$. In the second switch the paths cross each other as $c_2 = 1$. A rising signal is presented as input to the first switch on both switch inputs. The two signals race each other through the switches on the two delay paths determined by the challenge. Behind the n switches is a comparator positioned, called *arbiter*, which outputs a 1 if a signal appears first on the lower input of the arbiter and 0 if a signal appears

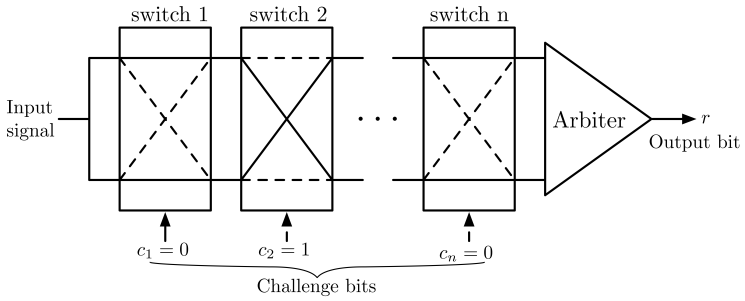


Figure 6.1.1: The basic structure of an Arbiter PUF circuit

first on the upper input. It does not matter which signal arrives first at the arbiter, in fact the two signals are indistinguishable. By the construction of the switches the two paths are symmetrical. Hence the delay difference of the two signals arriving at the arbiter depends only on the manufacturing imperfections on a nanoscale level. So, the response r of an Arbiter PUF to a given challenge c is influenced by the different delays of the paths through the switches.

6.1.1 *Additive Delay Model*

To model a noise free Arbiter PUF mathematically we first describe the intuitive approach and thereafter present the additive delay model introduced by Gassend et al. [Gas+04]. In this model we assume that the final delay of a path through the Arbiter PUF is the sum of the delays of the path segments through the individual components. As the two signals are indistinguishable, we are only interested in the delay difference of the two signals arriving at the arbiter. The intuitive approach needs the $4n$ delay values as variables to model an Arbiter PUF. The additive delay model by Gassend et al. operates on $2n$ variables. But it is possible to transform the representation of the Arbiter PUF to a linear threshold function (LTF) as introduced by Lim [Lim04] (compare Definition 2.11). This LTF model has become standard to describe the functionality of Arbiter PUFs with only $n+1$ variables.

The general goal of a model $f : \{0,1\}^n \to \{0,1\}$ of an Arbiter PUF $P : \{0,1\}^n \to \{0,1\}$ is to recreate the behavior of P on a challenge c, *i. e.*

$$\Pr_{c \sim \{0,1\}^n} (f(c) = P(c)) = 1 \,.$$

To reproduce the same response r to a challenge c, a model needs the delay difference of the two signals arriving at the arbiter; thereafter, r can be determined as described above. The intuitive approach models each switch with its delay values independently to create a model of the targeted Arbiter PUF. Thereafter, the delay differences can be added to determine r.

Let P be an Arbiter PUF. We first take a look at a single switch of P. Figure 6.1.2 shows such a switch with its four paths and their delay times $\delta_i^1, \delta_i^2, \delta_i^3, \delta_i^4$. We model the delay times as independent identically distributed random variables selected at fabrication [PDW89]. As the signals run through the switch only crossed or uncrossed, the path delay times appear only in pairs.

Definition 6.1. Let $P : \{0,1\}^n \to \{0,1\}$ be an Arbiter PUF with n switches and let each switch consist of four paths with delays $\delta_i^1, \delta_i^2, \delta_i^3, \delta_i^4$. The delay difference $\Delta\delta_i(c_i)$ caused by the ith switch depending on the challenge bit c_i is expressed by

$$\Delta\delta_i(c_i) = \begin{cases} \delta_i^1 - \delta_i^4 & \text{if } c_i = 0, \\ \delta_i^2 - \delta_i^3 & \text{if } c_i = 1\,. \end{cases} \tag{6.1.1}$$

The delay difference $\Delta D_i(c)$ that occurs *after* the ith switch determined by the challenge bits $c_1, \ldots, c_i \in c$ is displayed by

$$\Delta D_i(c) = \sum_{j=1}^{i} (-1)^{p_j^i(c)} \Delta\delta_i(c_i)\,, \tag{6.1.2}$$

where $p_j^i(c) = c_j \oplus c_{j+1} \oplus \ldots \oplus c_i$ is the parity function of the bits j to i of the challenge c. The term $(-1)^{p_j^i(c)}$ takes in account if the paths cross each other an odd number of times from switch j till switch i. In that case the delay difference has to be switched because the paths are now reversed to each other [MKP08].

To eliminate the case distinctions in Equation 6.1.1 the equation can be rewritten as follows.

Lemma 6.1. *The delay difference $\Delta\delta_i(c_i)$ determined by the ith switch can be rewritten to a case-distinction-free from*

$$\Delta\delta_i(c_i) = c_i \cdot (\delta_i^2 - \delta_i^4) + (1 - c_i) \cdot (\delta_i^1 - \delta_i^4)\,. \tag{6.1.3}$$

Proof. Evaluating $\Delta\delta_i(0)$ and $\Delta\delta_i(1)$ with Equation 6.1.3 yields the same results as in Equation 6.1.1. □

In the next step we reduce the number of variables needed to model an Arbiter PUF. Since we are only interested in the delay difference between the signals and not the actual runtime, we model in the variable $\delta_i^{(0)}$ the delay difference between the two signals in one switch if $c_i = 0$. In that case they pass the switch uncrossed, so

$$\delta_i^{(0)} = \delta_i^1 - \delta_i^4\,.$$

If $c_i = 1$ and the signals pass the switch crossed we model the delay difference in the variable

$$\delta_i^{(1)} = \delta_i^2 - \delta_i^3\,.$$

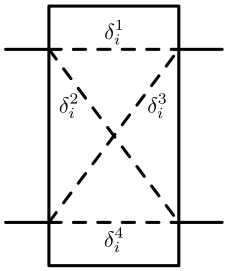


Figure 6.1.2: A single switch component

Hence, the delay difference $\Delta\delta_i$ between the two signals caused by the ith switch and input bit $c_i \in \{0,1\}$ can be described in case-distinction-free form by

$$\Delta\delta_i(c_i) = c_i \cdot \delta_i^{(1)} + (1 - c_i) \cdot \delta_i^{(0)} . \tag{6.1.4}$$

After modeling the delay difference between and after the switches an Arbiter PUF can be modeled as follows.

Lemma 6.2. *Let $P : \{0,1\}^n \to \{0,1\}$ be an Arbiter PUF, then P can be modeled via the function $f : \{0,1\}^n \to \{0,1\}$ as*

$$f(c) = \begin{cases} 0 & \text{, if } \Delta D_n(c) \leq 0 \\ 1 & \text{, if } \Delta D_n(c) > 0 \end{cases} .$$

Proof. To model a complete Arbiter PUF we consider the the delay difference after the nth and last switch and output 0 or 1 depending on where a signal arrived first on the arbiter, as described above. □

The first step towards the transformation of the model into a LTF representation with even fewer variables is to transform the model to accept input challenges from $\{-1,1\}^n$ instead from $\{0,1\}^n$. For this we substitute in Equation 6.1.4 the terms c_i and $(1 - c_i)$ to fulfill the requirements for the $\{-1,1\}$-notation.

Lemma 6.3. *The delay difference $\Delta\delta_i(c_i)$ caused by the ith switch and input bit $c_i \in \{-1,1\}$ can be expressed in a case-distinction-free form as*

$$\Delta\delta_i(c_i) = \frac{1}{2}\left(\left(\delta_i^{(0)} - \delta_i^{(1)}\right) c_i + \delta_i^{(0)} + \delta_i^{(1)}\right) .$$

Proof. As we transform a bit $c_i \in \{0,1\}$ into $\{-1,1\}$-notation via the function $b(x) = (-1)^x$, we seek a function that evaluates to $\rho_1(1) = 0$ and $\rho_1(-1) = 1$ for the $\delta_i^{(1)}$ part of Equation 6.1.4. This requirement is met by

$$\rho_1(x) = -\frac{1}{2}x + \frac{1}{2} .$$

For the $\delta_i^{(0)}$ term of Equation 6.1.4 we seek a function that fulfills $\rho_0(1) = 1$ and $\rho_0(-1) = 0$. This is met by

$$\rho_0(x) = \frac{1}{2}x + \frac{1}{2} .$$

Substituting the c_i and $(1 - c_i)$ terms in Equation 6.1.4 yields the expected result:

$$\begin{aligned}\Delta\delta_i &= \left(-\frac{1}{2}c_i + \frac{1}{2}\right) \cdot \delta_i^{(1)} + \left(\frac{1}{2}c_i + \frac{1}{2}\right) \cdot \delta_i^{(0)} \\ &= \frac{1}{2}\left(\left(\delta_i^{(0)} - \delta_i^{(1)}\right) c_i + \delta_i^{(0)} + \delta_i^{(1)}\right) .\end{aligned}$$

□

To model the delay difference $\Delta D_i(c)$ between the two paths after the ith switch, Equation 6.1.2 can be rewritten to take challenges $c \in \{-1, 1\}^i$ according to Lemma 6.3, *i. e.*

$$\Delta D_i(c) = \sum_{j=1}^{i} \frac{1}{2}\left(\left(\delta_j^{(0)} - \delta_j^{(1)}\right) c_j + \delta_j^{(0)} + \delta_j^{(1)}\right) \prod_{l=j}^{i} c_l \,, \tag{6.1.5}$$

where this time the term $\prod_{l=j}^{i} c_l$ expresses the parity function in $\{-1, 1\}$-notation. To determine the output in the $\{-1, 1\}$-notation it is now sufficient to take the sign of the final delay difference ΔD_n between the two signals.

Lemma 6.4. *Let $P : \{0,1\}^n \to \{0,1\}$ be an Arbiter PUF, then P can be modeled via the function $f : \{-1,1\}^n \to \{-1,1\}$ as*

$$f(c) = \operatorname{sgn}(\Delta D_n(c)) \,, \tag{6.1.6}$$

where $P(c) = b(f(c))$.

Proof. The proof follows from Equation 6.1.5, the definition of $b(x)$, and the behavior of the arbiter. □

We now have an additive delay model of an Arbiter PUF that operates on challenges from $\{-1, 1\}^n$ and $2n$ variables.

6.1.2 *LTF Representation of Arbiter PUFs*

To obtain an LTF representation of an Arbiter PUF that maps from $\{-1,1\}^n$ to $\{-1,1\}$ we start by expanding the delay difference after the nth switch from the Equation 6.1.5:

$$\begin{aligned}
\Delta D_n &= \sum_{i=1}^{n} \frac{1}{2} \left(\left(\delta_i^{(0)} - \delta_i^{(1)} \right) c_i + \delta_i^{(0)} + \delta_i^{(1)} \right) \prod_{j=i}^{n} c_j \\
&= \sum_{i=1}^{n} \left(\left(\underbrace{\frac{1}{2}\delta_i^{(0)} - \frac{1}{2}\delta_i^{(1)}}_{:=\alpha_i} \right) c_i + \underbrace{\frac{1}{2}\delta_i^{(0)} + \frac{1}{2}\delta_i^{(1)}}_{:=\beta_i} \right) \prod_{j=i}^{n} c_j \\
&= \sum_{i=1}^{n} (\alpha_i c_i + \beta_i) \prod_{j=i}^{n} c_j \\
&= (\alpha_1 c_1 + \beta_1) \prod_{j=1}^{n} c_j + (\alpha_2 c_2 + \beta_2) \prod_{j=2}^{n} c_j + \ldots + (\alpha_n c_n + \beta_n)\, c_n \\
&= \beta_1 \prod_{j=1}^{n} c_j + (\alpha_1 + \beta_2) \prod_{j=2}^{n} c_j + \ldots + (\alpha_{n-1} + \beta_n) c_n + \alpha_n \,. \quad (6.1.7)
\end{aligned}$$

Using this expansion an Arbiter PUF can be modeled as a LTF as the following Lemma shows.

Lemma 6.5. *Let* $P : \{0,1\}^n \to \{0,1\}$ *be an Arbiter PUF. Then* $f : \{-1,1\}^n \to \{-1,1\}$ *models* P *as LTF with*

$$f(c) = \operatorname{sgn}\left(w_0 + w_1 x_1 + \ldots + w_n x_n\right) ,$$

where $w_i = \frac{1}{2}(\delta_i^{(0)} - \delta_i^{(1)} + \delta_i^{(0)} + \delta_i^{(1)})$ *for* $1 \leq i \leq n$, $\delta_0^{(0)} = \delta_0^{(1)} = 0$, $w_0 = \frac{1}{2}(\delta_n^{(0)} - \delta_n^{(1)})$, *and* $x_i = \prod_{j=i}^{n} c_j$.

Proof. Using the expansion of ΔD_n in Equation 6.1.7 and Lemma 6.4 yields the required LTF form of the Arbiter PUF model f. □

We now can model an Arbiter PUF as a LTF. Note the transformation of the challenge bits of c to an input x for the LTF. This transformation is a bijection, we can convert the challenge back and forth without obstacle.

Noise and stability

Up until now we only considered a noise-free environment for PUFs. But implementing Arbiter PUFs in hardware produces noisy responses due to environmental influences such as temperature- or voltage fluctuations. Delvaux and Verbauwhede [DV13] were the first to propose an Arbiter PUF model that includes the environmental noise. The noise for all switches and the arbiter can be summarized and modeled in one noise source at the arbiter input. Therefore, an additional time difference ΔD_{Noise} is introduced, which is assumed to be normally distributed with zero mean and a variance σ^2_{Noise} depending on measurement conditions and implementation. This additional

time difference expands the Arbiter PUF model from Lemma 6.4 so that the response r of an Arbiter PUF to a given challenge c is determined by

$$f(c) = \operatorname{sgn}\left(\Delta D_n(c) + \Delta D_{\text{Noise}}\right) .$$

Entering the noise time difference into the equation leads us to the definition of the stability of a challenge.

Definition 6.2. Let $f : \{-1,1\}^n \to \{-1,1\}$ be the model of an Arbiter PUF P. For a given challenge $c \in \{-1,1\}^n$ let $\Delta D_n(c)$ be the model delay difference and let ΔD_{Noise} follow a normal distribution. Then the *stability* $\operatorname{Stab}(c)$ of c is defined as

$$\operatorname{Stab}(c) = \Pr_{\Delta D_{\text{Noise}}}\left(\operatorname{sgn}\left(\Delta D_n(c)\right) = \operatorname{sgn}\left(\Delta D_n(c) + \Delta D_{\text{Noise}}\right)\right) .$$

Hence, the stability describes the probability that P responds to a challenge with the corresponding model answer, *i. e.* the current environmental noise does not affect the outcome of the PUF.

The linear additive delay model and the representation as LTF are the main reasons why a single Arbiter PUF is breakable and therefore unusable as a standalone primitive in a cryptographic context. The next sections elaborates the learnability of Arbiter PUFs.

6.2 ATTACKS ON ARBITER PUFS

To analyze attacks on Arbiter PUFs we use an oracle access model. An adversary that wants to perform an attack on a PUF communicates with an oracle. Through the oracle, the adversary can obtain any number of genuine PUF responses to (adaptively) chosen challenges. Notice that responses to equal challenges do not necessarily have always the same value, since responses can be noisy. Note that we do not consider attacks where the adversary has physical possession of the PUF as additional and very specific assumptions would have to be made. As Tajik et al. [Taj+17] demonstrated, it is possible to completely characterize an Arbiter PUF and even its extended versions (see Section 6.3) via photonic emission analysis.

Any attacker can be understood as a probabilistic PAC learning algorithm that returns a model of the PUF, as first used in a study by Ganji et al. [GTS16]. The probabilistic algorithm has two properties $0 \leq \delta \leq 1$ and $0 \leq \varepsilon \leq 1$. We call δ the confidence parameter and ε the accuracy parameter of the algorithm. It outputs with probability $1 - \delta$ a model of the PUF that has error rate ε. We call an attack successful if $\delta < \frac{1}{2} - c$ for a constant c and if $\varepsilon < \frac{1}{2} - c'$ for a constant c'. It is sufficient for δ to have a constant distance from $\frac{1}{2}$ to obtain a reasonable result, as we can improve the value of δ with s repetitions of the algorithm to $\delta' < 1 - \frac{1}{2^{cs}}$. The accuracy ε is less easy to be improved. Schapire [Sch90] presented the first provable polynomial-time algorithm that boosts the accuracy.

An attacker can model the complete Arbiter PUF if she knows all the delay values of the switches. Therefore, the security of an Arbiter PUF depends on the secrecy of the delay values, respectively on the inability of an attacker to model the LTF of an Arbiter PUF. With the oracle model the adversary can collect any number of challenge response pairs (CRPs), this set is separated in a training set and in a test set. The training set is used by a supervised machine learning (ML) algorithm to create a model with probability $1 - \delta$. The test set is used to validate the model and determine the error rate ε. The goal of the adversary is to create from this set a LTF model of an Arbiter PUF with a preferably small error rate. The learning of LTFs is a well-studied field in ML. It consists mainly in the search for the separating hyperplane of the LTF. It is been long known that noise-free LTFs are learnable in polynomial time using linear programming (LP). In this case, each collected CRP provides one linear constraint and an attacker can simply use a LP solver to solve them [Kar84]. Noisy LTFs are also learnable in time $poly(n, 1/\varepsilon)$ [Blu+96]. Using the perceptron algorithm [Ros57] noisy LTFs can be learned even faster in polynomial time [Byl94; MT94].

Perceptron algorithm

The *perceptron algorithm* is a learning algorithm for binary classifiers. It simulates single layered neural network by mapping the input vector x of one neuron to 1, if the scalar product with a vector of weights w is greater than a given threshold. To train a neuron on a set of input vectors, one iterates through the input vectors. If the scalar product of one input with the weights produces an correct result, nothing is changed. If the result is incorrect, each one of the neuron's weights is changed in the direction to produce the correct output. The inputs are normally in $\{-1, 1\}$-notation and the weights are rational numbers. If the training set can be learned by the neural network the perceptron algorithm converges to a solution that satisfied all training data.

Gassend et al. [Gas+04] demonstrated in their initial presentation of the Arbiter PUF on silicon ICs that, using the additive delay model, Arbiter PUFs are learnable with the perceptron algorithm. Using an advanced version of the perceptron, the logistic regression (LR) algorithm, Rührmair et al. [Rüh+10; Rüh+13] presented an even faster way to learn the LTF model of Arbiter PUFs in polynomial time with a polynomial number of CRPs and a error rate under 0.05. They achieved the best learning results using the LR algorithm with the Resilient Propagation (RProp) gradient descent function. The easy modeling of Arbiter PUFs led to efforts to harden the circuits and models of Arbiter PUFs.

6.3 XOR AND COMBINED ARBITER PUFS

The power of modeling attacks against Arbiter PUFs led to various propositions to increase the complexity of the circuit and models of

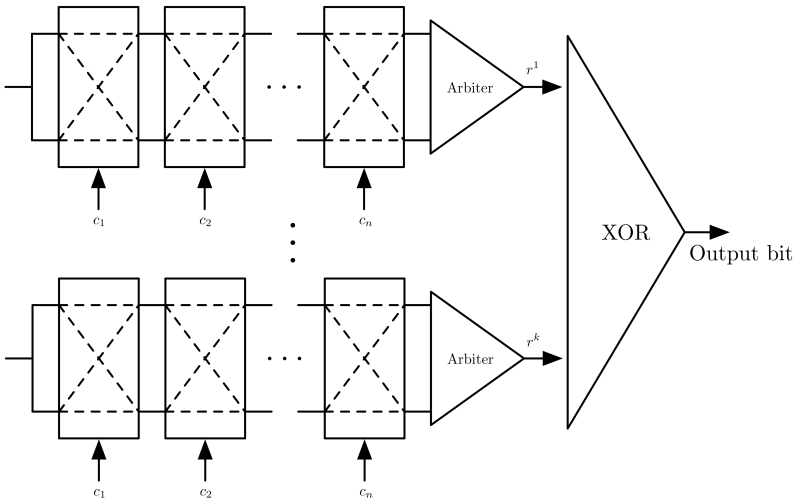


Figure 6.3.1: The structure of an XOR Arbiter PUF

Arbiter PUFs. Gassend et al. [Gas+04] suggested Feed-Forward Arbiter PUFs, a PUF with additional arbiters in intermediate points in the circuit, and delay buffers for the paths through the PUF circuit. Suh and Devadas [SD07] proposed to XOR multiple outputs of different challenges fed into one Arbiter PUF to create the final output of the Arbiter PUF. Thereafter, Devadas [Dev09] proposed to use multiple Arbiter PUFs in parallel and XOR the output of the individual Arbiter PUFs to obtain the final response. Unfortunately, due to the instability of individual Arbiter PUFs it is not possible to use more than approximately 12 Arbiter PUFs in parallel without decreasing the stability of the combined PUF construction enormously [Bec15, Table 2].

Figure 6.3.1 depicts the structure of such an *XOR Arbiter PUF* in detail. Let $P_1, \ldots, P_k$ be k different Arbiter PUFs, each consisting of n stages. If a challenge $c \in \{0,1\}^n$ is presented to the XOR Arbiter PUF $XORPUF$ each individual Arbiter PUF $P_i, 1 \leq i \leq k$ receives the same challenge c and evaluates it to $P_i(c) \in \{0,1\}$. To produce the output $r \in \{0,1\}$ of $XORPUF$ the XOR of the individual responses is computed, *i. e.*

$$XORPUF(c) = P_1 \oplus \ldots \oplus P_k\,.$$

To model an XOR Arbiter PUF we use the LTF model in $\{-1,1\}$-notation presented in Section 6.1.1. The XOR of k variables in $\{0,1\}$-notation corresponds to the multiplication of k variables in $\{-1,1\}$-notation, *i. e.* let $z_1, \ldots, z_k \in \{-1,1\}$ then

$$XOR(z_1, \ldots, z_k) = \prod_{i=1}^{k} z_i\,.$$

Lemma 6.6. *Let $XORPUF : \{-1,1\}^n \to \{-1,1\}$ be an XOR Arbiter PUF with k individual Arbiter PUFs $P_1, \ldots, P_k$ modeled by their respective*

LTF $f_i : \{-1,1\}^n \to \{-1,1\}, 1 \leq i \leq k$. Then XORPUF can be modeled by a LTF $F : \{-1,1\}^n \to \{-1,1\}$ consisting of $\mathcal{O}((n+1)^k)$ weights.

Proof. Let

$$f_i(c) = \operatorname{sgn}\left(w_0^{(i)} + w_1^{(i)} x_1 + \ldots + w_n^{(i)} x_n\right)$$

be the model of the ith Arbiter PUF P_i, where $x_i = \prod_{j=i}^{n} c_j$. This leads to the following equation of the model of an XOR Arbiter PUF:

$$\begin{aligned} XORPUF(c) &= \prod_{i=1}^{k} f_i(c) \\ &= \prod_{i=1}^{k} \operatorname{sgn}\left(w_0^{(i)} + w_1^{(i)} x_1 + \ldots + w_n^{(i)} x_n\right) \\ &= \operatorname{sgn}\left(\prod_{i=1}^{k} \left(w_0^{(i)} + w_1^{(i)} x_1 + \ldots + w_n^{(i)} x_n\right)\right), \end{aligned}$$

while the last transformation is possible due to the homomorphic property of the sign function. Expanding the product of the LTFs yields a polynomial threshold function (PTF)

$$\begin{aligned} &= \operatorname{sgn}\Big(w_0^{(1)} \cdot \ldots \cdot w_0^{(k)} + \\ &\quad \left(\left(w_0^{(1)} \cdot \ldots \cdot w_0^{(k-1)} w_1^{(k)}\right) + \ldots + \left(w_1^{(1)} w_0^{(2)} \cdot \ldots \cdot w_0^{(k)}\right)\right) x_1 \\ &\qquad + \ldots + w_n^{(1)} \cdot \ldots \cdot w_n^{(k)} \cdot x_n^k\Big). \end{aligned}$$

Substituting the terms in front of the x variables and linearizing the variables with a power greater than one leads to the LTF representation of an XOR Arbiter PUF. Due to the fact that not all combinatorial possibilities of the variables appear (some collapse, like in the second term) the LTF representation does not contain exactly $(n+1)^k$ monomials rather than a number close to $(n+1)^k$. □

Learning XOR Arbiter PUFs

Since the weakness of a single Arbiter PUF circuit was known from the beginning, the research concentrated on the machine learning of improved Arbiter PUFs schemes. Currently there are two known state-of-the-art ML algorithms that can learn XOR Arbiter PUFs. Rührmair et al. [Rüh+10; Rüh+13] use the LR algorithm with RProp to learn XOR Arbiter PUFs up to $n = 128$ stages and $k = 6$ parallel used Arbiter PUFs. They need approximately 500.000 CRPs for the biggest attacked structure at achieve an error rate of 0.01. Their attack is carried out both on simulated PUFs and on Silicon Arbiter PUFs. Becker [Bec15] uses a different approach to learn XOR Arbiter PUFs. He utilizes the reliability of the challenges to learn individual Arbiter PUFs; Becker's notion of reliability corresponds to our concept of stability (see Definition 6.2). His attack uses a *Covariance*

Matrix Adaptation Evolution Strategy (CMA-ES) machine learning evolution algorithm that assesses fitness based on the reliability of the responses of the individual Arbiter PUFs. To learn physically realizable constructions of XOR Arbiter PUFs ($k \leq 12$) in reasonable time with a relatively small number of CRPs (approximately 150.000 CRPs for similar XOR Arbiter PUFs learned by Rührmair et al.), Becker's attack uses the information revealed by the reliability of individual challenges.

Fact 6.1. *Current XOR Arbiter PUFs structures with up to $n = 128$ stages and $k \leq 12$ parallel used Arbiter PUFs can be learned in time $\mathcal{O}(n^k)$ using the LR algorithm by Rührmair et al. [Rüh+10; Rüh+13] and in time poly$(n, k, \frac{1}{\varepsilon})$ using the CMA-ES algorithm by Becker [Bec15], where ε is the stability of the individual Arbiter PUFs.*

We will in the following discuss shortly the runtimes of the ML attacks on XOR Arbiter PUFs by Rührmair et al. and Becker.

Attack of Rührmair et al.

For the learning of XOR Arbiter PUFs, the best results were delivered again by the LR algorithm with RProp as optimization method. The smallest number of CRPs N_{CRP} needed to obtain a result with error rate ε grows in time poly$(n, k, \frac{1}{\varepsilon})$ with n being the number of stages in a PUF and k the number of XORed PUFs. Additionally the number of trials N_{trial} needed to achieve the global optimum is required because the LR algorithm is not guaranteed to find it on the first try. As Rührmair et al. point out [Rüh+13, Sec. 4.], the number of trials grows in $\mathcal{O}(d_x/N_{CRP})$, where d_x is the dimension in which the model of the PUF is separable and learnable with $d_x \approx (n+1)^k/k!$ for $k \ll n$. This yields an exponential number of trials in k before the LR algorithm outputs the global optimum, hence, a total time $\mathcal{O}(n^k)$.

Attack of Becker

To learn physically realizable constructions of XOR Arbiter PUFs ($k \leq 12$) in reasonable time with a relatively small number of CRPs, Becker's attack uses information revealed by the stability of individual challenges. In each iteration the reliability-based machine learning algorithm will converge to one of the individual Arbiter PUFs of the XOR PUF. As some of the Arbiter PUFs are "harder" to learn than others, the algorithm will more often converge to some PUF instances that to others. Therefore, if only a few Arbiter PUFs remain, one can learn the remaining PUF models using the LR algorithm of Rührmair et al. One iteration of the CMA-ES algorithm needs time poly$(n, k, \frac{1}{\varepsilon})$ to learn one individual Arbiter PUF, where ε is the stability of the PUFs. The restarts the algorithm needs to perform can be approximated by $\mathcal{O}(k \cdot \log \frac{k}{k-4})$ if one wants to learn all but 4 Arbiter PUFs, as the learning of the individual Arbiter PUFs can be modeled as the *Coupon collector's problem* [Fel50, p. 213]. Becker himself states that for the last few Arbiter PUFs the traditional LR algorithm is faster than the evolution strategy algorithm. That means that the number of restarts for the CMA-ES algorithm has no significant impact on the runtime and can be treated as a constant.

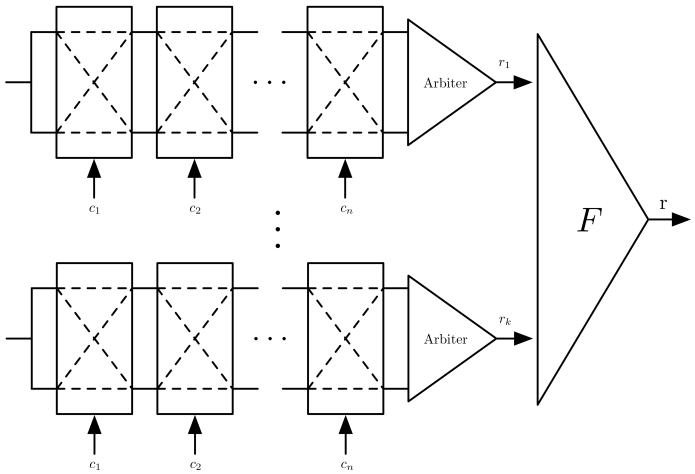


Figure 6.3.2: Structure of a Combined Arbiter PUF

The Fact 6.1 leads to three possible methods for increasing the hardness of Arbiter PUF structures. The first technique would be to try to increase the number of Arbiter PUFs used in parallel. This approach is at the current state of Arbiter PUFs defeated by environmental noise and the stability of Arbiter PUFs. The second method is to transform the challenge before feeding it to the individual Arbiter PUFs like attempted by Majzoobi et al. [MKP08]. The third method is to evaluate the combining method of the individual Arbiter PUFs. We will in the following study the the last method. The evaluation of combining functions of nonlinear combination generators revealed weaknesses in their structure and led to the use of more secure functions—*bent functions*. Using XOR to combine the outputs of multiple Arbiter PUFs is at the current state of the research the only employed function. We extend the notation of the XOR Arbiter PUF to allow different usage of the outputs of multiple individual Arbiter PUFs. This modular approach leads hopefully in some cases to an increased complexity and security of the whole PUF structure.

F−Combined Arbiter PUF

Definition 6.3. Let $P_1, \ldots, P_k$ be k Arbiter PUFs in n stages each with their respective models $f_i : \{-1,1\}^n \to \{-1,1\}, 1 \leq i \leq k$. Let $F : \{-1,1\}^k \to \{-1,1\}$ be a Boolean function in k variables. The function F is called the *combining function* and such an Arbiter PUF is called *$F-$Combined Arbiter PUF.* Figure 6.3.2 depicts the structure of such a $F-$Combined Arbiter PUF. Upon raising a challenge $c \in \{-1,1\}^n$ to the structure each Arbiter PUF evaluates c to $r_i = f_i(c)$. The final response $r \in \{-1,1\}$ is computed by processing $r = F(r_1, \ldots, r_k)$.

Using this definition of composite Arbiter PUFs we will carry out attacks on combining functions that violate one or more security properties presented in Chapter 5. We chose this approach due to the similarity of the structure of nonlinear combination generators and $F-$Combined Arbiter PUFs. As neglecting these security properties

allows attacks against the PUF structure we suggest using bent functions as combining functions.

7 ATTACKS ON COMBINED ARBITER PUFS

In the last chapter we presented Arbiter PUFs and their widespread usage as XOR Arbiter PUFs. Unfortunately, both schemes are learnable using ML algorithms. In an attempt to harden the structure of Arbiter PUFs and prevent machine learning, we defined $F-$Combined Arbiter PUFs. This scheme of multiple Arbiter PUFs uses an arbitrary combining function to process the output of the individual Arbiter PUFs.

In this chapter we will examine the robustness of $F-$Combined Arbiter PUFs against attacks targeting the combining function in the case that certain security properties are not met by the combining function F.

To study the runtime results of the attacks in this chapter we use a lower and an upper comparison value. The lower comparison value is given by the runtime $\mathcal{O}(n^k)$ of the attack by Rührmair et al. [Rüh+10; Rüh+13] and the runtime $\text{poly}(n, k, \frac{1}{\varepsilon})$ of the attack by Becker [Bec15] against XOR Arbiter PUFs. We will use this comparison value as XOR is the algebraic function with the lowest degree to combine multiple outputs of individual Arbiter PUFs. As upper comparison value serves the theoretical runtime of the learning of the concept class

$$\mathcal{C} = \{F(f_1, \ldots, f_k) \mid f_1, \ldots, f_k : \{-1,1\}^n \to \{-1,1\} \text{ are LTFs}\},$$

where $F : \{-1,1\}^k \to \{-1,1\}$ is an arbitrary Boolean function. This corresponds to the model of $F-$Combined Arbiter PUFs. The concept class $\mathcal{C}$ is learnable in time $n^{\mathcal{O}(k^2/\varepsilon^2)}$, where ε is the desired accuracy (see [ODo14, p. 131]).

In the first section we will look upon correlating combining functions, the second section considers algebraic attacks on the combining function.

7.1 CORRELATION ATTACKS

Considering the successful attack on nonlinear combination generators with a combining function whose output correlates with one or multiple inputs (see Section 4.2), we will carry out the attack on $F-$Combined Arbiter PUFs with correlating combining functions.

Suppose that $P : \{-1,1\}^n \to \{-1,1\}$ is an $F-$Combined Arbiter PUF structure as in Definition 6.3. Let $F : \{-1,1\}^k \to \{-1,1\}$ be an arbitrary Boolean function that is the combining function for k

Arbiter PUFs modeled by k LTFs $f_1, \ldots, f_k : \{-1,1\}^n \to \{-1,1\}$. Now suppose that the first input x_1 of F correlates with the output of F, *i.e.*

$$\Pr_{x_i \sim \{-1,1\}} (F(x_1, \ldots, x_k) = x_1) = \frac{1}{2} \pm \eta =: p\,,$$

Correlation between one input and the output

where $0 \leq \eta \leq \frac{1}{2}$.

An attacker A can attack and learn P via the oracle access model exploiting the correlation in the following manner. First, A collects a number N of CRPs $(c_1, r_1), \ldots, (c_N, r_N)$ from its interaction with the oracle. Using the correlation of F with its first input A can model the first Arbiter PUF of P. To that end, the attacker learns the LTF f_1 using the LR algorithm as described in Section 6.2 with a desired error rate $\varepsilon, 0 < \varepsilon \leq \frac{1}{2}$ of the model. This procedure is possible, since the LR algorithm is noise resistant. This version of the LR algorithm is called the robust logistic regression (rLR) algorithm [Pre82; BK12; BK13]. As the rLR algorithm always converges [BK12], A is able to learn f_1 with an accuracy $(1/2 + |\eta| - \varepsilon/2)$ from $\mathcal{O}(n/\varepsilon^2 \cdot \ln 1/\varepsilon\delta)$ noisy samples, where δ is the confidence parameter [Byl94]. The runtime of the LR algorithm depends on the size of the sample set. As in our case the number of samples depends on the targeted accuracy of $(1-\varepsilon)$, the runtime to create the model of f_1 is $\mathcal{O}(n/\varepsilon^2 \cdot \ln 1/\varepsilon\delta)$. Note that the runtime of the learning becomes exponential, if the error rate ε is exponentially small.

After the learning of the first LTF, the remaining $(k-1)$ LTFs can be learned in time $n^{\mathcal{O}((k-1)^2)}$ with the general attack on combined LTFs for a constant error rate. This yield a total runtime of

$$\mathcal{O}\left(\frac{n}{\varepsilon^2} \cdot \ln \frac{1}{\varepsilon\delta}\right) + n^{\mathcal{O}((k-1)^2)}$$

for the learning of a 1–correlating F-Combined Arbiter PUF using $\mathcal{O}(n/\varepsilon^2 \cdot \ln 1/\varepsilon\delta)$ samples. Note that, if the correlation is close to $\frac{1}{2}$, then A can not learn f_1 with an error rate that is significantly better than $\frac{1}{2}$. The runtime can improve depending on the the combining function F, as the learning of combined LTFs can improve, if a more efficient learning algorithm can be found for specific combination functions than the general attack on combined LTFs.

Correlation between k input and the output

The correlation attack on $F-$Combined Arbiter PUFs can be extended to $(k-1)$–correlation combining functions. But the runtime and accuracy of the model depend much more on the combining function F than in the case of only one existing correlation in the combining function. Suppose again that $P : \{-1,1\}^n \to \{-1,1\}$ is an $F-$Combined Arbiter PUF with $F : \{-1,1\}^k \to \{-1,1\}$ as combining function. The goal of an attacker A is once more to learn the k underlying Arbiter PUFs of P $f_1, \ldots, f_k$ in their LTF representation. Now

suppose that F has a individual correlation between each input and the output, *i. e.*

$$\begin{aligned}
\Pr\left(F(x_1,\ldots,x_k)=x_1\right) &= \frac{1}{2}\pm\eta_1 =: p_1\\
\Pr\left(F(x_1,\ldots,x_k)=x_2\right) &= \frac{1}{2}\pm\eta_2 =: p_2\\
&\ldots\\
\Pr\left(F(x_1,\ldots,x_k)=x_k\right) &= \frac{1}{2}\pm\eta_k =: p_k\,.
\end{aligned}$$

By the robustness of the rLR algorithm A can try to learn each LTF independently from one set of samples. As the LR algorithm chooses its starting weights randomly, each iteration of the algorithm with the same sample sets will yield another LTF. For this step A needs $m = \mathcal{O}(n/\varepsilon^2 \cdot \ln 1/\varepsilon\delta)$ samples, where ε is the smallest targeted error rate and δ is the confidence. Hence, A requires $k \cdot \mathcal{O}(m)$ time to model the LTFs. Note that due to the correlation each LTF f_i can only be modeled with an accuracy of $(1/2 + |\eta|)$. The total error becomes larger while iterating the individual LTFs.

Additionally, A cannot target which LTF she wants to learn. Therefore, the algorithm has to be restarted $\mathcal{O}(k)$ times according to the *Coupon collector's problem* [Fel50, p. 213]. After this, the next step is to arrange the k LTFs in the right input order for F. Supposing that each combination order of inputs is equally possible, there are $k!$ possible arrangements of the LTFs $f_1,\ldots,f_k$. This yields a total worst-case runtime of

$$\mathcal{O}\left(k\cdot\frac{n}{\varepsilon^2}\cdot\ln\frac{1}{\varepsilon\delta}\right)+k!\,.$$

For a constant k and $k \ll n$ this attack has a much better runtime in n than the general attack on combined LTFs which has a runtime of $\mathcal{O}(n^{\mathcal{O}((k-1)^2)})$. But, with larger $k \approx n$ the runtime of the $(k-1)$–correlation attack in n becomes by far worse than the general attack because of the factorial term in the runtime. To counter correlation attacks, correlation immune combiner functions are required.

7.2 ALGEBRAIC ATTACKS

In this section we will try to model an $F-$Combined Arbiter PUF as a system of equations to mimic the algebraic attack on nonlinear combination generators in Section 4.3. It turns out, that due to the LTF representation of Arbiter PUFs and the inherent sign function, classical algebraic attacks are not possible against Arbiter PUF constructions.

Suppose again that $P : \{-1,1\}^n \to \{-1,1\}$ is an $F-$Combined Arbiter PUF structure as in Definition 6.3. Let $F : \{-1,1\}^k \to \{-1,1\}$ be an arbitrary Boolean function that is the combining function for k

Arbiter PUFs modeled by k LTFs $f_1, \ldots, f_k$ that map from $\{-1,1\}^n$ to $\{-1,1\}$. Recall that each LTF has the representation

$$f_i(x) = \operatorname{sgn}\left(w_0^{(i)} + w_1^{(i)}x_1 + \ldots + w_n^{(i)}x_n\right) = \operatorname{sgn}\left\langle w^{(i)}, x\right\rangle$$

for all $1 \leq i \leq k$, where $\left\langle w^{(i)}, x\right\rangle$ is in this case the scalar product over $\mathbb{R}$.

Using the oracle access model, an attacker A can attack P as follows. First, she collects N CRPs $\{(c^{(j)}, r^{(j)}); 1 \leq j \leq N\}$. The goal of the attack is to use each CRP to create an equation and thereafter to solve this system of equations to derive the transformed delay values for the $F-$combined Arbiter PUF. As elaborated in Section 4.3 the ability to solve such a system depends firstly on the degree of the equations. If the degree of the system of equations is 1, one can use the Gaussian elimination to solve this system in time $\mathcal{O}(l^3)$, where l is the number of used variables. If the degree is higher than 1, the most promising approach is to apply relinearization to the system of equations and substitute each monomial of a higher degree with a new, linear variable and thereafter apply the Gaussian elimination algorithm. Another method is to search for an annihilator of f with a lower degree than f and substitute f with its annihilator in the equation system.

Using the set of collected CRPs A can devise from the jth pair $(c^{(j)}, r^{(j)})$ the equation

$$F\left(f_1(x^{(j)}), \ldots, f_k(x^{(j)})\right) = r^{(j)},$$

where x^i the the transformation of the challenge c^j according to Lemma 6.5. Expanding this notation yields

$$F\left(\operatorname{sgn}\left\langle w^{(1)}, x^{(j)}\right\rangle, \ldots, \operatorname{sgn}\left\langle w^{(k)}, x^{(j)}\right\rangle\right) = r^{(j)}.$$

The major challenge now arises during the further expansion of the combining function F. A polynomial of a Boolean function can consist only of two operations, $\cdot$ and $\oplus$ in $\mathbb{F}_2$ notation, respectively max and $\cdot$ in the $\{-1,1\}$ notation, respectively AND and XOR in general. More precisely, the $\cdot$ operation in $\mathbb{F}_2$ notation

$$\begin{aligned} \mathrm{AND}_2 : \mathbb{F}_2^2 &\to \mathbb{F}_2 \\ (x_1, x_2) &\mapsto x_1 \cdot x_2 \end{aligned}$$

corresponds to the max operation in $\{-1,1\}$ notation

$$\begin{aligned} \mathrm{AND}_2 : \{-1,1\}^2 &\to \{-1,1\} \\ (x_1, x_2) &\mapsto \max(x_1 x_2) \end{aligned}$$

and the $\oplus$ operation in $\mathbb{F}_2$ notation

$$\begin{aligned} \mathrm{XOR}_2 : \mathbb{F}_2^2 &\to \mathbb{F}_2 \\ (x_1, x_2) &\mapsto x_1 \oplus x_2 \end{aligned}$$

corresponds to the $\cdot$ operation in $\{-1,1\}$ notation

$$\begin{aligned} \mathrm{XOR}_2 : \{-1,1\}^2 &\to \{-1,1\} \\ (x_1, x_2) &\mapsto x_1 \cdot x_2 \,. \end{aligned}$$

As we are currently in the $\{-1,1\}$-notation, the XOR of two LTFs can be represented as the sign of the multiplication of the two scalar products due to the multiplicative nature of the sign function, *i. e.*

$$\mathrm{sgn}\left(\langle a, b\rangle\right) \cdot \mathrm{sgn}\left(\langle c, d\rangle\right) = \mathrm{sgn}\left(\langle a, b\rangle \cdot \langle c, d\rangle\right) \,.$$

That is, that if one encounters an XOR in the expansion process of F one can factorize the sign function outside of the brackets.

The AND of two LTFs on the other hand corresponds to the max function. Unfortunately, there exists no algebraic representation over $\mathbb{F}_2$ that can eliminate the sign function of the individual LTFs since the sign function is generally not additive, *i. e.*

$$\mathrm{sgn}(a+b) \neq \mathrm{sgn}(a) + \mathrm{sgn}(b) \,.$$

That means that there is no useful algebraic expansion for F over $\mathbb{F}_2$, if F contains one ore more AND operations.

If F consists just of XOR operations, then the algebraic expansion of one equation from the jth CRP (c^j, r^j) of F is equally to the representation in Lemma 6.6, *i. e.*

$$\mathrm{sgn}\left(\prod_{i=1}^{k} \left\langle w^{(i)}, x^{(j)} \right\rangle\right) = r^{(j)} \,.$$

In that case, F is a linear function and P is a regular XOR Arbiter PUF. The sign function outside the brackets does only allow a inequality

$$\prod_{i=1}^{k} \left\langle w^{(i)}, x^{(j)} \right\rangle \leq r^{(j)} \,,$$

therefore an algebraic attack is not directly possible. But, using relinearization each CRP yields a linear constraint and an attacker can use a linear programming (LP) solver to solve the inequalities [Kar84]. As the expansion yields $\mathcal{O}(n^k)$ variables (cf. Section 4.3) the LP solver would take exponential time in k. As the LR algorithm is more efficient that LP solver, an adversary could use alternatively the discussed ML algorithms in Fact 6.1 by Rührmair et al. and by Becker to attack and learn P.

If the combining function contains AND operations, F is not a linear function over $\mathbb{F}_2$ and an algebraic attack is not realizable, since there is no feasible algebraic representation of F. But, as Beigel et al. [BRS95] proved, the sign function can be approximated via a polynomial over $\mathbb{R}$ with grade 2^n. That means that an algebraic representation over $\mathbb{R}$ can be achieved and with relinearization in 2^n variables.

This results in an exponential runtime in the number n of variables for the Gaussian elimination.

We conclude that an exact algebraic attack can not be carried out against combining functions with a higher algebraic degree than 1. In the case of an algebraic degree of 1, a LP solver can be used to model the $F-$Combined Arbiter PUF as an alternative to machine learning algorithms. Nevertheless, an equation system can be constructed that approximates the sign function. But this approach has exponential runtime.

To strengthen the circuit containing multiple Arbiter PUFs, we looked at attacks that target the combining function of the structure. A correlation attack is feasible, if the combining function has correlation between one or multiple inputs and the output. An algebraic attack is only viable if F is a linear function. But in that case, the targeted $F-$Combined Arbiter PUF corresponds to the XOR Arbiter PUF and ML algorithms are a much better way to attack these structures. To harden the combination of multiple Arbiter PUFs we propose to use bent functions as combination functions. In the next chapter we will introduce and study such *Bent Arbiter PUFs.*

8

ARBITER PUFS AND BENT FUNCTIONS

In the last chapter we investigated attacks on the combining function of $F-$Combined Arbiter PUFs. It became apparent, that correlation attacks are feasible if the function F reveals a correlation between one ore more of its inputs and its output. Algebraic attacks on $F-$Combined Arbiter PUFs are not as successful as correlation attacks if the function F hat at least one AND operation due to the nature of the sign function. Approximations of the polynomial of F are still possible have an exponential representation in the number of used variables.

Let $P : \{-1,1\}^n \to \{-1,1\}$ be a $F-$Combined Arbiter PUF as in Definition 6.3. Let $F : \{-1,1\}^k \to \{-1,1\}$ be an arbitrary Boolean function that is the combining function for k Arbiter PUFs modeled by k LTFs $f_1, \ldots, f_k$ that map from $\{-1,1\}^n$ to $\{-1,1\}$. Recalling the desired security properties from Chapter 5 for the combining functions of nonlinear combination generators, we formulate the following required security properties for the combining function of $F-$Combined Arbiter PUFs.

- *Degree of the combining function*: The degree $d = \deg(F)$ of the combining function corresponds to the degree of its ANF representation (recall Definition 2.2) and can be computed in time $k \cdot 2^k$ using the fast Fourier transform [MS77, Chap. 13]. If $d = 1$ then F is a linear function and hence the $F-$Combined Arbiter PUF corresponds to the XOR Arbiter PUF. In that case known and fast ML algorithm attacks are applicable. If this property is fulfilled exact algebraic attacks can not be carried out against $F-$Combined Arbiter PUFs. Therefore, F should have a degree higher than 1.

- *Nonlinearity of the combining function*: To avoid the approximation of F via linear functions and hence algebraic and ML attacks, the combining function should have a high nonlinearity. This property can be indicated by the AI of F, the notion on nonlinearity, and the Walsh spectrum of F. While the computation of the AI is not straight forward, the Walsh spectrum can be determined in $k \cdot 2^k$ steps.

- *Distribution of the combining function*: To avoid statistical attacks, the combining function should also be unbiased. The distribution of F can be verified by the computation of the Walsh coefficient on 0, *i. e.* $\hat{\chi}_f(0)$, in 2^k steps.

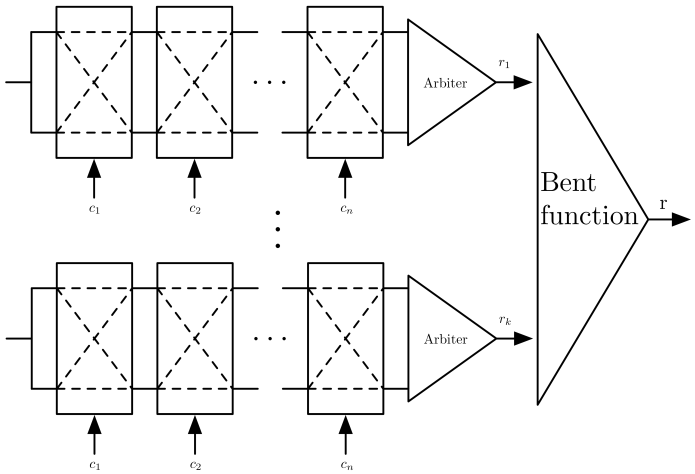


Figure 8.0.1: Structure of a Bent Arbiter PUF

- *Correlation immunity*: The combining function F should be correlation immune of order $k-1$ To prevent correlation attacks entirely. Recall, that unfortunately, as Fact 2.2 shows, a correlation immunity of order $k-1$ leads to a degree of $\deg(F) \leq 2$. Combined with the demand for a balanced output of f the degree is even $\deg(F) \leq 1$ and hence F a linear function. This opens up the opportunity for algebraic attacks. Therefore, one has to weigh the desired properties in this case, as a high order correlation immunity and a high algebraic degree can not be met at the same time. Recall from Lemma 2.7 that the correlation immunity can also be computed with the help of the Walsh spectrum of f in time $\mathcal{O}(2^k)$.

As not all desired properties can be fulfilled at the same time—recall the tradeoff between the correlation immunity and the algebraic degree in Fact 2.2—we propose the use of bent functions as combining functions. We call the resulting Arbiter PUF structure *Bent Arbiter PUFs*.

Definition 8.1. Let $k \in \mathbb{N}, k \geq 8$ be an even integer number and let $P_1, \ldots, P_k$ be k Arbiter PUFs in n stages each with their respective models $f_i : \{-1,1\}^n \to \{-1,1\}, 1 \leq i \leq k$. Let $F : \{-1,1\}^k \to \{-1,1\}$ be a bent function in k variables. The function F is called the *combining function* and we call such a composed Arbiter PUF scheme *Bent Arbiter PUF*. Figure 8.0.1 depicts the structure of such a $F-$Combined Arbiter PUF. Upon raising a challenge $c \in \{-1,1\}^n$ to the structure each Arbiter PUF evaluates c to $r_i = f_i(c)$. The final response $r \in \{-1,1\}$ is computed by processing $r = F(r_1, \ldots, r_k)$.

Note that at least 8 Arbiter PUFs should used in parallel for a Bent Arbiter PUF, to ensure an almost equal distribution of the output. Although bent functions are not perfectly balanced, their bias

approaches 0 very fast. Consider a Bent Arbiter PUF with $k = 8$ individual Arbiter PUFs and a bent function as a combining function. Then the bias of the output is

$$\frac{1}{2^{\frac{8}{2}+1}} = \frac{1}{2^5} = 0.03125\,,$$

assuming a uniformly distributed output of the individual Arbiter PUFs. We can safely assume the fact of uniformly distributed outputs due to results by Pelgrom et al. [PDW89].

In the following we discuss the feasibility of correlation and algebraic attacks against Bent Arbiter PUFs.

Correlation attack

Let $P : \{-1,1\}^n \to \{-1,1\}$ be a Bent Arbiter PUF with k individual Arbiter PUFs and a bent combining function $F : \{-1,1\}^k \to \{-1,1\}$. Since bent functions are not correlation immune, an attacker A can carry out the correlation attack described in Section 7.1 and she can learn P in time

$$\mathcal{O}\left(k \cdot \frac{n}{\varepsilon^2} \cdot \ln \frac{1}{\varepsilon\delta}\right) + k!\,,$$

where ε is the smallest error rate and δ is the confidence. However, since the correlation of each subset of the input variables to the output of the combination is equally small and around $\frac{1}{2}$, the best possible accuracy of each LTF is very low. Hence, the smallest desired error rate ε must be exponentially close to 0 for each LTF to obtain a reasonable result, yielding an exponential sample size $\mathcal{O}(n/\varepsilon^2 \cdot \ln 1/\varepsilon\delta)$ and therefore an exponential runtime in the number of samples. Additionally, the chance of learning the same Arbiter PUF increases, since there is no possibility to distinguish the individual learned LTFs because of the same correlation between all input subsets. Therefore, Bent Arbiter PUFs can not be successfully modeled exploiting a low correlation immunity of F.

Algebraic attack

Let again $P : \{-1,1\}^n \to \{-1,1\}$ be a Bent Arbiter PUF with k individual Arbiter PUFs and a bent combining function $F : \{-1,1\}^k \to \{-1,1\}$. Let an attacker A carry out the algebraic attack described in Section 7.2. We showed that an algebraic attack in the classical way is not possible due to the sign function inherent in the LTF representation of Arbiter PUFs. Additionally, in the case of Bent Arbiter PUFs, the modeling of the equations fails due to the fact that bent functions are not linear (in that case the Walsh spectrum of F would be equally distributed) and have therefore at least one AND operation. Hence, an algebraic attack can not be carried out against Bent Arbiter PUFs.

Bent functions seem to be a suitable alternative to the so far only used combination function XOR. Although bent functions do not have maximum correlation immunity and the highest possible algebraic degree for the number of used variables, their high nonlinearity and their equal correlation immunity over all subset of input variables makes them eligible combining functions. However, other attacks, especially machine learning attacks, may still be feasible against *Bent*

Arbiter PUFs. The study of this feasibility will be addressed in our future work.

9 CONCLUSIONS

In this thesis we have proposed the use of bent functions as combining functions for multiple Arbiter PUFs. The resulting *Bent Arbiter PUFs* are a promising new approach to increase the complexity of Arbiter PUF circuits and allow for the further use of this lightweight solution in a cryptographic environment. We have investigated attacks targeting the combining function–correlation and algebraic attacks–known from nonlinear combination generators. It has turned out that Bent Arbiter PUFs are safe against these type of attacks.

However, Bent Arbiter PUFs have to be examined further. The machine learning attacks of Rührmair et al. [Rüh+10; RS14] and Becker [Bec15] against XOR Arbiter PUFs should be examined regarding their suitability to learn Bent Arbiter PUFs. This has to be done for as many bent functions as possible, since it is unclear if some bent functions are better qualified as combining functions than others. This step would clarify if combining functions have any impact to the learning ability of Arbiter PUFs.

Future work on combined Arbiter PUFs should also extend the idea of Majzoobi et al. [MKP08] and investigate the impact of input transformation on the learning algorithms of Rührmair et al. and Becker.

At last, the ability to learn PUFs ignoring their internal structure should be considered. This technique would rely on the interpolation of multivariate polynomials over a finite field and would be similar to the attack of Jakobsen and Knudsen [JK97] on block ciphers.

Part IV

APPENDIX

BIBLIOGRAPHY

[Arm+11] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Francois-Xavier Standaert, and Christian Wachsmann. „A Formalization of the Security Features of Physical Functions." In: *SP '11: Proceedings of the 2011 IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2011. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5958042 (cit. on pp. 8, 9, 12).

[Arm+16] Frederik Armknecht, Daisuke Moriyama, Ahmad-Reza Sadeghi, and Moti Yung. „Towards a Unified Security Model for Physically Unclonable Functions." In: *Proceedings of the RSA Conference on Topics in Cryptology - CT-RSA 2016 - Volume 9610*. Columbia University. Cham: Springer-Verlag New York, Inc., Feb. 2016, pp. 271–287. DOI: 10.1007/978-3-319-29485-8_16. URL: http://link.springer.com/10.1007/978-3-319-29485-8_16 (cit. on pp. 8, 9, 12).

[Bec15] Georg T. Becker. „The Gap Between Promise and Reality - On the Insecurity of XOR Arbiter PUFs." In: *CHES* 9293.Chapter 27 (2015), pp. 535–555. DOI: 10.1007/978-3-662-48324-4_27. URL: http://link.springer.com/10.1007/978-3-662-48324-4_27 (cit. on pp. v, vi, 8, 11, 12, 57–59, 63, 73).

[BRS95] Richard Beigel, Nick Reingold, and Daniel A Spielman. „PP Is Closed under Intersection." English. In: *J. Comput. Syst. Sci.* 50.2 (1995), pp. 191–202. DOI: 10.1006/jcss.1995.1017. URL: http://linkinghub.elsevier.com/retrieve/pii/S0022000085710173 (cit. on p. 67).

[Blu+96] Avrim Blum, Alan M Frieze, Ravi Kannan, and Santosh Vempala. „A Polynomial-Time Algorithm for Learning Noisy Linear Threshold Functions." In: *FOCS* (1996). URL: http://ieeexplore.ieee.org/abstract/document/548492/ (cit. on pp. 12, 56).

[BK12] Jakramate Bootkrajang and Ata Kabán. „Label-Noise Robust Logistic Regression and Its Applications." English. In: *ECML/PKDD* 7523.Chapter 15 (2012), pp. 143–158. DOI: 10.1007/978-3-642-33460-3_15. URL: https://link.springer.com/chapter/10.1007/978-3-642-33460-3_15 (cit. on p. 64).

[BK13] Jakramate Bootkrajang and Ata Kabán. „Learning a Label-Noise Robust Logistic Regression: Analysis and Experiments.“ English. In: *Intelligent Data Engineering and Automated Learning – IDEAL 2013*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, Oct. 2013, pp. 569–576. DOI: 10.1007/978-3-642-41278-3_69. URL: https://link.springer.com/chapter/10.1007/978-3-642-41278-3_69 (cit. on pp. 12, 64).

[Byl94] T. Bylander. „Learning linear threshold functions.“ In: *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*. Oct. 1994, 1166–1171 vol.2. DOI: 10.1109/ICSMC.1994.400002. URL: http://ieeexplore.ieee.org/document/400002/ (cit. on pp. 56, 64).

[CM03] Nicolas Courtois and Willi Meier. „Algebraic Attacks on Stream Ciphers with Linear Feedback.“ In: *EUROCRYPT* 2656.Chapter 21 (2003), pp. 345–359. DOI: 10.1007/3-540-39200-9_21. URL: http://link.springer.com/10.1007/3-540-39200-9_21 (cit. on p. 18).

[CS09] Thomas W. Cusick and Pantelimon Stanica. *Cryptographic Boolean functions and applications*. New York, NY: Academic Press, 2009. URL: http://cds.cern.ch/record/1991075 (cit. on pp. 12, 23).

[DR99] J Daemen and V Rijmen. „The rijndael block cipher: Aes proposal.“ In: *First Candidate Conference (AES1)* (1999). URL: http://www.cs.technion.ac.il/~cs236612/00/slides/rijndael.ps.gz (cit. on p. 6).

[DV13] Jeroen Delvaux and Ingrid Verbauwhede. „Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise.“ In: *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. IEEE, 2013, pp. 137–142. DOI: 10.1109/HST.2013.6581579. URL: http://ieeexplore.ieee.org/document/6581579/ (cit. on p. 54).

[DV14] Jeroen Delvaux and Ingrid Verbauwhede. „Fault Injection Modeling Attacks on 65 nm Arbiter and RO Sum PUFs via Environmental Changes.“ In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 61.6 (2014), pp. 1701–1713. DOI: 10.1109/TCSI.2013.2290845. URL: http://ieeexplore.ieee.org/document/6728716/ (cit. on p. 10).

[Dev09] Srini Devadas. „Physical Unclonable Functions and Secure Processors.“ English. In: *Cryptographic Hardware and Embedded Systems - CHES 2009*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, 2009, pp. 65–65. DOI: 10.1007/978-3-642-04138-9_5. URL: https://link.springer.com/

chapter/10.1007/978-3-642-04138-9_5 (cit. on pp. 8, 9, 11, 57).

[DH76] Whitfield Diffie and Martin E Hellman. „New directions in cryptography." English. In: *IEEE Trans. Information Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638. URL: http://ieeexplore.ieee.org/document/1055638/ (cit. on pp. 5, 6).

[Dun01] R Duncan. „An overview of different authentication methods and protocols." In: *Report submitted to SANS Institute* (2001). URL: http://moreilly.com/CISSP/Dom3-1-an_overview_of_different_authent.pdf (cit. on p. 10).

[Fel50] W. Feller. *An introduction to probability theory and its applications: Volume I*. 1950. URL: http://ca.wiley.com/cda/product/0,,0471257087,00.html (cit. on pp. 59, 65).

[GTS16] Fatemeh Ganji, Shahin Tajik, and Jean-Pierre Seifert. „PAC learning of arbiter PUFs." English. In: *J. Cryptographic Engineering* 6.3 (2016), pp. 249–258. DOI: 10.1007/s13389-016-0119-4. URL: http://link.springer.com/10.1007/s13389-016-0119-4 (cit. on pp. 10, 55).

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Mathematical sciences. New York, NY: Freeman, 1979. URL: http://cds.cern.ch/record/210237 (cit. on p. 37).

[Gas+02] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. „Controlled physical random functions." In: *Eighteenth Annual Computer Security Applications Conference*. IEEE Comput. Soc, 2002, pp. 149–160. DOI: 10.1109/CSAC.2002.1176287. URL: http://ieeexplore.ieee.org/document/1176287/ (cit. on p. 11).

[Gas+04] Blaise Gassend, Daihyun Lim, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. „Identification and authentication of integrated circuits." English. In: *Concurrency - Practice and Experience* 16.11 (2004), pp. 1077–1098. DOI: 10.1002/cpe.805. URL: http://doi.wiley.com/10.1002/cpe.805 (cit. on pp. v, vi, 8–10, 12, 49, 50, 56, 57).

[Gol01] Oded Goldreich. *Foundations of cryptography*. Cambridge: Cambridge University Press, Cambridge, 2001. DOI: 10.1017/CBO9780511546891. URL: http://dx.doi.org/10.1017/CBO9780511546891 (cit. on p. 12).

[Gua+07] Jorge Guajardo, Sandeep S Kumar, Geert Jan Schrijen, and Pim Tuyls. „FPGA Intrinsic PUFs and Their Use for IP Protection." In: *CHES* 4727.Chapter 5 (2007), pp. 63–80. DOI: 10.1007/978-3-540-74735-2_5. URL: http:

//dx.doi.org/10.1007/978-3-540-74735-2_5 (cit. on p. 9).

[JK97] Thomas Jakobsen and Lars R Knudsen. „The Interpolation Attack on Block Ciphers.“ In: *FSE* 1267.Chapter 3 (1997), pp. 28–40. DOI: 10.1007/BFb0052332. URL: http://link.springer.com/10.1007/BFb0052332 (cit. on p. 73).

[Kar84] N. Karmarkar. *A new polynomial-time algorithm for linear programming*. New York, New York, USA: ACM, Dec. 1984. DOI: 10.1145/800057.808695. URL: http://portal.acm.org/citation.cfm?doid=800057.808695 (cit. on pp. 56, 67).

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography; 2nd ed.* Chapman & Hall/CRC cryptography and network security series. Hoboken, NJ: CRC Press, 2014. URL: http://cds.cern.ch/record/2018982 (cit. on p. 12).

[Ker83] Auguste Kerckhoffs. *La cryptographie militaire, ou, Des chiffres usités en temps de guerre: avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef*. Paris, 1883. URL: http://scholar.google.com/scholar?q=related:Jdbn3wtW-BEJ:scholar.google.com/&hl=en&num=20&as_sdt=0,5 (cit. on p. 5).

[KS99] Aviad Kipnis and Adi Shamir. „Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization.“ English. In: *CRYPTO* 1666.Chapter 2 (1999), pp. 19–30. DOI: 10.1007/3-540-48405-1_2. URL: http://dx.doi.org/10.1007/3-540-48405-1_2 (cit. on p. 37).

[Lee+04] J. W. Lee, Daihyun Lim, Blaise Gassend, G. E. Suh, Marten van Dijk, and Srinivas Devadas. „A technique to build a secret key in integrated circuits for identification and authentication applications.“ In: *2004 Symposium on VLSI Circuits. Digest of Technical Papers*. Widerkehr and Associates, 2004, pp. 176–179. DOI: 10.1109/VLSIC.2004.1346548. URL: http://ieeexplore.ieee.org/document/1346548/ (cit. on p. 12).

[Lim04] Daihyun Lim. „Extracting Secret Keys from Integrated Circuits.“ PhD thesis. May 2004 (cit. on p. 50).

[Lim+05] Daihyun Lim, J. W. Lee, Blaise Gassend, G. E. Suh, Marten van Dijk, and Srinivas Devadas. „Extracting secret keys from integrated circuits.“ In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.10 (2005), pp. 1200–1205. DOI: 10.1109/TVLSI.2005.859470. URL: http://ieeexplore.ieee.org/document/1561249/ (cit. on pp. 10, 12).

[MT94] Wolfgang Maass and György Turán. „How Fast Can a Threshold Gate Learn?“ In: *Proceedings of a Workshop on Computational Learning Theory and Natural Learning Systems (Vol. 1) : Constraints and Prospects: Constraints and Prospects*. Cambridge, MA, USA: MIT Press, 1994, pp. 381–414. URL: http://dl.acm.org/citation.cfm?id=192827.188543 (cit. on pp. 12, 56).

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. North-Holl Math. Libr. Amsterdam: North-Holland, 1977. URL: http://cds.cern.ch/record/109400 (cit. on pp. 17, 43, 69).

[MV10] R Maes and I Verbauwhede. „A discussion on the Properties of Physically Unclonable Functions.“ In: *TRUST 2010 Workshop* (2010). URL: https://www.esat.kuleuven.be/cosic/publications/talk-191.pdf (cit. on p. 9).

[MKP08] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. „Lightweight secure PUFs.“ In: *ICCAD* (2008). URL: http://dblp.org/rec/conf/iccad/MajzoobiKP08 (cit. on pp. 9, 11, 12, 51, 60, 73).

[MPC04] Willi Meier, Enes Pasalic, and Claude Carlet. „Algebraic Attacks and Decomposition of Boolean Functions.“ English. In: *EUROCRYPT* 3027.Chapter 28 (2004), pp. 474–491. DOI: 10.1007/978-3-540-24676-3_28. URL: https://link.springer.com/chapter/10.1007/978-3-540-24676-3_28 (cit. on p. 18).

[MS89] Willi Meier and Othmar Staffelbach. „Nonlinearity Criteria for Cryptographic Functions.“ English. In: *Advances in Cryptology — EUROCRYPT '89*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, Apr. 1989, pp. 549–562. DOI: 10.1007/3-540-46885-4_53. URL: https://link.springer.com/chapter/10.1007/3-540-46885-4_53 (cit. on pp. 12, 44).

[MVV96] Alfred J Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. „Handbook of Applied Cryptography.“ In: (1996). URL: https://books.google.de/books/about/Handbook_of_Applied_Cryptography.html?id=MhvcBQAAQBAJ (cit. on pp. 12, 31, 33).

[Mer+11] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. „Side-Channel Analysis of PUFs and Fuzzy Extractors.“ In: *TRUST* 6740.Chapter 3 (2011), pp. 33–47. DOI: 10.1007/978-3-642-21599-5_3. URL: http://link.springer.com/10.1007/978-3-642-21599-5_3 (cit. on p. 10).

[ODo14] Ryan O'Donnell. *Analysis of Boolean functions*. Cambridge: Cambridge University Press, New York, 2014. DOI: 10.1017/CBO9781139814782. URL: http://dx.doi.org/10.1017/CBO9781139814782 (cit. on pp. 12, 63).

[Ojh] P. C. Ojha. „On the Enumeration of Linear Threshold Functions." In: *IEEE Transactions on Neural Networks* (). URL: http://plouffe.fr/simon/OEIS/citations/JUCSLtfenum.pdf (cit. on p. 27).

[Pap+02] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. „Physical One-Way Functions." English. In: *Science* 297.5589 (Sept. 2002), pp. 2026–2030. DOI: 10.1126/science.1074376. URL: http://www.sciencemag.org/content/297/5589/2026.full (cit. on pp. 7, 9, 12).

[PDW89] M. J. M. Pelgrom, A. C. J. Duinmaijer, and A. P. G. Welbers. „Matching properties of MOS transistors." English. In: *IEEE Journal of Solid-State Circuits* 24.5 (Oct. 1989), pp. 1433–1439. DOI: 10.1109/JSSC.1989.572629. URL: http://ieeexplore.ieee.org/document/572629/ (cit. on pp. 51, 71).

[PM15] Rainer Plaga and Dominik Merli. „A new Definition and Classification of Physical Unclonable Functions." In: *arXiv.org* (Jan. 2015), pp. 7–12. DOI: 10.1145/2694805.2694807. arXiv: 1501.06363v1 [cs.CR]. URL: http://dl.acm.org/citation.cfm?doid=2694805.2694807 (cit. on p. 12).

[Pre82] Daryl Pregibon. „Resistant Fits for Some Commonly Used Logistic Models with Medical Applications." In: *Biometrics* 38.2 (June 1982), p. 485. DOI: 10.2307/2530463. URL: http://www.jstor.org/stable/2530463?origin=crossref (cit. on p. 64).

[Qua+16] Shahed E Quadir, Junlin Chen, Domenic Forte, Navid Asadizanjani, Sina Shahbazmohamadi, Lei Wang 0003, John A Chandy, and Mark Tehranipoor. „A Survey on Chip to System Reverse Engineering." English. In: *JETC* 13.1 (2016), pp. 1–34. DOI: 10.1145/2755563. URL: http://dl.acm.org/citation.cfm?doid=2917757.2755563 (cit. on pp. v, 7).

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. „A method for obtaining digital signatures and public-key cryptosystems." In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342. URL: http://dx.doi.org/10.1145/359340.359342 (cit. on p. 5).

[Ros57] Frank Rosenblatt. *The Perceptron - a perceiving and recognizing automaton*. English. Tech. rep. 85-460-1. Jan. 1957. DOI: 10.2514/8.12665. URL: http://arc.aiaa.org/doi/abs/10.2514/8.12665 (cit. on pp. 12, 56).

[Rot76] O. S. Rothaus. „On "bent" functions." English. In: *Journal of Combinatorial Theory, Series A* 20.3 (May 1976), pp. 300–305. DOI: 10.1016/0097-3165(76)90024-8. URL: http://linkinghub.elsevier.com/retrieve/pii/0097316576900248 (cit. on pp. 12, 25, 26, 44).

[RD13] Ulrich Rührmair and Marten van Dijk. „PUFs in Security Protocols: Attack Models and Security Evaluations." English. In: *2013 IEEE Symposium on Security and Privacy (SP) Conference*. IEEE, 2013, pp. 286–300. DOI: 10.1109/SP.2013.27. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6547116 (cit. on p. 12).

[RH14] Ulrich Rührmair and Daniel E. Holcomb. „PUFs at a glance." In: *Design Automation and Test in Europe*. New Jersey: IEEE Conference Publications, 2014, pp. 1–6. DOI: 10.7873/DATE.2014.360. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6800561 (cit. on p. 8).

[Rüh+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. *Modeling attacks on physical unclonable functions*. New York, New York, USA: ACM, Oct. 2010. DOI: 10.1145/1866307.1866335. URL: http://portal.acm.org/citation.cfm?doid=1866307.1866335 (cit. on pp. v, vi, 8, 12, 56, 58, 59, 63, 73).

[RSS09] Ulrich Rührmair, J. Sölter, and F. Sehnke. „On the Foundations of Physical Unclonable Functions." In: *IACR Cryptology ePrint Archive* (2009). URL: http://eprint.iacr.org/2009/277.pdf (cit. on pp. 8, 12).

[RS14] Ulrich Rührmair and Jan Sölter. „PUF modeling attacks - An introduction and overview." In: *DATE* (2014), pp. 1–6. DOI: 10.7873/DATE.2014.361. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6800562 (cit. on pp. 11, 73).

[Rüh+13] Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, and Srinivas Devadas. „PUF Modeling Attacks on Simulated and Silicon Data." In: *IEEE Trans. Information Forensics and Security* 8.11 (2013), pp. 1876–1891. DOI: 10.1109/TIFS.2013.2279798. URL: http://ieeexplore.ieee.org/document/6587277/ (cit. on pp. v, vi, 8, 11, 12, 56, 58, 59, 63).

[Sch90] Robert E Schapire. „The Strength of Weak Learnability.“ English. In: *Machine Learning* 5.2 (1990), pp. 197–227. DOI: 10.1007/BF00116037. URL: https://link.springer.com/article/10.1007/BF00116037 (cit. on pp. 10, 55).

[Sie84] Thomas Siegenthaler. „Correlation-immunity of nonlinear combining functions for cryptographic applications.“ English. In: *IEEE Trans. Information Theory* 30.5 (1984), pp. 776–780. DOI: 10.1109/TIT.1984.1056949. URL: http://dx.doi.org/10.1109/TIT.1984.1056949 (cit. on pp. 22, 23).

[Sko05] S. P. Skorobogatov. „Semi-invasive attacks: a new approach to hardware security analysis.“ In: (2005). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.228.2204&rep=rep1&type=pdf (cit. on pp. 7, 10).

[SD07] G. Edward Suh and Srinivas Devadas. „Physical Unclonable Functions for Device Authentication and Secret Key Generation.“ In: *DAC* (2007), pp. 9–14. DOI: 10.1145/1278480.1278484. URL: http://doi.acm.org/10.1145/1278480.1278484 (cit. on pp. v, vi, 8, 9, 12, 57).

[Taj+17] Shahin Tajik, Enrico Dietz, Sven Frohmann, Helmar Dittrich, Dmitry Nedospasov, Clemens Helfmeier, Jean-Pierre Seifert, Christian Boit, and Heinz-Wilhelm Hübers. „Photonic Side-Channel Analysis of Arbiter PUFs.“ English. In: *J. Cryptology* 30.2 (2017), pp. 550–571. DOI: 10.1007/s00145-016-9228-6. URL: http://link.springer.com/10.1007/s00145-016-9228-6 (cit. on pp. 10, 55).

[TW11] M. Tehranipoor and C. Wang. *Introduction to Hardware Security and Trust*. 2011. DOI: 10.1007/978-4419-8079-3. URL: https://books.google.com/books/about/Introduction_to_Hardware_Security_and_Tr.html?id=bNiw9448FeIC (cit. on p. 10).

[WT85] A. F. Webster and S. E. Tavares. „On the Design of S-Boxes.“ English. In: *Advances in Cryptology — CRYPTO '85 Proceedings*. Berlin, Heidelberg: Springer, Berlin, Heidelberg, Aug. 1985, pp. 523–534. DOI: 10.1007/3-540-39799-X_41. URL: https://link.springer.com/chapter/10.1007/3-540-39799-X_41 (cit. on p. 22).

[Win60] Robert O. Winder. „Single stage threshold logic.“ In: *SWCT* (1960), pp. 321–332. DOI: 10.1109/FOCS.1961.29. URL: http://ieeexplore.ieee.org/document/5397273/ (cit. on p. 27).

[XM88] Guo Zhen Xiao and James L Massey. „A spectral characterization of correlation-immune combining functions.“ In: *Institute of Electrical and Electronics Engineers. Transactions on Information Theory* 34.3 (1988), pp. 569–571. DOI: 10.1109/18.6037. URL: http://dx.doi.org/10.1109/18.6037 (cit. on p. 23).

[XB14] Xiaolin Xu and Wayne Burleson. „Hybrid side-channel/machine-learning attacks on PUFs - A new threat?“ In: *DATE* (2014), pp. 1–6. DOI: 10.7873/DATE.2014.362. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6800563 (cit. on p. 10).

[YI65] S Yajima and T Ibaraki. „A Lower Bound of the Number of Threshold Functions.“ In: *IEEE Transactions on Electronic Computers* EC-14.6 (1965), pp. 926–929. DOI: 10.1109/PGEC.1965.264090. URL: http://ieeexplore.ieee.org/document/4038615/ (cit. on p. 27).

DECLARATION

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben. Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, 11th May 2017

Tudor Alexis Andrei
Soroceanu

COLOPHON

This document was typeset using `classicthesis` style developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. It is available for LaTeX and LyX at

`https://bitbucket.org/amiede/classicthesis/`

Happy users of `classicthesis` usually send the author a real postcard; the collection of postcards received so far is featured at

`http://postcards.miede.de/`

Final Version as of May 11, 2017 (`BentArbiterPUF` version 1.0).