Physically Uncloneable Functions in the Stand-Alone and Universally Composable Framework

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor rerum naturalium (Dr. rer. nat.)

von

Dipl.-Inform. Heike Schröder

geboren in Hildesheim.

Referenten: Prof. Dr. Stefan Katzenbeisser
Prof. Dr. Marc Fischlin

Tag der Einreichung: 22. Mai 2013
Tag der mündlichen Prüfung: 03. Juli 2013

Darmstadt, 22.05.2013
Hochschulkennziffer: D 17

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

_____

Wissenschaftlicher Werdegang

Oktober 2001 - September 2007 Studium der Informatik an der Technische Universität Braunschweig mit Nebenfach Medizin.

seit April 2008 Wissenschaftliche Mitarbeiterin in der Security Engineering Group "Seceng" unter der Leitung von Prof. Stefan Katzenbeisser an der Technischen Universität Darmstadt.

September 2009 - Dezember 2009 Forschungsaufenthalt an der Stony Brook University, USA, unter der Leitung von Prof. Radu Sion.

Publications

Contributions to Books

U. Rührmair, H. Busch, and S. Katzenbeisser. "Strong PUFs: Models, Constructions, and Security Proofs" in A. Sadeghi, P. Tuyls (eds.), *Towards Hardware Intrinsic Security: Foundations and Practice*, Springer, 2010.

Peer Reviewed Conference und Workshop Publications

Physically Uncloneable Functions

C. Brzuska, M. Fischlin, H. Schröder, and S. Katzenbeisser. "Physically Uncloneable Functions in the Universal Composition Framework" *Advances in Cryptology - **CRYPTO 2011***. Lecture Notes in Computer Science, Volume 6841, pp. 51–70, Springer-Verlag, 2011.

S. Katzenbeisser, Ü. Koçabas, V. van der Leest, A.-R. Sadeghi, G. J. Schrijen, H. Schröder and C. Wachsmann. "Logical Reconfigurable Physically Unclonable Functions" *Cryptographic Hardware and Embedded Systems - **CHES 2011***. Lecture Notes in Computer Science, Volume 6917, pp. 374–389, Springer-Verlag, 2011.

H. Busch, M. Sotáková, S. Katzenbeisser, and R. Sion. "The PUF Promise" *Trust and Trustworthy Computing - **TRUST 2010***. Lecture Notes in Computer Science, Volume 6101, pp. 290–297, Springer-Verlag, 2010.

H. Busch, S. Katzenbeisser, and P. Baecher. "PUF-Based Authentication Protocols - Revisited" *Information Security Applications - **WISA 2009***. Lecture Notes in Computer Science, Volume 5932, pp. 296–308, Springer-Verlag, 2009.

Outsourcing and Cloud Computing

Dominique Schröder and Heike Schröder. "Verifiable Data Streaming" *ACM Conference on Computer and Communications Security - **CCS 2012***. pp. 953–964, ACM, 2012.

Signatures

C. Brzuska, H. Busch, Ö. Dagdelen, M. Fischlin, M. Franz, S. Katzenbeisser, M. Manulis, C. Onete, A. Peter, B. Poettering, and D. Schröder. "Redactable Signatures for Tree-Structured Data: Definitions and Constructions" *Applied Cryptography and Network Security* - **ACNS 2010**. Lecture Notes in Computer Science, Volume 6123, pp. 87–104, Springer-Verlag, 2010.

Secure Multi-Party Computation

M. Franz, B. Deiseroth, K. Hamacher, S. Jha. S. Katzenbeisser, and H. Schröder. "Secure Computations on Non-Integer Values with Applications to Privacy-Preserving Sequence Analysis" *Information Security Technical Report.* Volume 17, Issue 3, pp 117-128, 2013.

M. Franz, B. Deiseroth, K. Hamacher, S. Jha. S. Katzenbeisser, and H. Schröder. "Towards Secure Bioinformatics Services (Short Paper)" *Financial Cryptography and Data Security* - **FC 2011**. Lecture Notes in Computer Science, Volume 7035, pp. 276–283, Springer-Verlag, 2011.

B. Deiseroth, M. Franz, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. "Secure Computations on Real-Valued Signals" *IEEE Workshop on Information Forensics and Security* - **WIFS'10**. IEEE Press.

Acknowledgments

Diesen Abschnitt habe ich bis zuletzt vor mir hergeschoben, da er eine der schwersten Herausforderungen darstellte. Es ist schwer, all denen Menschen gerecht zu werden, die durch ihre fachliche und persönliche Unterstützung zur Fertigstellung dieser Dissertation beigetragen haben.

Zuerst möchte ich meinem Doktorvater, Stefan Katzenbeisser, danken. Durch ihn wurde mir erst die Möglichkeit gegeben, die Promotion zu beginnen. Während dieser Zeit habe ich viel von ihm gelernt— es war eine spannende und herausfordernde Zeit. Ganz besonders möchte ich mich jedoch bei Stefan dafür bedanken, dass er den Glauben an meine Promotion nicht verloren hat und mich bis zum Ende ermutigt und bestärkt hat, diese Arbeit fertig zu stellen. Ich danke dir sehr dafür.

Sehr mit Dank verbunden bin ich Marc Fischlin. Für seine Unterstützung an meiner PUF Forschung bin ich ihm besonders dankbar. Es war schwer an der theoretischen Sicht in diesem sehr praktischem Forschungsgebiet festzuhalten, dennoch haben mich die Diskussionen mit Marc bestärkt.

Frau Volkamer, Herrn Buchmann und Herrn Hollick möchte ich dafür danken, dass sie sich dazu bereit erklärt haben, meiner Kommission beizutreten.

Christina Brzuska und Bertram Poettering möchte ich an dieser Stelle von Herzen danken. Sie stand mir immer sowohl fachlich als auch persönlich zur Seite. Es hat riesig Spaß gemacht mit Christina zu arbeiten — ich kann mich an zahlreiche Diskussionen mit vielen Kannen Tee und bunten Beweis-Skizzen erinnern. Aber auch neben der Arbeit habe ich die Zeit mit Christina und Bertram genossen — ganz besonders sind mir die wunderbaren Sonntagsfrühstücke in Erinnerung geblieben.

Ein ganz großes Dankeschön möchte ich Jennifer Gerling aussprechen. Obwohl sie durch ihren Job und ihre Familie sehr eingespannt ist, hat sie dennoch die Zeit gefunden, diese Arbeit gegenzulesen. Ich stehe mit Respekt davor.

Auch möchte ich Ulrike und Barthel Schröder für ihre Unterstützung danken. Sie haben mir geholfen, über meine Promotion zu reflektieren und mich kritisch mit meiner Arbeit auseinanderzusetzen. Für den motivierenden Zuspruch habt vielen Dank!

Meinen Eltern, Marlies und Eberhard Busch, und meinem Bruder, Stefan Busch, möchte ich dafür danken, dass sie mich stets auf meinem Promotionsweg motiviert haben und mir stets Rückhalt geboten haben. Gerade am Ende hat mir ihr Zuspruch und ihre Zuversicht viel weitergeholfen — ich danke euch dafür von ganzem Herzen!

Der allergrößte Dank gebührt jedoch meinem Mann Dominique. Er hat mich bereits während des Studiums dazu motiviert, immer drei Schritte weiterzugehen, als ich eigentlich gegangen wäre und an meine Leistungen zu glauben. Somit würde ich ohne seinen endlosen Zuspruch und seine immerwährende Unterstützung nicht an diesem Punkt stehen. Es ist schön, dass es dich gibt — hab tausend Dank!

Heike Schröder

Abstract

In this thesis, we investigate the possibility of basing cryptographic primitives on *Physically Uncloneable Functions* (PUF). A PUF is a piece of hardware that can be seen as a source of randomness. When a PUF is evaluated on a physical stimulus, it answers with a noisy output. PUFs are unpredictable such that even if a chosen stimulus is given, it should be infeasible to predict the corresponding output without physically evaluating the PUF. Furthermore, PUFs are uncloneable, which means that even if all components of the system are known, it is computational infeasible to model their behavior. In the course of this dissertation, we discuss PUFs in the context of their implementation, their mathematical description, as well as their usage as a cryptographic primitive and in cryptographic protocols.

We first give an overview of the most prominent PUF constructions in order to derive subsequently an appropriate mathematical PUF model. It turns out that this is a non-trivial task, because it is not certain which common security properties are generally necessary and achievable due to the numerous PUF implementations.

Next, we consider PUFs in security applications. Due to the properties of PUFs, these hardware tokens are good to build authentication protocols that rely on challenge/response pairs. If the number of potential PUF-based challenge/response pairs is large enough, an adversary cannot measure all PUF responses. Therefore, the attacker will most likely not be able to answer the challenge of the issuing party even if he had physical access to the PUF for a short time. However, we show that some of the previously suggested protocols are not fully secure in the attacker model where the adversary has physical control of the PUF and the corresponding reader during a short time.

Finally, we analyze PUFs in the universally composable (UC) framework for the first time. Although hardware tokens have been considered before in the UC framework, designing PUF-based protocols is fundamentally different from other hardware token approaches. One reason is that the manufacturer of the PUF creates a physical object that outputs pseudorandom values, but where no specific code is running. In fact, the functional behavior of the PUF is unpredictable even for the PUF creator. Thus, only the party in possession of the PUF has full access to the secrets. After formalizing

PUFs in the UC framework, we derive efficient UC-secure protocols for basic tasks like oblivious transfer, commitments, and key exchange.

Zusammenfassung

In dieser Arbeit untersuchen wir die Möglichkeit, kryptographische Primitive auf *Physikalisch Unkopierbare Funktionen* (PUF) zu gründen. Ein PUF ist ein Stück Hardware, welches als Zufallsquelle angesehen werden kann. Wird ein PUF durch einen physikalischen Stimulus angeregt, so antwortet er mit einer verrauschten Ausgabe. PUFs sind nicht vorhersehbar, sodass es unmöglich sein sollte, ohne physische Auswertung des PUFs, die entsprechende Ausgabe vorherzusagen, selbst bei einem gewählten und vorgegebenen Stimulus. Außerdem sind PUFs nicht kopierbar, was soviel bedeutet, dass, selbst wenn alle Komponenten des Systems bekannt sind, es berechnungsmäßig unmöglich ist, ihr Verhalten zu modellieren. Im Zuge dieser Arbeit diskutieren wir PUFs im Rahmen ihrer Implementierung, ihrer mathematischen Beschreibung, sowie ihrer Verwendung als kryptographisches Primitiv und in kryptographischen Protokollen.

Zunächst geben wir einen Überblick über die prominentesten PUF Konstruktionen, um anschließend ein geeignetes mathematisches PUF Modell abzuleiten. Es stellt sich heraus, dass dies eine nicht-triviale Aufgabe ist, da nicht sicher ist, welche gemeinsamen Sicherheitseigenschaften aufgrund der zahlreichen PUF Implementierungen im Allgemeinen notwendig und erreichbar sind.

Als nächstes betrachten wir PUFs in Sicherheitsanwendungen. Aufgrund der PUF Eigenschaften eignen sich diese Hardwarestücke gut dazu, Authentifizierungsprotokolle basierend auf Challenge/Response-Paaren zu bauen. Ist die Anzahl der potenziellen PUF-basierten Challenge/Response-Paaren groß genug, so kann ein Angreifer nicht alle PUF Antworten messen. Somit wird ein Angreifer höchstwahrscheinlich nicht in der Lage sein, die Herausforderung der fragenden Partei zu beantworten, auch wenn er für eine kurze Zeit den physischen Zugriff auf den PUF hatte. Wir zeigen hingegen, dass bereits vorgeschlagenen Protokolle nicht vollständig in dem Angreifermodell sicher sind, in dem der Gegner genau für eine kurze Zeit physische Kontrolle über den PUF und das entsprechende Lesegerät hat.

Zum Abschluß analysieren wir zum ersten Mal PUFs im Universally Composable (UC) Modell. Obwohl Hardwarestücke zuvor in dem UC Modell betrachtet worden sind, unterscheidet sich das Designen von PUF-basierten Protokollen grundlegend von anderen Ansätzen, die auf Hardwarestück basieren. Ein Grund dafür ist, dass der Hersteller

des PUFs ein physisches Objekt herstellt, welches pseudozufällige Werte ausgibt, jedoch keinen spezifischen Code ausgeführt. Tatsächlich ist das funktionale Verhalten des PUF auch für den PUF Fabrikanten unvorhersehbar. Somit hat nur die Partei im Besitz des PUFs vollen Zugang zu den Geheimnissen. Nach der Formalisierung von PUFs im UC Modell leiten wir effiziente UC-sichere Protokolle für grundlegende Anwendungen wie Oblivious Transfer, Bit Commitment und Schlüssel-Austausch ab.

Contents

List of Abbreviations

AEGIS	Single-Chip Secure Processor
AKE	Authenticated Key Exchange
APUF	Arbiter Physically Uncloneable Function
ASIC	Application-Specific Integrated Circuit
ATM	Automated Teller Machine
BPUF	Butterfly Physically Uncloneable Function
COA	Certified of Authenticity
COPUF	Coating Physically Uncloneable Function
CPUF	Controlled Physically Uncloneable Function
CRP	Challenge/response Pair
FPGA	Field-Programmable Gate Array
IC	Integrated Circuit
IP	Intellectual Property
LR-PUF	Logically Reconfigurable Physically Unclone-able Function
MAC	Message Authentication Code
OT	Oblivious Transfer
PPUF	Public Physically Uncloneable Function
PPT	Probabilistic Polynomial Time
PRF	Pseudorandom Function
PUF	Physically Uncloneable Function
RAM	Random Access Machine

RF-DNA	Radio-Frequency Identification Tag
RFID	Radio-Frequency Identification
ROPUF	Ring Oscillator Physically Uncloneable Function
RPUF	Reconfigurable Physically Uncloneable Function
RSA	Cypher and Signature Algorithm of Rivest, Shamir and Adleman
SD	Statistical Distance
SIMPL	SImulation Possible but Laborious
SPUF	Silicon Physically Uncloneable Function
SRAM	Static Random Access Memory
TAP	PUF-based Authentication Protocol
UC	Universally Composable

1 Introduction

Communication technology has changed the world over the last decade. As a matter of course, we use our credit and debit card to withdraw cash from ATM's all over the world, make use of the digital passport for identification, or use RFID (*R*adio-*F*requency *ID*entification) tags in commerce. These examples of modern communication have two things in common: (1) The devices are computationally bounded and can only compute simple functions. (2) The devices are used in a highly networked environment. Consider, for example, the case where a customer wishes to withdraw money from an ATM using his debit card. The ATM checks the current balance of his bank account and only returns the money if the daily cash withdrawal limit has not been exceeded.

From a cryptographic point of view, both demands (computationally bounded devices that are secure in a highly networked environment) are quite contradictory: A system that interacts with many different protocols and primitives usually requires strong security guarantees. But strong security guarantees can usually only be achieved by relying on strong cryptographic primitives. These primitives, however, are computationally-intensive to be utilized. Thus, complex computations, such as all kind of public-key operations, are prohibitively expensive for these computationally weak devices.

Summing up, the above-mentioned demands bring up the question how adequate security requirements can be achieved by computational limited devices that are integrated into the network. If a device is computationally incapable of performing most of the cryptographic algorithms, how can we guarantee secure and authenticated communication channels for it?

This dissertation embarks on a new path towards answering these questions by relying on *physical assumptions* instead of computational assumptions. In fact, instead of assuming that some number-theoretic assumptions, such as that the RSA assumption, hold, we consider the existence of *Physically Uncloneable Functions* (PUFs). A PUF is a simple hardware token and we use it to build cryptographic primitives and protocols.

1.1 Physically Uncloneable Functions

Physically Uncloneable Functions have been introduced by Pappu in [Pappu, 2001]. He defined a PUF as a physical object which can be seen as a source of randomness. These devices have the fascinating property that they are hard to clone—they get this property through an uncontrollable manufacturing process. The PUF can be evaluated by a physical stimulus (aka. challenge), on which it provides a noisy output (aka. response).

An illustrative example of a PUF is a so-called *optical* PUF. An optical PUF consists of a transparent material, to which many light scattering particles are randomly added during its production. When a laser beam shines on the PUF, a random speckle pattern will arise. The position and angle of the laser represent the challenge, while the speckle pattern is recorded, quantized, and encoded to form the PUF's response.

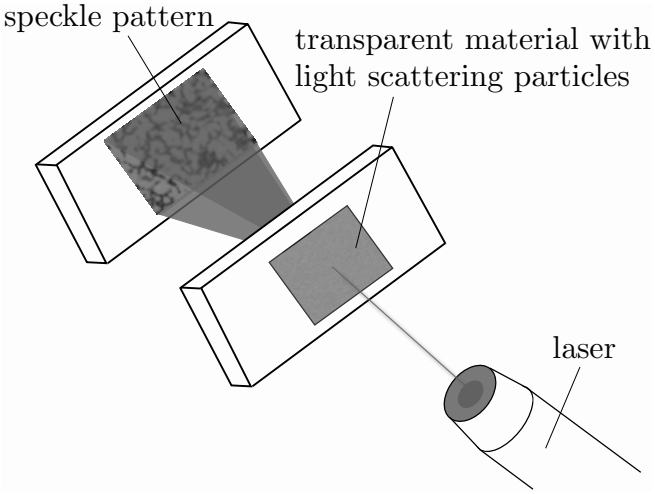


Figure 1.1: Configuration of an optical Physically Uncloneable Function. The position and angle of the laser represent the challenge and the speckle pattern forms the response. The speckle pattern is taken from [Pappu, 2001].

Following Pappu's seminal work, researchers have investigated the implementation of PUFs exploiting physical characteristics other than the optical approach; e.g., *silicon* PUFs, which use timing and delay characteristics of silicon circuits or *coating* PUFs, which measure the capacitance of a coating layer that masks an integrated circuit. These and several different implementations are discussed comprehensively later in this thesis.

1.1.1 Physically Uncloneable Functions vs. Hardware Tokens

It is well known that strong security demands can be efficiently achieved by making several types of setup assumptions such as, e.g., a common reference string (CRS). In the CRS model, a trusted party computes a string according to some distribution and passes this string to the parties that run an execution of a protocol. The CRS guarantees the participants that it was generated honestly in a restricted manner. If the string was computed in a malicious manner, then all private data in the network could potentially be revealed.

However, setup assumptions are often unrealistic or rather expensive to realize in practice. A promising approach to obtain cryptographic protocols which are simultaneously highly efficient and achieve strong security requirements without relying on (trusted) setup assumptions, is to use hardware components like signature cards [Hofheinz et al., 2005], one-time programs [Goldwasser et al., 2008], standard smart cards [Hazay and Lindell, 2008], or even more complex tokens [Katz, 2007]. However, most of these hardware-tokens implement additional cryptographic operations which are based on number-theoretical assumptions or computational assumptions, such as PRFs or MACs. Protocols designed with these tokens provide security under the presumption that certain computational assumptions holds. In contrast, PUFs are a hardware primitive which neither is based on computational nor on number-theoretic assumptions, but which relies only on a physical assumption.

1.1.2 Physically Uncloneable Functions vs. Mathematical Functions

Although PUFs include the name *function*, they do not really fulfill the properties of a function in a mathematically sense. In fact, their output behavior is noisy, such that querying the PUF twice on the same challenge may yield different responses. This phenomena results from the fact that the PUF's output depends heavily on the physical characteristics of the hardware the PUF is implemented on. By nature, the hardware characteristics are sensitive to environment variations such as temperature or supply voltage. Thus, the PUF will always respond with a slightly different output on the same input.

1.2 Security Models

An important approach in modern cryptography is provable security. Thereby, one defines a mathematical model which represents the security properties needed in the real world and proves subsequently that schemes are secure in this model. In the following, we distinguish between two basic security models: the *stand-alone model* and the *universally composable framework* (UC framework).

The stand-alone model analyzes protocol executions that run in isolation without considering other protocols that may run simultaneously. However, in the previously mentioned examples of modern communication (credit/debit card and RFID), the protocol should remain secure even when (honest) parties are running many other protocols concurrently. Protocols that are secure in such a model are called *universally composable*.

PUFs in the Stand-Alone Model In the stand-alone model, a single set of parties executes a single protocol in isolation. This protocol is usually based on a hard problem. The security then follows by contradiction versus a game-based approach: The game takes place between a challenger and an adversary. The challenger not only creates an instantiation of the protocol (whose security is based on some hard problem) but also answers all the queries which are requested (and allowed) from the adversary. At some point, the adversary stops, outputting a putative solution. The proof of security then derives a contradiction by showing that any successful adversary can be used to break the underlying hard problem. In other words, since the underlying problem is believed to be hard, such an adversary cannot exist and the security of the cryptosystem follows. In the context of PUFs, one of these security properties is unpredictability. Loosely speaking, this means that the challenger gives the adversary access to a PUF, which it can query arbitrarily. At the end of the game, the adversary has to predict a valid challenge/response pair for exactly this PUF which was never asked before.

PUFs in the UC Framework Security games in the stand-alone model have the advantage that they are often easy to understand and easy to work with. The drawback, however, is that the security is no longer guaranteed when the protocol may be run together with many other protocols in unpredictable ways, which occurs thoroughly

in the real world. To guarantee security in a setting where a secure protocol runs concurrently with arbitrary other protocols, Canetti introduced the UC framework in [Canetti, 2001]. It turns out that the UC framework allows us to model the properties of a PUF such as uncloneability and tamper-resistance in a more natural way.

The basic idea of the UC framework is to specify how the protocol should ideally behave—this is called the *ideal functionality*. Subsequently, the framework compares the real protocol execution with the ideal functionality in order to prove that a certain protocol implementation running in a given environment securely realizes the ideal functionality. Unfortunately, shortly after the introduction of the UC framework, it was shown that a number of interesting applications, such as adaptively secure commitments, cannot be realized in the "plain" UC framework by two-party protocols [Canetti and Fischlin, 2001, Canetti et al., 2006, Lindell, 2009]. Hence, follow-up works explored various modifications of the UC framework in order to guarantee secure computations, e.g., the assumption of a common reference string (CRS) [Canetti, 2001, Canetti and Fischlin, 2001, Canetti et al., 2002]. While these setup assumptions allow us to realize many interesting tasks, they still can be seen as some kind of trusted setup assumption. In order to avoid any kind of trusted setup assumption, Katz [Katz, 2007] suggested to use hardware tokens with the goal to rely on *physical* assumptions rather than on an assumption of trust in some external entity. If UC protocols will ever be used in practice, the underlying setup assumptions will play an important role.

The advantage of using the UC framework in the context of PUFs is that it supports an easy modeling of tamper-proof hardware tokens via ideal functionalities. Roughly, the ideal functionality captures the abstract security properties of the token, and one considers a hybrid world in which real-world protocols and parties also have access to this ideal functionality (and thus the token).

1.3 Contribution

Modeling PUFs appropriately is a highly non-trivial task. The major difficulty is that there are different types of PUFs with different (physical) properties and that there does not seem to be a general agreement upon the common security properties of PUFs even for a single type only (e.g., whether a PUF is one-way or not, or if the output is

pseudo-random). After reviewing implementations and applications of PUFs in Chapter 2, we subsequently propose new game-based definitions and relate them to concurrent work in Chapter 3.

Contribution 1.3.1. *How can we cryptographically formalize Physically Uncloneable Functions?*

Contribution 1.3.1 includes the results of my publications [Busch et al., 2010] published at TRUST'10 and [Brzuska et al., 2011a] published at CRYPTO'11.

The properties of PUFs inspired researchers to build authentication protocols that rely on challenge/response pairs. In these protocols, one party issues a previously measured challenge which the other party has to answer by measuring the according PUF. If this PUF response matches a pre-recorded response held at the issuing party, the challenging party is authenticated. If the number of potential challenge/response pairs of a PUF is large and an adversary cannot measure all PUF responses, he will most likely not be able to answer the challenge of the issuing party even if he had physical access to the PUF for a short time. However, Chapter 4 deals with the security of current PUF-based authentication protocols. We show that some of them are not fully secure in the above-mentioned attacker model when the attacker has physical control of the PUF and the corresponding reader during a short time.

Contribution 1.3.2. *Are current PUF-based authentication protocols fully secure when an adversary has physical control of the PUF and the corresponding reader during a short time?*

Contribution 1.3.2 includes the results of my publication [Busch et al., 2009], which was published at WISA'09.

The above-mentioned protocols are *only* secure in the stand-alone model. However, schemes with strong security guarantees in the UC framework are a desirable goal. In Chapter 5, we consider PUFs in the UC framework.

Contribution 1.3.3. *How can we formalize PUFs in the UC framework?*

Using our proposed UC notion of PUFs, we then derive efficient UC-secure protocols for basic tasks like oblivious transfer, commitments, and key exchange. It turns out

that designing PUF-based protocols is fundamentally different from the designing other hardware tokens. For one part this is because the functional behavior is unpredictable even for the creator of the PUF. This causes an asymmetric situation in which only the party in possession of the PUF has full access to the secrets.

Contribution 1.3.4. *Can we realize the following protocols that are based on our proposed UC notation of PUFs: Oblivious Transfer, commitment scheme, and key exchange?*

Contribution 1.3.3 and 1.3.4 include the results of my publication [Brzuska et al., 2011a], which was published at CRYPTO'11.

1.4 Follow-Up Works

Several follow-up papers have been published recently continuing the line of research that was introduced by the publications contained in this thesis. Some of the serious follow-up research is presented below.

1.4.1 Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions

Based on our results, [Ostrovsky et al., 2012] extend our model by malicious PUFs (recall that we assume that all PUFs are trusted). Ostrovsky et al. hold the view that our assumption is unrealistic since one could create a hardware token which looks like a PUF but implements an arbitrary function. Thus, the authors broaden our PUF definition so that an adversary can create and use untrusted PUFs for which the common PUF security properties are not ensured.

1.4.2 Physical Unclonable Functions in Cryptographic Protocols: Security Proofs and Impossibility Results.

Rührmair and van Dijk propose another PUF definition which is weaker one than ours in [van Dijk and Rührmair, 2012]. In our definition, the min-entropy condition is required

to hold for every challenge, while the definition of Rührmair and van Dijk only holds for randomly chosen challenges. The authors also build PUF-based oblivious transfer, bit commitment, and key exchange protocols, however, their constructions are not secure in the UC framework. Furthermore, Rührmair and van Dijk analyze our OT protocol and mention that if an adversary $\mathcal{A}$ (on the receiver's side) is allowed to evaluate the PUF on $2^{n/2}$ queries (n is the length of the query), then $\mathcal{A}$ gets the two sender's strings. Consequently, Rührmair and van Dijk obtain a quadratic speed-up over the brute force attack. This attack is applicable if the challenge space of the PUF family is small, such that an adversary is able to evaluate the PUF on $2^{n/2}$ queries. However, in our work we assume that the PUF has large input spaces and, thus, this attack is not feasible. Nevertheless, the attack highlights the interesting gap between foundational works and their concrete realization in the real world.

2 Constructions and Applications of Physically Uncloneable Functions

In 2001, Pappu proposed the idea of a Physically Uncloneable Function: A PUF is a function that is embedded inseparably into a physical system [Pappu, 2001]. Roughly speaking, the function can be seen as a source of randomness where the randomness arises from uncontrollable manufacturing variations during the fabrication process. For PUF evaluation, the physical system is queried with a stimulus (called challenge) which results in a physical output (called response). The output of a PUF is the result of a very complex interaction between the stimulus and the (random) components of the physical system. Even if all components are known, it should still be computational infeasible to model this interaction and its behavior. This property is called uncloneability. Moreover, if a chosen stimulus is given, it is infeasible to predict the corresponding output without physically evaluating the PUF. This property is called unpredictability. Uncloneability and unpredictability are the main security properties of PUFs, but they can also have uncorrelated output or be one-way and tamper-resistant. In the following we survey existing PUF implementations, including optical PUFs, silicon PUF,s, arbiter PUFs, ring oscillator PUFs, reconfigurable PUFs, SRAM PUFs, butterfly PUFs, coating PUFs, and controlled PUFs. Furthermore, we review typical PUF applications, such as identification, authentication, key extraction, certificates of authenticity, as well as PUF as cryptographic primitives.

2.1 Constructions of Physically Uncloneable Functions

In this section, we review existing PUF implementations and discuss their advantages and disadvantages as well as their security. Pappu suggested as a first PUF imple-

mentation a construction based on an optical medium. Based on Pappu's approach, researchers tried to enhance his idea by applying it to other physical media, such as silicon or coating technology.

2.1.1 Optical PUFs

As a first way of implementing Physically Uncloneable Functions, Pappu proposed an optical approach [Pappu, 2001]. An optical PUF consists of a transparent material, to which many light scattering particles are added randomly during production. Such a device causes a random speckle pattern when shining a laser beam onto it; here, the position and angle of the laser (and possibly other parameters such as amplitude and wave-length) represent the challenge, while the speckle pattern is recorded, quantized and encoded to form the PUF response (see Figure 1.1).

Naturally, a PUF should support a large number of challenge/response pairs (CRPs), in order to make it unfeasible to learn responses to challenges that were not yet issued. It is possible to estimate the entropy of an optical PUF ($\geq 4 \cdot 10^6$ per 5cm^2) and the information contained in one CRP [Tuyls et al., 2005]. Based on this data, the authors calculate the corresponding number of independent CRPs ($\geq 3 \cdot 10^4$ per 5cm^2), which turns out to be much lower than the number of all possible (not necessarily independent) challenges ($\sim 10^{10}$). As the number of independent CRPs is rather low and they can all be pre-recorded by an attacker who has unlimited physical access to the PUF once, optical PUFs do not offer security in the information-theoretic sense. However, in [Tuyls et al., 2005], the authors claim that interpolation of the PUF's behavior is computationally costly and, therefore, a lot more challenges need to be measured in order for the attacker to successfully predict the response to a fresh challenge. To prevent the attacker from exhaustively reading out all the CRPs (meaning, not only the independent ones), a method for decreasing the measurement rate is proposed. For instance, if 10ms are required to measure one challenge, the attacker can measure about $\frac{1}{100}$ of all challenges ($\sim 10^8$) in a week of uninterrupted access to the optical PUF [Tuyls et al., 2005]. As a drawback, developing a reliable measurement apparatus for optical PUFs is a complex problem, which requires costly high-precision mechanics and thus limits their usage.

2.1.2 Silicon PUFs, Arbiter PUFs, and Ring Oscillator PUFs

Gassend et al. [Gassend, 2003] proposed a PUF that uses silicon technology. Based on the approach of [Thompson, 1996], which shows that uncontrollable process variations during chip production make chips measurably different, *Silicon Physically Uncloneable Functions* (SPUF, also known as Silicon Physical Random Functions) exploit inherent variations in integrated circuits [Gassend et al., 2002b, Gassend et al., 2004]. Even for chips that were produced with identical layout masks, an SPUF supports still an unpredictable behavior. The SPUF implementation in [Gassend, 2003] is based on the observation that the timing behavior of chips differs. The silicon PUF consists of a number of switch delay elements, which are connected in series. Every element has two inputs and uses a two-to-one multiplexer[1] to swap its inputs depending on one challenge bit: If the challenge bit is 0, both signals go straight trough the element. Otherwise, the top and bottom signals are switched. To operate the PUF for a specific challenge, a rising signal is input into a sequence of n delay elements (see Figure 2.1). At each element, the signal is either swapped or simply passed on, depending on one challenge bit. The PUF response is the time required for the signal to travel through all n elements. Since each delay element doubles the number of paths the signals can possibly take, a SPUF with n elements can produce 2^n delay paths. An important advantage of silicon PUFs is that their production does not require any special devices on top of classic chip manufacturing equipment.

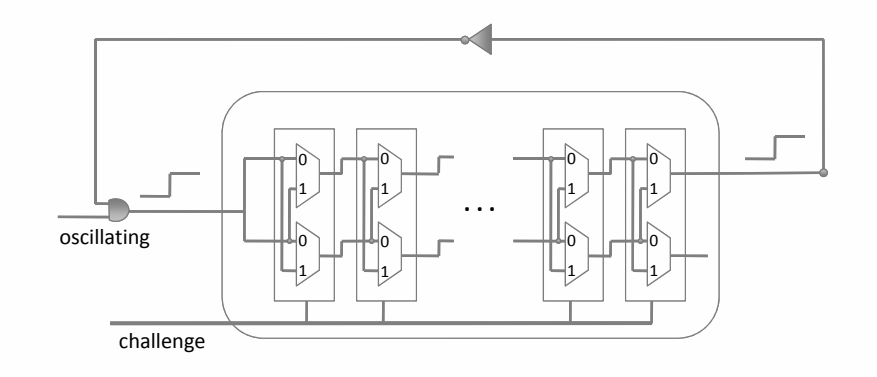


Figure 2.1: Silicon PUF [Gassend, 2003].

[1] A multiplexer is a device that selects one of many analog or digital input signals and forwards the selected input into a single line.

Environmentally induced noise plays a crucial role in the context of silicon PUFs because circuit delays are, for example, sensitive to temperature. This sensitivity carries over to the PUF. Lim et al. [Lim, 2004, Lim et al., 2005, Lee et al., 2004, Gassend et al., 2003] enhance the reliability of silicon PUFs against environmental noise by introducing *Arbiter Physically Uncloneable Functions* (APUF). Instead of measuring the absolute delay of a single signal, they use the relative timing behavior of two signals. To compute the output for a specific challenge, a rising signal is given to the two inputs at the same time. Both signals race through the device; at the end, an arbiter circuit determines which signal passed the device faster. Thus, the challenge of the PUF still determines the path that both signals take through the device, while the response will now be a *single* bit $r \in \{0, 1\}$. Again, due to n delay elements with two possible paths, there exists 2^n delay paths. To obtain an m-bit response, one can either duplicate the circuit m times or evaluate the device consecutively on m different challenges and paste together the results.

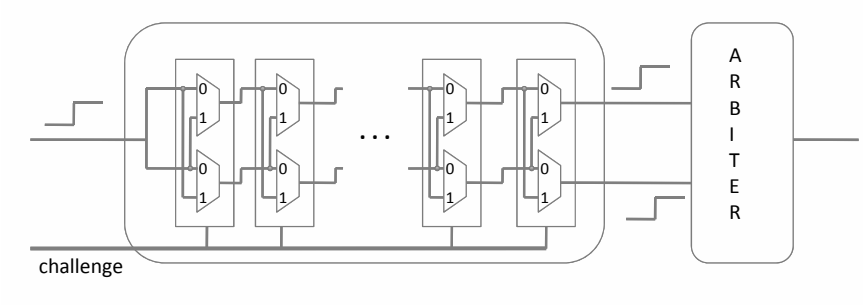


Figure 2.2: Arbiter PUF [Lim, 2004].

Concerning the security of PUFs, it was shown that the response of an integrated circuit[2] (IC) based PUF circuit can be represented as a linear function of a challenge [Lim, 2004]. If an attacker knows all delays of each element of a path through the circuit, it can derive (predict) a response for a given challenge by calculating the sum of the delays of each element. Since measuring the delays at each element is a hard problem, an attacker can use machine learning techniques to build a software circuit that models the PUF circuit. With this model, the attacker can simulate the PUF and can predict a response for a random challenge; e.g., in [Lim, 2004], the authors achieved a prediction error rate of $3,55\%$ after using 5000 training CRPs on an ASIC implementation by means of a

[2] An integrated circuit is an electronic circuit on a semiconductor material (called chip) which consists of several transistors and other electronic components.

Support Vector Machine. Note, that using the linear delay model implies that the PUF response is statistical close to the ideal distribution. In reality, however, this is not the case due to measurement or environmental variations. Nevertheless, Lim generalized this model to a probabilistic one to model all the environmental variations. The author also suggests methods to modify the arbiter PUF such that the above mentioned model is no longer possible [Lim, 2004, Rührmair et al., 2010].

All PUF implementations that are based on delay characteristics in ICs are not protected against environmentally induced noise. Consequently, a PUF produces different measurements for the same stimulus. Furthermore, if the variations of the PUF measurements are too high and the measurement variations are not adequately improvable, a PUF may not be uniquely identified. Lim et al. handle the problem of environmentally induced noise by analyzing the coherence between environmental variations and circuit delays such as temperature and power supply [Lim, 2004, Lim et al., 2005]. Firstly, the authors measured an *inter-chip variation*, which states how many bits (in %) of two responses measured by two different PUFs for the same challenge are different. The average inter-chip variation of a PUF should be ideally close to 50%, since an ideal PUF response consists of uniformly distributed independent random bits. Subsequently, the authors analyze the *environmental variation*, which states how many bits of PUF responses will change if they are measured from the same PUF (the noise of the PUF response). The average environmental variation of a PUF should be ideally close to 0%, since a PUF should produce the same bits reliably using the same PUF input. For an arbiter PUFs, the authors obtained the average inter-chip variation of 23% and an environmental variation of $\approx 4,82\%$, if the temperature increases more than $40°C$ from $27°C$ and $\approx 3,74\%$, if the voltage variation increases by $\pm 2\%$, respectively. This shows that an arbiter-based PUF reduces the environmental variations well enough below the average inter-chip variation of 23% [Katzenbeisser et al., 2012].

Adapted from arbiter PUFs, Suh and Devadas look for a higher reliability and an easier way to implement PUFs on Application-Specific Integrated Circuits (ASIC) and Field-Programmable Gate Arrays (FPGA) [Suh and Devadas, 2007]. For this purpose, the authors introduce so-called *Ring Oscillator Physically Uncloneable Functions* (ROPUF)[3]. These PUFs are based on delay loops, which are commonly used to gen-

[3]Note that a first very basic Ring Oscillator PUF was already proposed in [Gassend et al., 2002b]. The ROPUF is based on an arbiter PUF construction whereat the delay circuit is placed in a self-oscillating circuit. A frequency counter receives the oscillating signal and counts the number

erate random bit strings. A delay loop, or ring oscillator, is a simple circuit that oscillates with a frequency influenced by manufacturing variations and thus cannot be predicted. However, the frequency can easily be determined by a counter. The PUF construction uses n such circuits and compares the frequency of two selected ones: depending on which oscillator is faster, an output of 1 or 0 is produced. To produce an output of several bits, one picks randomly a set of such oscillators according to the challenge; comparing each pair produces one output bit. Analyzing this approach, one can generate $\frac{n(n-1)}{2}$ pairs out of n oscillators. However, Suh and Devadas state in [Suh and Devadas, 2007] that in reality the number of independent bits that the circuit can generate (the entropy of the circuit) is less than $\frac{n(n-1)}{2}$. This is due to the fact that independent pair-wise comparisons are difficult to obtain since they are correlated. The authors gave the following example: If oscillator A is faster than oscillator B (which outputs 1) and if oscillator B is faster than C (which also outputs 1), then the comparison between oscillator A and C will yield that oscillator A will also be faster than oscillator C (which again outputs 1) and thus, these bits are correlated. Subsequently, Suh and Devadas mentioned that the number of independent comparisons that can be generated by the circuit is restricted by the number of possible orderings of the oscillator's frequencies. There exists $n!$ possible orderings of ring oscillators if the frequencies are independent and identically distributed. Each of the possible $n!$ orderings is equally likely and thus we have $\Theta(n \log n)$ bits of entropy. Furthermore, the authors apply a technique, called 1-out-of-k masking scheme, to enhance the reliability of ring oscillator frequencies (they are sensitive to environment conditions). In doing so, they analyze k oscillators, choose the pair with the maximum difference in frequency, and take the result of this comparison as output. Experimental results show that using 15 FPGAs with $k = 8$ generates PUF outputs with an average inter-chip variation of $46,15\%$ and a intra-chip variation of 0.48% even under temperatures like $120°C$ and voltage like +10%. Later on, Suh et al. used their proposed ROPUF in the development of the so-called AEGIS processor [Suh et al., 2005b, Suh et al., 2007, Suh et al., 2005a], which can resist both *software* and *physical* attacks. In particular, they use the PUF to store secrets in a secure, uncloneable, and cost-effective way.

of oscillating loops for a predefined interval. However, since the ROPUF of Gassend et al. is also based on delay circuits like APUFs, their construction is not secure against modeling attacks.

Arbiter PUFs vs. Ring Oscillator PUFs Although ring oscillator PUFs are more reliable and easier to implement on ASICs and FPGAs than arbiter PUFs, arbiter PUFs are faster, smaller, and less power-consuming than ROPUFs. Thus, arbiter PUFs are better suitable for resource constrained platforms such as RFIDs, in which context they are also commercially available [Verayo, 2010, ID, 2011].

2.1.3 Reconfigurable PUFs

[Kursawe et al., 2009] picked up the idea of Pappu and suggested a new variant of optical PUFs, called *Reconfigurable Physically Uncloneable Functions* (RPUF). The idea of a reconfigurable PUF is to equip the regular PUF with a mechanism that transforms it into a new PUF with a different and unpredictable challenge/response behavior. The mechanism of reconfigurable PUFs allows for using a PUF for security purposes, even if its original challenge/response behavior has been learned. In this case, it is essential to protect the PUF update process, as otherwise an adversary may be able to reconfigure the PUF and cause a denial of service. A first implementation realized the reconfiguration step by heating the PUF with a laser so that the light-scattering particles slowly changed position. A second implementation realized the reconfiguration step by changing the phase memory. For a detailed discussion, see [Kursawe et al., 2009]. As a new approach in this context, Katzenbeisser et al. introduced the concept of a logically reconfigurable PUF (LR-PUF) [Katzenbeisser et al., 2011]. Using an LR-PUF does not change the challenge/response behavior of a PUF physically but logically. An LR-PUF is based on a regular PUF and a control logic circuit. The control logic consists of a state, that is stored in the non-volatile memory, a querying, and a reconfiguration algorithm. A reconfiguration of the LR-PUF means to update the state due to an appropriate state-update mechanism, which results in a completely different challenge/response behavior of the LR-PUF. The authors present several input/output transformation and state update mechanisms that come up with new security objectives. For a detailed discussion, see [Katzenbeisser et al., 2011].

2.1.4 SRAM PUFs and Butterfly PUFs

Guajardo et al. [Guajardo et al., 2007, Guajardo et al., 2008] suggested a PUF which relies on *SRAM* (Static Random Access Memory). These PUFs consist of a number

of memory cells, involving two *cross-coupled*[4] inverters, and have two stable states, commonly denoted by 0 and 1. After power-up, cells will randomly end up in state 0 or 1; the state that a specific memory cell will reach is mainly dependent on the production process. A challenge is represented by a subset of the memory cells to be read-out after power-up; the response is their respective power-up state. Currently available ICs can incorporate $\sim 10^6$ to 10^7 SRAM cells. Yet, without any additional mechanism for decreasing the read-out rate, SRAM PUFs are vulnerable to an exhaustive read-out attack.

Since not all FPGAs support uninitialized SRAM memory, [Kumar et al., 2008] enhanced the concept of SRAM-based PUFs to *Butterfly Physically Uncloneable Functions* (BPUF). These PUFs provide a new way of exploiting circuit delays. Butterfly PUFs use unstable cross-coupled circuits, just like SRAM PUFs. While SRAM cells are based on cross-coupled inverters, in butterfly cells inverters are replaced by latches or flip-flops. Latches are circuits which store information and can be cleared (turns output to 0) or preset (turns output to 1). Like SRAM cells, butterfly cells have only two stable states. To read out the PUF, one of the latches is cleared and simultaneously the other one is preset. This brings the BPUF into an unstable condition. The butterfly cell falls back into one of the stable states, depending on the circuit delays, which are were determined by the manufacturing process.

SRAM PUFs vs. Butterfly PUFs There is no big difference between SRAM PUFs and BPUFs and they are very similar in their construction and operation. Unlike SRAM PUFs, BPUFs are suitable for all types of FPGAs. Note that FPGAs of course can contain SRAM cells. However, these SRAM cells cannot be used as a PUF because their power-up state is usually not accessible [Maes et al., 2008].

2.1.5 Coating PUFs

Another approach to obtain a PUF is based on an *active coating*, a covering that is applied to the surface of an object. Posch [Posch, 1998] suggests to protect a device by embedding a unique signature into the coating material used in smart cards. Tuyls et al. apply this idea to PUFs and introduce the concept of *Coating Physically Uncloneable*

[4]The output of the first inverter is connected to the input of the second one, and vice versa.

Functions (COPUF) [Tuyls et al., 2006, Škorić et al., 2006a]. A coating PUF employs a protective coating, covering an integrated circuit. The opaque coating material is doped with dielectric particles, have random properties concerning their size, shape, and location. Below the coating layer, a comb structure of metal wire sensors is used to measure the local capacitance of the coating. The measured values, which are random due to the randomness present in the coating, form the responses to challenges, each of them specified by a voltage of a certain frequency and amplitude and applied to a region of the sensor array. Coating PUFs can be used, e.g., for RFID-tags. As for key extraction, it is possible to generate on the average 45 uniformly distributed bits by using 30 sensors [Tuyls et al., 2006].

The advantage of coating PUFs is that their production price is very low. Another benefit of coating PUFs is that they are suitable for detecting a certain level of physical tampering. If a device is physically attacked, its response behavior is likely to change; thus, tampering can be uncovered by measuring the PUF with specific challenges. Due to tampering, the responses usually change only locally, which could even allow the determination of specific attack positions on the chip surface.

2.1.6 Controlled PUFs

The security of cryptographic protocols based on PUFs crucially relies on the unpredictability of their responses. Yet, in many implementations, nothing prevents an attacker to query all possible challenges (in particular if there are not too many of them), store the CRPs, and then successfully simulate the PUF in software.

To overcome this problem, Gassend et al. [Gassend et al., 2002a] introduced the concept of *Controlled Physically Uncloneable Functions* (CPUF). Controlled PUFs can only be evaluated via a control layer that is physically linked to a PUF in an (assumed) inseparable way [Gassend et al., 2008, Gassend et al., 2004]. That is, any interaction with the PUF requires evaluating an algorithm, specified in this layer. Furthermore, any attempt to remove the control layer will damage the device, e.g., the control layer is embedded within the physical system of the PUF. Thus, the PUF protects the control layer from invasive attacks, while the control layer protects the PUF from protocol attacks.

In [Gassend et al., 2002a], the authors present a protocol for establishing a shared secret between a PUF-device and a remote user, which can then be used as a subroutine in solving other cryptographic tasks, like certified execution or smart-card authentication. The security of the protocol relies on computational assumptions, as it employs one-way functions (for the channel-authentication purposes), but also public-key encryption.

2.2 Applications of Physically Uncloneable Functions

The properties of a PUF make them usable for many applications. In particular they are suitable for identification, authentication, or key storage. Pappu [Pappu, 2001], for example, uses the PUF's challenge/response pairs to identify specific devices or to extract cryptographic keys [Pappu, 2001, Pappu et al., 2002, Pappu, 2003]. To this end, the PUF is measured right after production on a few random challenges (this step is referred to as *enrollment*) to obtain a database of valid challenge/response pairs for a particular device. A device can subsequently be identified once it is placed in the field, by measuring the response for one of the challenges selected during enrollment (this process is usually called verification). If the response matches the expected, pre-recorded response, the device is authenticated and the response can be used to derive keys.

2.2.1 PUFs and Fuzzy Extractors

In [Škorić et al., 2005] the authors point out that the output of a PUF is noisy by nature. In some application, such as authentication, it is enough to apply a distance function to deal with noisy measurements. In other application scenarios, however, a distance function is not enough because a noise-free output with a uniform distribution (such as cryptographic keys) is required. To deal with this problem, *fuzzy extractors* of Dodis et al. [Dodis et al., 2008] are applied—a secure form of error correction that enables a reliable extraction of a uniform key from a noisy non-uniform input with sufficient entropy (see Section 3.5).

Some errors can also be avoided by employing a calibration operation, which is driven by PUF challenge/response pairs, as described in [Škorić et al., 2005]: the PUF response is measured for a small number of publicly known challenges; the differences between the measured and the publicly available expected responses allows to calibrate the measurement process.

2.2.2 PUF-based Identification, Authentication, and Key Extraction

A typical application of PUFs lies in the area of smart-card based identification, where a low-cost PUF is embedded directly into the card [Pappu, 2001, Gassend, 2003]. This would defeat card cloning, where the content of a card (possibly including a cryptographic key) is illicitly read out by a reader and written onto a blank card. PUFs can counter this attack in two ways: first, they can implicitly "store" a cryptographic key, which makes read-out of a key more complex. To store means that ideally, the cryptographic key would not be permanently stored in the device but generated only when required. Second, due to their physical uncloneability, physical presence of a specific card in a reader can be assured.

In the original work Pappu [Pappu, 2001] achieved this by embedding the PUF into the card, while the expensive optical equipment, such as the laser or the speckle detector, only needs to be present in the card reader. The reader is connected via a secure channel to a server, which possesses several PUF CRPs for each card (measured during enrollment). Upon a request of the card reader, it obtains a randomly chosen challenge from the server. With this challenge, the reader measures the PUF and obtains the corresponding response. After forwarding the response to the server, the server checks that the received response matches the expected response. Depending on this result, the card is either accepted or rejected.

An adversary, trying to spoof the device, either has to clone the card, or to predict the PUF input-output behavior in some other way. Note that this only works if the challenge/response space is large, else one can evaluate the PUF in software. The first attack is impossible due to uncloneability of the PUF, while the second one is infeasible since PUF responses are (assumed) difficult to learn.

The obvious drawback of this mechanism is the increase in the cost of the card, as an optical PUF must be included. It would be beneficial to utilize a PUF implementation, which comes "for free"—this is, which uses the same technology as the card itself. In the context of smart-cards using conventional integrated circuits, this could be a PUF based on silicon technology. A second shortcoming lies in the on-line nature of the protocol.

2.2.3 IP Protection

Protecting software that runs on embedded systems is a problem of growing importance. Instead of building functionality entirely in hardware, many vendors utilize standard computing equipment and differentiate through software. Unfortunately, software can easily be copied and reverse-engineered. A particular problem is professional product piracy, where software is copied from one legitimate device, and installed on many other (unauthorized) products. Many vendors thus want to bind software against a specific hardware platform or even to a specific instance.

The latter can be achieved by PUFs as shown by [Simpson and Schaumont, 2006]. The basic idea is to include a mutual authentication protocol between the provider's software (also called Intellectual Property, IP) and the hardware platform. In this scenario, the PUF is part of an FPGA and it is used for hardware authentication and key generation. The FPGA bit-stream is distributed in encrypted form, where the key is derived from a response to a specific PUF challenge. Thus, the bit-stream cannot be decrypted and run on a FPGA that it has not been personalized for. Guajardo et al. [Guajardo et al., 2007] revisited the result of Simpson and Schaumont and suggested improvements—instead of treating the PUF as a black-box, they propose a FPGA based IP protection mechanism, which relies on SRAM PUFs [Guajardo et al., 2007, Guajardo et al., 2008]. Similarly, Gora et al. [Gora et al., 2009] and Atallah et al. [Atallah et al., 2008] proposed the use of PUFs in binding software against specific hardware.

2.2.4 Unique Labels

A related concept is that of Certificates of Authenticity (COA). COAs are unique labels to identify objects and link them with a notion of "authenticity", and can be viewed

as PUFs, in the sense that they are uncloneable (despite having only a single challenge). The work mainly started in the domain of anti-counterfeiting of banknotes and valuable objects; in this application domain, the labels are used to check the originality of an object. There has been an increased interest in the use of uncloneable structures, similar to PUFs, for product anti-counterfeiting. For example, DeJean and Kirovski [DeJean and Kirovski, 2007] apply COAs in the context of Radio-frequency identification tags (RFID-tags), called RF-DNA. The RF-DNA concept supports unique and uncloneable physical fingerprints. When the device is exposed with an electromagnetic wave the subtleties in the reaction of the device form the fingerprints. The advantage of these unique identifiers is that they can be read wireless using radio waves.

2.2.5 PUFs and Cryptographic Primitives

Increasingly, cryptographers have been drawn to PUFs as an attractive alternative to traditional cryptographic primitives. Since the output of a PUF is (assumed) unpredictable and random, one can use PUFs as a hardware source of randomness. Moreover, we can see a PUF as an uncloneable storage mechanism for private keys, which makes them suitable for authentication protocols. PUF-based challenge/response authentication protocols have been proposed in [Tuyls et al., 2007c, Frikken et al., 2009]. In [Busch et al., 2009] the authors note that current PUF-based authentication only prevents the impersonation of the client and do not prevent impersonation of the server in the context of physical attacks. To solve this problem, one needs a mechanism that allows the client to distinguish between challenges selected by the server in the enrollment step and an attacker. For a detailed discussion see Chapter 4. PUFs have also been considered in the context of RFID authentication to prevent man-in-the-middle attacks [Hammouri and Sunar, 2008].

Since many applications in cryptography require random numbers (consider for example the generation of nonces) O'Donnell et al. build a cheap and practical hardware random number generator from a silicon PUF [O'Donnell et al., 2004]. This hardware generator does not require any private seed because its random output results from uncontrollable manufacturing variations.

A major goal in cryptography is the ability to securely generate, store and retrieve keys. In [Maes et al., 2012] the authors proposed the first PUF-based key generator. Beside

their theoretical study the authors presented a fully functional implementation, which can generate a cryptographically secure 128-bit key.

Armknecht et al. propose a first formal PUF model in [Armknecht et al., 2009]. The authors introduce a block cipher, which is entirely based on PUFs; to this end, the authors apply the well-known Luby-Rackoff transformation, which turns pseudo-random functions into pseudo-random permutations. Furthermore, the authors explicitly distinguish between algorithmic and physical properties of a PUF. Algorithmic properties mean that the PUF is a kind of a noisy function (noise) for which the distribution of PUF-outputs is usually non-uniform and two different PUFs show completely independent behavior (non-uniform output and independence) [Armknecht et al., 2009]. Physical properties mean that a PUF is not physically cloneable and that it is tamper evident.

Researchers also tried to use PUFs in the context of public-key cryptography. Beckmann and Potkonjak proposed a Public Physical Uncloneable Function (PPUF) in [Beckmann and Potkonjak, 2009]. A PPUF is a PUF that is indeed simulatable but it takes very large time. The authors apply PPUFs in the context of secret key exchange protocols and public key protocols. A very similar concept was proposed by Rührmair in [Rührmair, 2009]. The author calls his system SIMPL that stands for SImulation Possible but Laborious.

In [Rührmair, 2010] the author realize for the first time an PUF-based oblivious transfer (OT) protocol. Subsequently, Brzuska et al. present an oblivious transfer protocol in the universally composable framework [Brzuska et al., 2011a], which bears some similarity to the PUF-based OT-protocol of [Rührmair, 2010]. His protocol, however, has a high round complexity due to an interactive hashing step. Still, [Rührmair, 2010] points out that, using symmetry of oblivious transfer in the sense that one can change the roles of sender and receiver, one obtains an oblivious transfer protocol in which the other party sends the PUF. We confirm that this symmetry also holds in the UC setting. For a detailed discussion of the PUF-based OT-protocol in the UC setting see Chapter 5. Later on, Rührmair and van Dijk analyze the results of [Brzuska et al., 2011a] in [Rührmair and van Dijk, 2012]. For more details see Section 1.4.

Ostrovsky et al. extended the results of [Brzuska et al., 2011a] since their model considers only trusted PUFs. But in real-world adversaries could be able to create mali-

cious PUFs [Ostrovsky et al., 2012]. Thus, the authors extend the PUF functionality of Brzuska et al. in that way that adversaries can create arbitrarily malicious PUFs.

3 Formalizing Physically Uncloneable Functions

3.1 Preliminaries

3.1.1 Notation

Throughout this thesis, $\lambda \in \mathbb{N}$ denotes the security parameter. Every algorithm A is a probabilistic Turing machine that runs in time polynomial in λ unless indicated otherwise. We say that a function is negligible if it vanishes faster than the inverse of any polynomial. We usually refer to such a function as $\mathrm{negl}(\lambda)$.

3.1.2 Distance, Entropy

Let A_ℓ be a probability distribution on $\{0,1\}^\ell$ and let $v = A_\ell$ denote the event that v has been sampled according to A_ℓ. For simplicity, we use in the following A instead of A_ℓ. Furthermore, we denote by U_ℓ the random variable with the uniform distribution on ℓ-bit binary strings.

The statistical distance SD of two random variables A and B, drawn from a set $\mathcal{V}$, is defined as

$$\mathrm{SD}(A,B) = \frac{1}{2}\sum_{v\in\mathcal{V}} |\,\mathrm{Prob}[\,A=v] - \mathrm{Prob}[\,B=v]\,|.$$

A set $\mathcal{M}$ with a distance function $\mathsf{dis} : \mathcal{M}\times\mathcal{M} \to \mathbb{R}^+ = [0,\infty)$ is called a metric space. Let w and w' be two equal-length strings. The number of locations in which they differ is denoted by $\mathsf{dis}_{\mathsf{ham}}(w,w')$ and is called the Hamming distance.

The ability that an adversary guesses for example a secret key is an important indication for security purposes. This ability is captured via the so-called *min-entropy*. The min-entropy of a random variable A is a lower-bound on its entropy and it is often used as a worst-case measure of the unpredictability of a random variable A [Barker and Kelsey, 2012]. In particular, the min-entropy of a random variable X is defined by

$$H_\infty(X) = -\log(\max_x \text{Prob}[X = x]).$$

The min-entropy of a random variable X is minus the logarithm of the most likely event where $\max_x \text{Prob}[X = x]$ denotes the probability that the random variable X is guessed correctly in one go.

Similarly, for two random variables X and Y, the average min-entropy of X given Y is defined as

$$\widetilde{H}_\infty(X) = -\log\left(\mathbb{E}_{y \leftarrow Y}\left[2^{-H_\infty(X|y=Y)}\right]\right).$$

Average min-entropy is simply the logarithm of the probability that the adversary, given the variable of Y, will guess the variable of X in a single attempt.

3.2 Existing Approaches of PUF Modeling

Modeling PUFs appropriately is a highly non-trivial task. Most importantly, there are different types of PUFs with different (physical) properties. Furthermore, there does not seem to be a general agreement upon common security properties of PUFs even for a single type only (e.g., whether a PUF should be one-way or not, or if the output should be pseudorandom).

There have been several approaches to define PUFs cryptographically. In the following we give an overview of the most prominent approaches of defining PUFs.

3.2.1 Physical One-way Function

Pappu introduced the concept of *physical one-way functions* in his seminal work, see [Pappu, 2001]. The author formalizes a physical one-way function as a deterministic physical interaction between a physical system, e.g., an epoxy, and a physical probe,

e.g., a laser beam that can be evaluated in constant time. On the other hand, inverting the system by a probabilistic-polynomial time adversary with non-negligible probability is not possible. Moreover, (1) simulating a response, given the physical system and the physical probe, and (2) constructing a new physical system (aka. clone), which exactly behaves as another physical system, should be hard.

However, Pappu's definition of a physical one-way function only fits to PUFs that are based on an optical medium—note that at that time the optical PUF was the only known PUF. Most of today's PUFs, e.g., IC-based PUFs, do not fulfill Pappu's formalization since the definition is really restrictive. Consider for example the one-wayness. An IC-based PUF has a very small output space, possibly only one bit, and thus, inverting this kind of PUF is very easy. Consequently, most IC-based PUF does not fulfill Pappu's formalization.

Moreover, Pappu characterizes a physical one-way function as a *deterministic* physical interaction. The terminus *deterministic* is really critical in the context of physical evaluations since all known PUFs imply noisy responses caused by uncontrollable physical effects (note, that the concept of a PUF exploits exactly this inaccuracy).

3.2.2 Physical Random Function

Gassend et al. introduced the term of *physical random function* when they proposed the first IC-based PUF [Gassend, 2003, Gassend et al., 2002b]. A physical random function is a function that (1) maps challenges to responses, (2) is embodied by a physical device, (3) is easy to evaluate, and (4) hard to characterize (i.e., after collecting a polynomial number of challenges and responses, it is hard for an adversary to predict the response to a randomly chosen challenge). An IC-based PUF exploits incidents in circuits, which are rather undesirable for engineers. That results in inherent variations in integrated circuits. For a detailed discussion see Chapter 2.1.2.

The approach of Gassend et al. is indeed a further step of formalizing PUFs; e.g., the assumption that a PUF is hard to characterize is a very simplified and first attempt of formalizing the unpredictability property of PUFs in which the one-wayness property of physical one-way functions is been eased. But still, it is rather informal and key PUF properties, such as uncloneability and unpredictability, are still missing.

3.2.3 Strong and Weak PUFs

Guajardo et al. [Guajardo et al., 2007] introduce the concept of memory-based PUFs and formalized PUFs as a system, which consists of a physical and an operating part. The physical part consists of an inherently uncloneable physical system where uncloneability comes from uncontrollable process variations during their fabrication. The operating part includes a function that maps challenges to responses. Beside the physical and operating part of PUFs, the authors also defined PUF properties: (1) the responses are independent of each other, (2) guessing the response to a challenge without evaluating the PUF is difficult, and (3) attacking the PUF changes its challenge/response behavior completely.

Guajardo et al. defined for the first time explicitly the uncloneability property of PUFs. Moreover, the authors divided PUFs into *strong* and *weak* PUFs. Strong PUFs have a large number of challenge/response pairs such that an efficient adversary, measuring only a few challenge/response pairs, cannot predict the response for a random challenge with high probability. If the number of different challenge/response pairs is rather small, the authors speak of a weak PUF. Guajardo et al. mentioned also for the first time that PUF measurements are influenced by noise. That is evaluating the PUF twice on the same challenge causes in different but related responses. However, for some applications such as cryptographic purposes it is required to have a completely noisy-free output with a perfect uniform distribution. Applying Fuzzy Extractors or Helper Data Algorithm can solve this problem. For a detailed discussion see Chapter 3.5.

3.2.4 On the Foundation of Physical Unclonable Functions

[Rührmair et al., 2009] gave a very detailed overview of existing PUF approaches. In doing so, the authors objects that (1) the entropy of finite physical systems is always polynomial bounded; (2) thus, there cannot exist a PUF which fulfills existing PUF definitions (recall, after collecting a polynomial number of challenges and responses, it is hard for an adversary to predict the response to a randomly chosen challenge, see 3.2.2). Rührmair et al. proposed a new PUF definition where, roughly speaking, it must be hard for an attacker to predict the PUF. More precisely, the authors based their approach on [Guajardo et al., 2007], but enhanced their PUF definition by using security games. In doing so, they renamed weak PUFs as obfuscating PUFs, since the

PUF is used as a non-volatile storage of a secret key. Then the authors defined the corresponding security game as follows: An adversary $\mathcal{A}$ has access to the PUF for a limited period of time. Afterwards it is infeasible for $\mathcal{A}$ to learn the key. In the security game for strong PUFs the adversary has access to both the PUF and a PUF oracle for a limited period of time. Subsequently, the adversary $\mathcal{A}$ has to predict the response to a randomly chosen challenge.

Although the definition of Rührmair et al. is a formal approach there are still some open issues, e.g., the authors do not take into account that responses are noisy measurements.

3.2.5 A Formal Foundation for the Security Features of Physical Functions

In concurrent and independent work, Brzuska et al. proposed their PUF definition at CRYPTO'11 [Brzuska et al., 2011a] and Armknecht et al. published also a PUF definition at the IEEE Symposium on Security and Privacy [Armknecht et al., 2011]. While the above-mentioned PUF definitions are either rather informal, or follow the more stringent game-based approach, but stipulate uncloneability and tamper-resistance as an external property "outside of the game", Armknecht et al. provide a game-based definition for uncloneability on a physical level. More precisely, the authors introduce a physical function (PF), which is a probabilistic procedure. The physical function is a combination of a device and an evaluation procedure that maps a set of challenges to a set of responses. In this context, the authors mentioned that the input/output behavior of the device depends on its physical properties and on uncontrollable random noise. Beside uncloneability, Armknecht et al. define also robustness and unpredictability of PUFs. Finally the authors show that their defined security properties can be achieved by known PUF instantiations.

3.3 Modeling PUFs

In the following we give a new approach of modeling PUFs. After describing the functional properties of PUFs we will define the PUF's properties such as uncloneability, unpredictability, uncorrelated output, one-wayness, and tamper-resistance.

A PUF-family $\mathcal{P}$ consists of two algorithms Sample and Eval. The index sampling algorithm Sample, which obtains as input the security parameter and returns as output an index id of the PUF family corresponds to the PUF fabrication process. The evaluation algorithm Eval takes as input a challenge c, evaluates the PUF on c, and generates as output the corresponding response r.

Note that the usage of hardware components, especially of PUFs, causes several unpleasant side effects, though. At foremost, PUFs are not known to be implementable by probabilistic polynomial-time (PPT) Turing machines; the manufacturing process seems to be inherently based on physical properties. Furthermore, we require the challenge space to be equal to a full set of strings of a certain length. For some classes of PUFs, this is naturally satisfied, for example arbiter PUFs and SRAM PUFs. For others types this can be achieved through appropriate encoding, as for angles in optical PUFs. A PUF is now defined as follows:

Definition 3.3.1 (Physically Uncloneable Functions). *Let rg be the dimension of the range of the PUF responses of the PUF family, and let d_{noise} be a bound on the PUF's noise. A pair $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ is a family of (rg, d_{noise})-PUFs if it satisfies the following properties:*

Index Sampling. *Let $\mathcal{I}_\lambda$ be an index set. The sampling algorithm Sample outputs, on input the security parameter 1^λ, an index $\mathsf{id} \in \mathcal{I}_\lambda$. We do not require that the index sampling can be done efficiently. Each index $\mathsf{id} \in \mathcal{I}_\lambda$ corresponds to a set $\mathcal{D}_{\mathsf{id}}$ of distributions. For each challenge $c \in \{0,1\}^\lambda$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(c)$ on $\{0,1\}^{rg(\lambda)}$. We do* not *require that $\mathcal{D}_{\mathsf{id}}$ has a short description or an efficient sampling algorithm.*

Evaluation. *The evaluation algorithm Eval gets as input a tuple $(1^\lambda, \mathsf{id}, c)$, where $c \in \{0,1\}^\lambda$. It outputs a response $r \in \{0,1\}^{rg(\lambda)}$ according to distribution $\mathcal{D}_{\mathsf{id}}(c)$. It is* not *required that Eval is a probabilistic polynomial-time algorithm.*

Bounded Noise. *For all indices $\mathsf{id} \in \mathcal{I}_\lambda$, for all challenges $c \in \{0,1\}^\lambda$, we have that when running $\mathsf{Eval}(1^\lambda, \mathsf{id}, c)$ twice, then the Hamming distance of any two outputs r_1, r_2 of the algorithm is smaller than $d_{\mathsf{noise}}(\lambda)$.*

Instead of $\mathcal{D}_{\mathsf{id}}(c)$, we usually write $\mathsf{PUF}_{\mathsf{id}}(c)$. Moreover, if misunderstandings are unlikely to occur, we write $\mathcal{D}(c)$ instead of $\mathcal{D}_{\mathsf{id}}(c)$ and PUF instead of $\mathrm{PUF}_{\mathsf{id}}$. Finally, we usually write rg instead of $rg(\lambda)$ and $\mathcal{I}$ instead of $\mathcal{I}_\lambda$.

3.4 Security Properties of PUFs

Various security properties of PUFs have been introduced in the literature such as uncloneability, unpredictability, uncorrelated output, one-wayness, or tamper-resistance. The main security properties of PUFs are *uncloneability* and *unpredictability*. Unpredictability is covered via an entropy condition on the PUF distribution. This condition also implies mild forms of uncloneability as well as uncorrelated outputs, as shown in Section 3.4.2. In the following we give a detailed analysis of the PUF's properties as well as the relation to existing approaches.

3.4.1 Uncloneability of PUFs

Loosely speaking uncloneability says that a malicious party cannot "copy" a PUF. This property can be defined in two flavors: *hardware-uncloneability* and *software-uncloneability* [Pappu, 2001, Tuyls et al., 2007b, Maes and Verbauwhede, 2010].

Hardware uncloneability, also called *physical uncloneability*, means that it is difficult to create a physical clone of a PUF. Recall that a PUF is produced through an uncontrollable manufacturing process. Even if an adversary has control over the complete manufacturing process it is assumed that producing two PUFs, which implement the same function is extremely hard or impossible. For some PUF implementation the authors showed that physical cloning of their implementation is impossible, see for example [Pappu, 2001, Lim, 2004, Suh and Devadas, 2007, Guajardo et al., 2007, Tuyls et al., 2006, Maes and Verbauwhede, 2010].

Physical properties are usually infinite properties, e.g., there exists infinite many configurations. However, for cryptographic applications, we are also interested in its discrete mathematical behavior as a (noisy) function and thus consider cloneability merely with respect to its input/output behavior, resp. its implemented function. Software uncloneability, also called *model building*, requires that it should be hard to provide an algorithm that models the function implemented by a PUF (with only collecting a polynomial number of queries because of the unpredictability property of PUFs). Model building usually considers mere software models, while we additionally allow the software to use a second, different PUF to achieve its cloning goal.

In [Sadeghi et al., 2010] the authors present a first approach of a game-based formalization of hardware uncloneability. However, as software uncloneability implies hardware uncloneability, the below game-based definition is strictly stronger than the one presented in [Sadeghi et al., 2010]. Cryptographic applications that allow the adversary to read out arbitrarily many challenge pairs, require software uncloneability. Note that, however, in a weakened adversarial model, physical uncloneability can be sufficient.

Now, we suggest to the definition of uncloneability. Intuitively, a probabilistic polynomial-time algorithm $\mathcal{A}$ intends to clone a PUF_1 while having access to arbitrarily many interdependently generated PUFs PUF_2,...,PUF_λ. Moreover, $\mathcal{A}$ might measure the challenge PUF_1 adaptively on challenges $c_1, ..., c_k$ of its choice. The algorithm $\mathcal{A}$ then finishes the cloning process. From now on, $\mathcal{A}$ is denied access to the device PUF_1 and shall simulate its input/output behavior. The challenger now draws a random challenge c. With high probability (see Section 3.4.3 where we define uncorrelated output), the challenge c is not close to any of its previous challenges. The adversaries' task is to output the corresponding response value r. Note that we restrict the observable parameters of the PUF to be its input/output behavior.

Game UNC$(\mathsf{PUF}, \mathcal{A}, \lambda)$

Learning Phase Proceeding adaptively, the adversary has access to an oracle Sample, which draws a PUF index id from the PUF family according to Sample as well as to Eval oracles that evaluate $\mathsf{PUF}_{\mathsf{id}}$. The index of the first sampling query is denoted by id_0.

Simulation Phase Eventually, we withdraw the adversary's possibility to query $\mathsf{PUF}_{\mathsf{id}_0}$. The adversary can now be queried with input challenge values c. We denote this adversary by $\mathcal{A}$. For a randomly chosen challenge c, it is required to output answers r that are (computationally) indistinguishable from answers given by $\mathsf{PUF}_{\mathsf{id}_0}$. An (efficient) distinguisher $\mathcal{D}$ is given two oracles that are either both equal to $\mathsf{PUF}_{\mathsf{id}_0}$, or one oracle equal to $\mathcal{A}$, and one oracle equal to $\mathsf{PUF}_{\mathsf{id}_0}$.

For each (efficient) distinguisher $\mathcal{D}$, we denote by $\mathbf{Unc}_{\mathcal{A},\mathcal{D}}(\lambda)$ the difference

$$|\operatorname{Prob}\left[\mathcal{D}^{\mathcal{A},\mathsf{PUF}_{\mathsf{id}_0}} = 1\right] - \operatorname{Prob}\left[\mathcal{D}^{\mathsf{PUF}_{\mathsf{id}_0},\mathsf{PUF}_{\mathsf{id}_0}} = 1\right].|$$

Definition 3.4.1 (Cloneability of PUFs). *A PUF is cloneable if there exists an efficient algorithm $\mathcal{A}$ such that for all (efficient) distinguisher $\mathcal{D}$, the distinguishing advantage* $\mathbf{Unc}_{\mathcal{A},\mathcal{D}}(\lambda)$ *is negligible in* λ.

Definition 3.4.2 (Uncloneability of PUFs). *A PUF is uncloneable if it is not cloneable.*

The latter definition is equivalent to the game-based definition of unpredictability provided in a concurrent and independent work by [Armknecht et al., 2011]. Note that Armknecht et al. consider a combination of a fuzzy extractor and a PUF, while the above definition considers a mere PUF.

3.4.2 Unpredictability of PUFs

As considered earlier, unpredictability of PUFs states that it is difficult to compute the output to a chosen stimulus without evaluating the PUF. This property should hold even if further CRPs are given (e.g., one party has collected several CRPs before). The difference to "regular" unpredictable functions is that the attacker may indeed be able to predict response values for challenges that are closely related to already known challenges. That is, some PUFs map closely related measurements to closely related outputs.

Now we turn to the definition of unpredictability. An attacker first collects several PUF measurements for challenge value of its choice up to a certain moment where it has no longer access to the device. Subsequently, the attacker has to output a valid challenge/response pair (c^*, r^*) such that the distance between its prediction and all previously measurements is at least ϵ. If this conditions is fulfilled and if $r^* \leftarrow \mathsf{PUF}(c^*)$, then the attacker wins the game. We define unpredictability of PUFs in the following game between a challenger and an adversary $\mathcal{A}$:

Game PRE$(\mathsf{PUF}, \mathcal{A}, \lambda)$

SETUP The challenger generates a PUF by running the generation procedure Sample to obtain a PUF index x.

QUERIES Proceeding adaptively, the adversary $\mathcal{A}$ may query the PUF oracle $\mathcal{O}_{\mathsf{PUF}}$ on challenges $c_i \in \{0,1\}^\lambda$. The adversary returns $r_i \leftarrow \mathsf{PUF}(c_i)$ to the attacker and adds c_i to a list $\mathcal{Q}$ (list of challenge/response pairs queried by the attacker).

OUTPUT Eventually, the adversary outputs a pair (c^*, r^*) with $c^* \notin \mathcal{Q}$. The game evaluates the PUF on $r \leftarrow \mathsf{PUF}(c^*)$. It outputs 1 if the metric distance $\mathsf{dis}(c^*, c) > \epsilon(\lambda)$ for all $c \in \mathcal{Q}$ and $r = r^*$.

We define $\mathbf{Pre}_{\mathcal{A}}(\lambda)$ be the probability that $\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, \lambda)$ outputs 1.

For simplicity we require that $r^* \leftarrow \mathsf{PUF}(c^*)$ since PUFs usually used in combination with an extractor. This means that r^* is either close enough to its defined output range, or the extractor does not work appropriately.

Definition 3.4.3 (Unpredictability of PUFs). *A PUF is ϵ-unpredictable with respect to the game* $\mathbf{PRE}(\mathsf{PUF}, \mathcal{A}, \lambda)$ *if for all efficient algorithms $\mathcal{A}$ the probability* $\mathbf{Pre}_{\mathcal{A}}(\lambda)$ *is negligible in λ.*

Summing up, we defined unpredictability in terms of a game where an efficient adversary, after seeing some challenge/response pairs, tries to predict the response for another challenge, which is not within close distance to the previous queries; the success probability should then be negligible.

In some contexts, such as the UC framework, this approach is not optimal since the fuzzy extractors, which are necessary to eliminate the noise of a PUF, usually need a *fixed* lower bound on the min-entropy in order to be applicable. Thus, we also give an entropy-based definition of unpredictability, which is derived from the notation of unpredictability for random variables.

The behavior of the PUF on input a challenge c should be unpredictable, i.e., have some significant amount of intrinsic entropy, even if the PUF has been measured before on several challenge values. Here, (conditional) min-entropy is a main tool. It indicates the residual min-entropy on a response value for a challenge c, when one has already measured the PUF on (not necessarily different) challenges $c_1, ..., c_\ell$ before. Since the random responses are not under adversarial control we can look at the residual entropy of the answer to r by taking the (weighted) average over all possible response values $r_1, ..., r_\ell$. Demanding that a PUF has a certain average min-entropy is weaker than asking for all possible responses $r_1, ..., r_\ell$, that the residual entropy remains above a

certain level [Dodis et al., 2008]. This weaker requirement suffices for our purposes. However, the challenges c are chosen by the adversary such that we will ask the average min-entropy to be high for all challenges and defined by the maximal probability of a possible response r.

Definition 3.4.4 (Average Min-Entropy of PUFs). *The average min-entropy of* $\mathsf{PUF}(c)$ *conditioned on the measurements of challenges* $\mathcal{C} = (c_1, \ldots, c_\ell)$ *is defined by* $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathsf{PUF}(\mathcal{C}))$

$$:= -\log\left(\mathbb{E}_{r_i \leftarrow \mathsf{PUF}(c_i)}\left[max_r \operatorname{Prob}[\mathsf{PUF}(c) = r | r_1 = \mathsf{PUF}(c_1), \ldots, r_\ell = \mathsf{PUF}(c_\ell)]\right]\right)$$
$$:= -\log\left(\mathbb{E}_{r_i \leftarrow \mathsf{PUF}(c_i)}\left[2^{-H_\infty(\mathsf{PUF}(c)|r_1=\mathsf{PUF}(c_1),\ldots,r_\ell=\mathsf{PUF}(c_\ell))}\right]\right)$$

where the probability is taken over the choice of id *from* $\mathcal{I}$ *and the choice of possible PUF responses on challenge* c*. The term* $\mathsf{PUF}(\mathcal{C})$ *denotes a sequence of random variables* $\mathsf{PUF}(c_1), ..., \mathsf{PUF}(c_\ell)$ *each corresponding to an evaluation of the PUF on challenge* c_k*.*

We also write $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathcal{C})$ as an abbreviation for $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathsf{PUF}(\mathcal{C}))$. We now turn to our definition of unpredictability, which is derived from the notion of unpredictability for random variables.

Definition 3.4.5 (Unpredictability of PUFs). *We call a* (rg, d_{noise})*-PUF family* $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ *for security parameter* 1^λ $(d_{\mathsf{min}}(\lambda), m(\lambda))$*-unpredictable if for any* $c \in \{0,1\}^\lambda$ *and any challenge list* $\mathcal{C} = (c_1, \ldots, c_\ell)$*, one has that, if for all* $1 \leq k \leq \ell$ *the Hamming distance satisfies* $\mathsf{dis}_{\mathsf{ham}}(c, c_k) \geq d_{\mathsf{min}}(\lambda)$*, then the average min-entropy satisfies* $\tilde{H}_\infty(\mathsf{PUF}(c)|\mathsf{PUF}(\mathcal{C})) \geq m(\lambda)$*. Such a PUF-family is called a* $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$*-PUF family.*

Note that one could also define a computational version of unpredictability via computational min-entropy (aka. HILL entropy, named after [Håstad et al., 1999]) where the entropy is defined via the entropy of computationally indistinguishable random variables. All proofs considered in this thesis carry through when replacing statistical by computational min-entropy; we nonetheless use the statistical variant for sake of simplicity.

As explained earlier, indistinguishability for defining computational min-entropy then needs to be considered with respect to distinguisher that have PUF power, and not with respect to mere probabilistic polynomial-time algorithms.

3.4.3 Uncorrelated Output of PUFs

If challenges are members of close-by regions, their corresponding responses could be correlated, for example in the case of optical PUFs [Pappu, 2001, Tuyls et al., 2005]. However, if two challenges c_1, c_2 are far away from each other, namely more than d_{min}, then their responses should not be close to each other. This property is implied by unpredictability.

If the outputs of two such different challenges were closely correlated, then knowing the PUF response to challenge c_1 would provide more information about the PUF response to c_2 than allowed by unpredictability. We now formalize the notion of uncorrelated outputs in the following game:

Game UNCOR$(\mathsf{PUF}, d_{\mathsf{min}}, \rho_{\mathsf{max}}, \mathcal{A}, \lambda)$

SETUP The challenger generates a PUF by running the generation procedure Sample to obtain a PUF index id. The adversary adaptively generates challenges c_i, which it may query to a PUF several times in arbitrary order. The t^{th} response to the challenge c_i is denoted by $r_{i,t}$. Each time the challenger returns $r_{i,t} \leftarrow \mathsf{PUF}(c_i)$ to the attacker, it adds $c_i, r_{i,t}$ to a list $\mathcal{Q}$.

OUTPUT Eventually, the adversary outputs $(c_i^*, r_i^*), (c_j^*, r_j^*)$. The game outputs 1 if the queries output by the adversary are in $\mathcal{Q}$, $\mathsf{dis}(c_i^*, c_j^*) > d_{\mathsf{min}}(\lambda)$ and the metric distance between the two responses is at most $\rho_{\mathsf{max}}(\lambda)$, e.g., $\mathsf{dis}(r_i^*, r_j^*) \leq \rho_{\mathsf{max}}(\lambda)$.

We define $\mathbf{Cor}_{\mathcal{A}}(\lambda)$ as the probability that $\mathbf{UNCOR}(\mathsf{PUF}, d_{\mathsf{min}}, \mathcal{A}, \lambda)$ returns 1.

Definition 3.4.6 (Uncorrelated Output of PUFs). *A PUF satisfies $(d_{\mathsf{min}}, \rho_{\mathsf{max}})$ uncorrelated output if for all efficient adversaries $\mathcal{A}$ the probability $\mathbf{Cor}_{\mathcal{A}}(\lambda)$ is negligible in λ.*

3.4.4 One-Wayness of PUFs

One-wayness of PUFs means that evaluating a PUF is easy while inverting it remains hard. That is, given a response r it should be hard to find a corresponding challenge c such that for $r' \leftarrow \mathsf{PUF}(c)$, the responses r and r' are close. A PUF family is one-way, if no algorithm $\mathcal{A}$ outputs such a challenge c with more than non-negligible probability. Note that for PUFs with a small input range one-wayness is not achievable as outputting

a random input already yields a non-negligible probability in inverting the PUF, e.g., simple arbiter PUFs. Moreover, measuring the PUF on all input values increases the probability to 1.

Our notion of unpredictability is incomparable to one-wayness, i.e., a PUF may satisfy unpredictability without being one-way, and conversely, a PUF can be one-way while not satisfying unpredictability (we give two separation examples at the end of this section). This shows that one-wayness does not imply that the PUF has any entropy at all which is, however, one of the main stated goals in PUF design. In the first work, Pappu considered PUFs in the context of entropy [Pappu, 2001]. Later on, people studied/revisited the notion of entropy of optical PUFs as well as of further PUF implementations [Tuyls et al., 2005, Škorić et al., 2006b, Škorić et al., 2006a, Ignatenko et al., 2006, Tuyls et al., 2007a]. This shows that one-wayness as a mere assumption is not sufficient. However, one-wayness is a desirable property, which, however, seems to be hard to realize, as many of the common PUF types are not one-way, such as arbiter PUFs, ring-oscillator PUFs, SRAM PUFs and coating PUFs. Towards applicability for many PUF types, we thus discard the assumption of one-wayness.

We first define one-wayness formally and then turn to the bidirectional separation between unpredictability and one-wayness.

Game INV$(\mathsf{PUF}, \mathcal{A}, \lambda)$

SETUP The challenger generates a PUF by running the generation procedure Sample to obtain a PUF index id. It picks a challenge c at random and evaluates $r \leftarrow \mathsf{PUF}(c)$. It hands r to the adversary.

OUTPUT The adversary may adaptively query the PUF and finally outputs a challenge c. The challenger computes $r' \leftarrow \mathsf{PUF}(c)$ and outputs 1 if $\mathsf{dis}(r', r) < d_{\mathsf{noise}}$.

We define $\mathbf{Inv}_{\mathcal{A}}(\lambda)$ as the probability that $\mathbf{INV}(\mathsf{PUF}, \mathcal{A}, \lambda)$ returns 1.

Definition 3.4.7. *A PUF is one-way with respect to the game* $\mathbf{INV}(\mathsf{PUF}, \mathcal{A}, \lambda)$ *if for any efficient algorithm* $\mathcal{A}$ *the probability* $\mathbf{Inv}_{\mathcal{A}}(\lambda)$ *is at most* $\delta_{\mathbf{Inv}}$.

The following separations illustrate that unpredictability and one-wayness do not imply each other.

Unpredictability does not imply One-Wayness Assume, a PUF-family satisfies our notion of unpredictability for suitable parameters. Then, we can define a new PUF-family PUF*, which on input challenge c queries PUF on challenge c. The PUF then returns a response r, and PUF* returns as its response R the pair $R = (c, r)$. The high-entropy condition of our unpredictability notion is still satisfied. However, PUF* is not one-way.

One-Wayness does not imply Unpredictability Let PUF be a family of one-way-functions f_i with an index set $\mathcal{I}$ and with efficient descriptions. We define a new function family f_i^* with an index set $\mathcal{I}$ as follows: On input a value c, f_i^* outputs $f_i(c)$ concatenated with a function description of f_i. This construction is still one-way. However, it is not unpredictable. After querying the function on c, one can easily derive the function value on any other input c'. However, this example assumes knowledge of the code of the PUF. Possibly this is not true, but then we can consider a second example: We define a function $\hat{f}(c) := f(c) \parallel f(\hat{c}) \parallel \hat{c}$, for some fixed value $\hat{c}$ (that is chosen uniformly at random during the creation phase). Now, there exists an adversary that easily predicts this value after querying the PUF once (on a random value). This concludes the separation.

3.4.5 Tamper-Evidence of PUFs

Tamper-resistance means that physical manipulation should be either impossible or easy to detect, e.g., tampering changes the input/output behavior of the PUF. Our modeling of PUFs implicitly assumes tamper-evidence, as the attacker is bound on querying the PUF. There are two ways to approach this topic: One may consider this as a human-related property, i.e., manipulating is easy to detect for a human. The other approach would be to assume that the mathematical behavior becomes atypical, or, that the mathematical behavior becomes very different. In the latter case, when receiving a PUF, the sender and the receiver needs to exchange a number of challenge/response pairs, i.e., the sender sends a couple of challenges, and the receiver returns the PUF's responses. If they are close enough to the values the sender has stored in its challenge/response pair list, the sender will confirm that no tempering has occurred. In our model, this property is implicit. When desired, one can define a more basic PUF definition which allows the adversary mathematical tempering. One would then show that this definition could be composed with the above protocol to emulate our PUF definition.

3.5 Fuzzy Extractors

The responses of PUFs are noisy by nature and thus, the output of a PUF cannot directly be used in applications that require noise-free output with a perfectly uniform distribution (such as cryptographic keys). In particular, measuring the PUF on the same stimuli results in closely related but different outputs. To deal with this problem we apply a fuzzy extractor. A fuzzy extractor, introduced by Dodis, Ostrovsky, Reyzin, and Smith [Dodis et al., 2008], is a secure form of error correction that enables a reliable extraction of a uniform key from a noisy non-uniform input.

A fuzzy extractor consists of a pair of algorithms $(\mathsf{Gen}, \mathsf{Rep})$. The generation algorithm Gen takes as input a noisy measurement w and generates as output a secret st together with helper data p. The helper data can be stored publicly, since it does not reveal information about the secret. It is later used to reproduce the same secret st from related measurements. That is, the reproduction algorithm Rep takes as input a noisy measurement w' and helper data p. If w and w' are sufficiently close, Rep gives the same reply st. The value st is distributed almost uniformly and can thus be used for cryptographic purposes.

Definition 3.5.1 (Fuzzy Extractor). *Let* dis *be a distance function for metric space* $\mathcal{M}$*. An* (m, ℓ, t, ϵ)*-fuzzy extractor consists of two efficient randomized algorithms* (Gen, Rep)*:*

Gen*: The algorithm* Gen *outputs on input* $w \in \mathcal{M}$ *a secret string* $st \in \{0,1\}^\ell$ *and a helper data string* $p \in \{0,1\}^*$*.*

Rep*: The algorithm* Rep *takes an element* $w' \in \mathcal{M}$ *and a helper data string* $p \in \{0,1\}^*$ *and outputs a string* st*.*

Correctness: *If* $\mathsf{dis}(w, w') \leq t$ *and* $(st, p) \leftarrow \mathsf{Gen}(w)$*, then* $\mathsf{Rep}(w', p) = st$*.*

Security: *For any distribution* $\mathcal{W}$ *on the metric space* $\mathcal{M}$ *of min-entropy* m*, the first component of the random variable* (st, p)*, defined by drawing* w *according to* $\mathcal{W}$ *and then applying* Gen*, is distributed almost uniformly, even if* p *is observed, i.e.,* $SD((st, p), (U_\ell, p)) \leq \epsilon$*.*

3.5.1 Applying Fuzzy Extractors to PUFs

We now determine parameters to combine a PUF and the fuzzy extractor in order to achieve almost uniformly random values. We let the parameters of the fuzzy extractor depend on the parameters of the PUF. Assume that we have a $(rg(\lambda), d_{\mathsf{noise}}(\lambda), d_{\mathsf{min}}(\lambda), m(\lambda))$-PUF family with d_{min} being in the order of $o(\lambda/\log\lambda)$. We now determine the corresponding parameters for the fuzzy extractor as follows: Let $\ell(\lambda) := \lambda$ be the length parameter for value st. Let $\epsilon(\lambda)$ be a negligible function and let $t(\lambda) = d_{\mathsf{noise}}(\lambda)$. For each λ, let $(\mathsf{Gen}, \mathsf{Rep})$ be a $(m(\lambda), \ell(\lambda), t(\lambda), \epsilon(\lambda))$-fuzzy extractor. The metric space $\mathcal{M}$ is $\{0,1\}^{rg(\lambda)}$ with Hamming distance $\mathsf{dis}_{\mathsf{ham}}$. Note that such fuzzy extractors only exist if $rg(\lambda)$ and $m(\lambda)$ are sufficiently large. In order to achieve this, several PUFs can be combined. When combining two PUFs of the same family, rg gets doubled and so does m. Thus, if there are PUFs with $m(\lambda)$ being non-negligible they can be combined to a useful PUF-family—even if a PUF-family has *at least polynomial fraction* entropy, it still can be combined to obtain a good PUF-family with outputs, which, has high entropy of many bits. Thus, we may assume that the PUF has corresponding parameters.

Definition 3.5.2 (Matching Parameters). *If a PUF and a fuzzy extractor* (Gen, Rep) *satisfy the above requirements, then they are said to have* matching parameters.

If a PUF and a fuzzy extractor have matching parameters, then the properties well-spread domain, extraction independence and response consistency follow.

Well-Spread Domain: For all polynomials $p(\lambda)$ and all sets of challenges $c_1, ..., c_{p(\lambda)}$, the probability of a random challenge to be within distance smaller d_{min} of any of the c_k is negligible.

Extraction Independence: For all challenges $c_1, ..., c_{p(\lambda)}$, it holds that the PUF evaluation on a challenge c with $\mathsf{dis}(c_k, c) > d_{\mathsf{min}}$ for all $1 \leq k \leq p(\lambda)$ and subsequent application of Gen yields an almost uniform value st even for those who observe p.

Response Consistency: The fuzzy extractor helps to map two evaluations of the same PUF to the same random string, i.e., if PUF is measured on challenge c twice and returns r and r', then for $(st, p) \leftarrow \mathsf{Gen}(r)$, one has $st \leftarrow \mathsf{Rep}(r', p)$.

We now prove that a PUF and a fuzzy extractor with matching parameters have a well-spread domain and meet extraction independence as well as response consistency.

Consider the well-spread domain property: the number of challenges in $\{0,1\}^\lambda$ within distance smaller d_{min} of at least one of the c_k can be upper bounded by the number of elements in a ball of radius d_{min} multiplied with the number $p(\lambda)$ of challenges, i.e.,

$$p(\lambda) \sum_{i=1}^{d_{\mathsf{min}}(\lambda)} \binom{\lambda}{i} \leq p(\lambda)\lambda^{d_{\mathsf{min}}(\lambda)},$$

which is a negligible fraction of 2^λ, as the term

$$p(\lambda)\lambda^{d_{\mathsf{min}}(\lambda)}2^{-\lambda} = p(\lambda)2^{d_{\mathsf{min}}(\lambda)\log\lambda - \lambda}$$

is negligible if $d_{\mathsf{min}}(\lambda) = o(\lambda/\log\lambda)$.

For extraction independence, we observe that $H_\infty(\mathsf{PUF}(c)|\mathcal{C})$ is greater than $m(\lambda)$. Thus, evaluating a PUF on challenge c corresponds to drawing a random value r from a distribution on the metric space $\{0,1\}^{rg(\lambda)}$ which has entropy greater than $m(\lambda)$. Hence, the distribution of the random variable (st, p) defined by drawing a response r from $\mathsf{PUF}(c)$, conditioned on $(r_1, c_1), ..., (r_{p(\lambda)}, c_{p(\lambda)})$ and applying Gen to it, is statistically close to uniform, i.e., $\mathrm{SD}((st, p), (U_\lambda, p)) \leq \epsilon$ by definition of the fuzzy extractor.

Finally, for response consistency, we notice that $d_{\mathsf{min}}(\lambda) = t(\lambda)$. Thus, the bounded noise property of the PUF assures that two PUF evaluations yield responses r, r' with distance smaller than $t(\lambda)$. Thus, the properties of the fuzzy extractor assure that for $(st, p) \leftarrow \mathsf{Gen}(r)$, one has that $\mathsf{Rep}(p, r')$ outputs st.

4 PUF-based Authentication Protocols in the Stand-Alone Framework

The properties of a Physically Uncloneable Functions inspired researchers to build authentication protocols relying on challenge/response pairs [Škorić and Tuyls, 2007, Batina et al., 2007, Guajardo et al., 2007]. In these protocols, one party issues a challenge in form of a stimulus, which the other party has to answer by measuring its PUF. If this PUF response matches a pre-recorded response held at the issuing party, the challenging party is authenticated. If the number of CRPs of a PUF is large and an adversary cannot measure all PUF responses, it will most likely not be able to answer the challenge of the issuing party even if it had physical access to the PUF for a short time.

In this chapter, we show that current PUF-based authentication protocols are not fully secure in the above-mentioned attacker model where the attacker has physical control of the PUF and the corresponding reader during a short time. In particular, we revisit the authentication protocol of Tuyls et al. called TAP in the sequel [Škorić and Tuyls, 2007].

4.1 The TAP Protocol

Tuyls et al. [Škorić and Tuyls, 2007] proposed a token-based challenge/response protocol to authenticate a credit card against a central bank authority (server). For the sake of simplicity, we describe the details of the protocol in the two party scenario where a single smart card is authenticated to a central server: A PUF is embedded in a personalized smart card in a non-separable way. Since the PUF is uncloneable

and its response is unpredictable, the owner can use it to authenticate itself against a server. More precisely, the protocol consists of two phases, an enrollment phase and a verification phase. During the enrollment phase, the PUF is embedded inseparably in the smart card. Then the server challenges the PUF with several stimuli and stores the resulting challenge/response pairs in its database. Afterwards the smart card handed out to the owner to the person. During verification, the holder of the card inserts it into the ATM (card reader). The server chooses a random PUF challenge and sends it to the reader. The reader measures the PUF and returns the response back to the server. If the measured PUF response matches the recorded response in the server's database, the server is convinced that the reader has physical access to the PUF. Thus, the holder of the smart card is authenticated. Furthermore, both parties can derive a session key from the PUF output, which can subsequently be used to establish an authenticated communication channel between the server and the card reader.

The main goal of the TAP protocol is to assure that an adversary cannot impersonate the client. This protocol, however, is insecure once the adversary has physical access to the card. In fact, an adversary gets enough information to impersonate the server: Consider an adversary, who has access to the reader as well as to the PUF for a short time period. In this time it can read the card reader including all secret information. Moreover, the adversary can challenge the PUF with any stimuli at will in order to collect a number of challenge/response pairs. This information is enough to impersonate the server: The adversary engages in the TAP protocol and chooses a valid challenge of its collected CRPs and sends it to the card reader. The reader measures the output of the PUF for the given challenge and forwards the corresponding response to the adversary. The reader will be unaware that it has authenticated to (and established a key with) the adversary instead of the server, as it cannot distinguish a challenge chosen by the server from one issued by the attacker.

Recalling the TAP Protocol In the enrollment phase, the server issues a smart card including a PUF together with a current identifier id. It generates a set of random challenges $\mathcal{C} = (c'_1, \ldots, c'_\ell)$ for the PUF and measures for each c'_i the corresponding responses r'_i. Furthermore, for each challenge c'_i a random secret st'_i and a helper data p'_i is computed by solving $\mathsf{Gen}(r'_i) = (st'_i, p'_i)$. The rewritable non-volatile memory on the card stores the identifier id, a usage counter n, indicating how many times the authentication protocol ran, and the current hash value $m = H(rd)$, where rd is a

random string generated by the server and H is a one-way hash function. The central server holds in its database the card identifier id, both values n' and $m' = rd$, as well as a list of challenges $\mathcal{C}$ with the particular corresponding secrets $\mathcal{S}'$ and helper data values $\mathcal{P}$. Once a card is ready for use, it is initialized with $n' = n = 0$, $m' = rd$ and $m = H(rd)$.

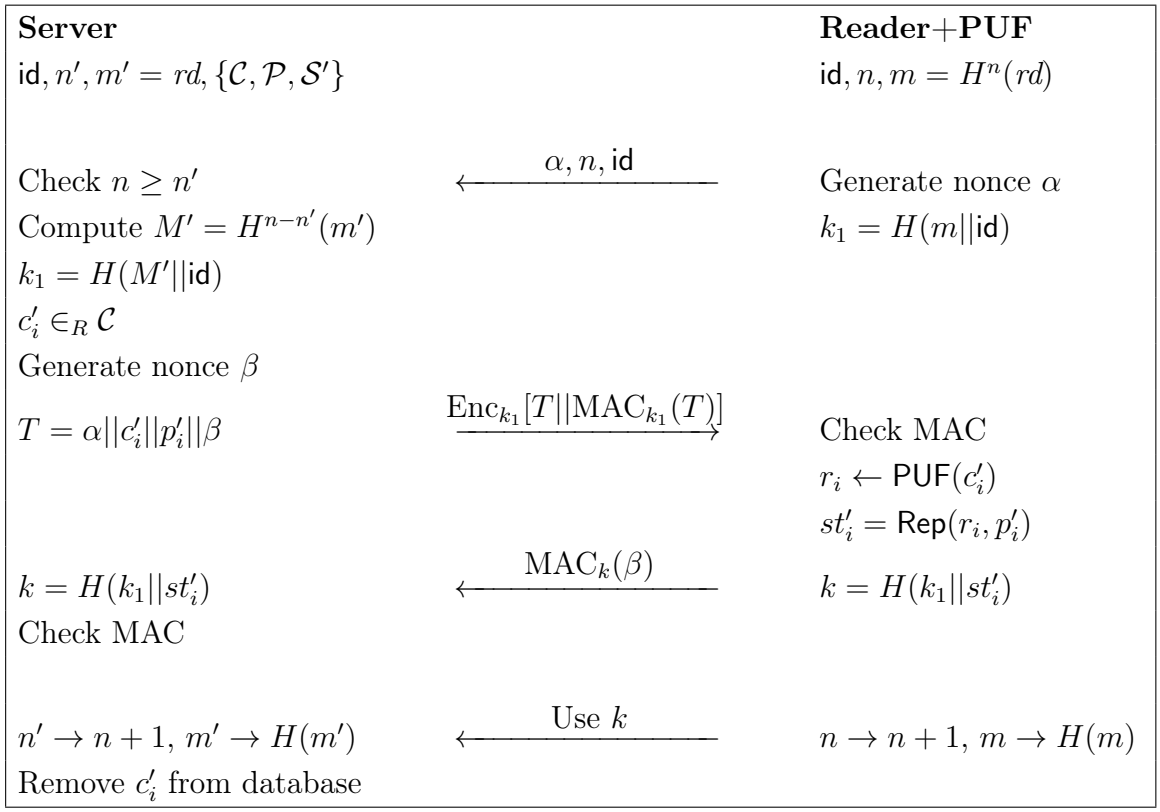


Figure 4.1: PUF-based authentication and key establishment protocol of [Škorić and Tuyls, 2007].

The PUF-based authentication and session key establishment protocol, depicted in Figure 4.1, consists of the following steps:

Reader+PUF The user inserts its card into a card reader. The reader sends an initialization message consisting of a nonce α, the usage counter n, and the identifier id to the server.

Server The server checks if $n \geq n'$. If this condition does not hold, then the server generates an error message and aborts. Otherwise, the server computes $M =$

$H^{n-n'}(m')$, where H^n denotes the n^{th} composition of H. Furthermore, the server computes a temporary key $k_1 = H(M||\mathsf{id})$. It then generates a nonce β, selects randomly a challenge $c'_i \in_R \mathcal{C}$ and computes the value $T = \alpha||c'_i||p'_i||\beta$ where p'_i is the helper data corresponding to c'_i. The server authenticates the quadruple $(\alpha, c'_i, p'_i, \beta)$ by computing a MAC with key k_1. It encrypts $T||\mathrm{MAC}_{k_1}(T)$ using k_1 and sends the result to the reader.

Reader+PUF The reader derives the temporary key by computing $k_1 = H(m||\mathsf{id})$, decrypts $\mathrm{Enc}_{k_1}[T||\mathrm{MAC}_{k_1}(T)]$ using k_1, and validates the MAC by using k_1. If the MAC is invalid or if the decrypted nonce α is wrong, then the reader aborts with an error message. Otherwise, the reader challenges the PUF with c'_i which produces a corresponding response r_i. The reader executes the helper data algorithm Rep with input r_i and helper data p'_i in order to obtain the secret st'_i. Now, the reader computes a session key $k = H(k_1||st'_i)$ based on the temporary key k_1 and the secret st'_i. Finally, the reader generates a MAC based on the nonce β using k and sends the MAC to the server.

Server The server computes its session key $k = H(k_1||st'_i)$ based on the temporary key k_1 as well as the secret st'_i corresponding to c'_i, which was generated in the enrollment phase and is stored in the database. Furthermore, the server verifies that the MAC is a valid tag on the nonce β (with respect to the secret derived from the response and the helper data). If the MAC is invalid, then the server aborts with an error message. Otherwise the server is convinced that the reader has physical access to the PUF and the holder of the smart card is authenticated.

Subsequently, both parties use the symmetric key k as a session key in order to set up a secure channel.

4.2 Limitations of the TAP Protocol

In the above protocol, an adversary who has access to the smart card and the PUF for a short period of time can impersonate the server, unless the communication between the bank and the reader is authenticated. Observe that even though the adversary gets access to the smart card, he does not learn the entire secret. The reason is that the shared secret is the set of challenges $\mathcal{C}$. In fact, Reader+PUF does not know $\mathcal{C}$, and

thus, the adversary in our attack chooses an arbitrary $c \notin \mathcal{C}$ and the Reader+PUF still answer.

We assume that the initial enrollment phase of the protocol is secure, ranging from the PUF fabrication up to the point where the user physically receives the smart card including the PUF. We now turn to the description of the adversary.

Let $\mathcal{A}$ be an adversary who has once access to the smart card including the PUF for a certain time. The algorithm $\mathcal{A}$ selects a small number of challenges $\mathcal{C}^*$ and measures the corresponding responses $\mathcal{R}^*$. Moreover, it reads the identifier id, the usage counter n and the current hash value m stored on the card memory. With this information, the adversary can impersonate the server in subsequent runs of the authentication protocol as follows:

Adversary The adversary computes the value $M^* = H^{n-n^*}(m)$, which is possible, because $\mathcal{A}$ obtained the usage counter n and the hash value $m = H(M)$ from the card memory. Now, $\mathcal{A}$ calculates $k_1^* = H(M^*||\mathsf{id})$ and generates a random nonce β^*. The adversary chooses a challenge $c_i^* \in \mathcal{C}^*$ and runs the algorithm Gen on input r_i in order to get helper data $p_i^* \in \mathcal{P}^*$ as well as a secret $st_i^* \in \mathcal{S}^*$. Furthermore, $\mathcal{A}$ produces a MAC on the quadruple $(\alpha, c_i^*, p_i^*, \beta^*)$ using the key k_1^*, encrypts the MAC with k_1^* and sends the resulting value $\mathrm{Enc}_{k_1^*}[(\alpha ||c_i^*||p_i^*||\beta^*)||\mathrm{MAC}_{k_1^*}(\alpha||c_i^*||p_i^*||\beta^*)]$ to the reader.

Reader+PUF The reader subsequently computes $k_1^* = H(m||\mathsf{id})$, decrypts $\mathrm{Enc}_{k_1^*}[(\alpha ||c_i^*||p_i^*||\beta^*)||\mathrm{MAC}_{k_1^*}(\alpha||c_i^*||p_i^*||\beta^*)]$ and checks whether the MAC is valid. Since the MAC and the decrypted nonce α are valid, the protocol does not abort with an error message. Consequently, the reader challenges the PUF with c_i^*, which produces a corresponding response r_i^*. Running the algorithm Rep on input r_i^* as well as p_i^*, the reader extracts the secret st_i^* from the output of the PUF. At last, the reader computes a MAC on the nonce β^* using the session key k. Afterwards, it sends the MAC to the adversary $\mathcal{A}$.

Adversary $\mathcal{A}$ computes its session key $k^* = H(k_1^*||st_i^*)$ by hashing the temporary key k_1^* and the secret st_i^*. Thus, the attack succeeded and the adversary is able to impersonate the server successfully. Moreover, the key k^* (respectively k) is a symmetric key established between the reader and the adversary.

The main reason why the attack works is that the adversary gets physical access to the smart card including the PUF at least once for a short period. During this time, the adversary measures the PUF and uses the obtained challenge/response pairs in subsequent authentication runs. Due to the symmetric nature of the protocol, the reader cannot decide whether a given challenge during a run of the protocol was initially measured by the server or subsequently by the adversary.

4.3 Using Bloom Filters and Hash Trees in AKE-Protocols

The above-mentioned weakness can be solved—without requiring an authenticated link between the server and reader—by storing a subset of "valid" challenges $\mathcal{V}$ initially measured by the server in the read-only memory on the smart card. However, storing all challenges is too expensive. Moreover, an adversary that reads the memory of the smart card would learn the subset of legal challenges. Thus, the set $\mathcal{V}$ has to be stored on the card in a compact form, which does not allow a computational bounded adversary to gain information on legal challenges. In this case impersonation can be prevented, as the adversary does not know the set of valid challenges and will most likely present an invalid challenge, which will not be accepted by the reader (this requires a strong PUF since the number of challenges must be large). We propose two solutions for compactly storing legal PUF challenges, one relies on Bloom Filters [Bloom, 1970] and the other one on hash trees [Merkle, 1980].

4.3.1 Bloom Filter

A Bloom Filter $\mathcal{B}$ is a probabilistic data structure that encodes a set of elements $\mathcal{X}$ into a ℓ bit array B in order to allow fast set membership tests [Bloom, 1970, Broder and Mitzenmacher, 2002, Mitzenmacher and Upfal, 2005]. The idea is to encode the elements by using several hash functions. The Bloom Filter $\mathcal{B}$ consists of ℓ bits $B[0], \ldots, B[\ell - 1]$, where all entries are initially set to 0, and a set $\mathcal{H}$ of b independently chosen hash functions $H_i : \{0,1\}^* \to \{0, \ldots, \ell - 1\}$. In order to encode a set $\mathcal{X} = \{x_1, \ldots, x_m\}$ into the Bloom Filter, the elements of $\mathcal{X}$ are added sequentially into

the array B according to the following rule: For each element $x \in \mathcal{X}$, we set $B[H_i(x)]$ to one for all $1 \leq i \leq b$.

The algorithm $\textsc{Check}(x', \mathcal{B}, \mathcal{H})$ verifies if a given element x' belongs to the Bloom Filter $\mathcal{B}$ and works as follows: $\textsc{Check}(\cdot)$ evaluates all b hash functions on x' and outputs 1 iff $B[H_i(x')] = 1$ for all $1 \leq i \leq b$. If one of the bits is set to 0, the algorithm outputs 0 since x' is clearly not in $\mathcal{B}$. Note that the length ℓ of the array B has to be chosen carefully (as well as the number of hash functions b). In case that many bits in B are set to 1, the probability that an arbitrary element $\hat{x}$ is recognized as a member of $\mathcal{X}$ increases. We call the event that an element $\hat{x} \notin \mathcal{X}$ is falsely recognized as member of $\mathcal{X}$ by the Bloom Filter, e.g., $\textsc{Check}(\hat{x}, \mathcal{B}, \mathcal{H}) = 1$, as a *false positive*. The parameters of the Bloom Filter have to be chosen in a way that makes this error very unlikely.

4.3.2 Hash Tree

A hash tree (or *Merkle tree*) $\mathcal{T}$ is a complete binary tree used to prove set membership [Merkle, 1980, Micali and Reyzin, 2001]. The hash tree consists of a root node v_{root}, several internal nodes v_i, and leaf nodes v_{leaf}. Each leaf node represents a data value d_i. Furthermore, each internal node stores a hash value of the concatenation of the values stored in its children. The hash values are computed with an collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^n$. Thus, if an internal node v_{i+1} has a left child $v_{i,0}$ storing the value $x_{i,0}$ and a right child $v_{i,1}$ storing the value $x_{i,1}$, then v_{i+1} stores the value $x_{i+1} \leftarrow H(x_{i,0}||x_{i,1})$. It is well known that it is, given a tree $\mathcal{T}$, infeasible to find another path that yields to the same root node v_{root}.

The algorithm $\textsc{Check}(x_r, d_i, \langle \tau_1, \ldots, \tau_t \rangle)$ verifies if a given element d_i belongs to the hash tree $\mathcal{T}$. Instead of exposing all leaf nodes, the algorithm is passed an *authentication path*, which consists of the hash values $\langle \tau_1, \ldots, \tau_t \rangle$ of the siblings of all nodes along the path from the leaf node v_{leaf} storing d_i to the root node v_{root}. Let x_i be the hash value of node v_i along the authentication path. We set $x_1 \leftarrow v_{\text{leaf}}$ and compute the remaining hash values $x_2, \ldots, x_t$ as follows: Let $i_1, \ldots, i_t$ be the binary representation of tree index i. If $i_j = 0$, we set $x_{j+1} \leftarrow H(x_j||\tau_j)$, and otherwise, if $i_j = 1$, we set $x_{j+1} \leftarrow H(\tau_j||x_j)$. The algorithm outputs 1 iff $x_{t+1} = x_{\text{root}}$, i.e., if the root hash matches, and 0 otherwise.

4.4 PUF-based AKE-Protocol based on Bloom Filters

The modified protocol, based on Bloom Filters, is depicted in Figure 4.2 and contains the following modifications: During the secure enrollment phase, the server computes

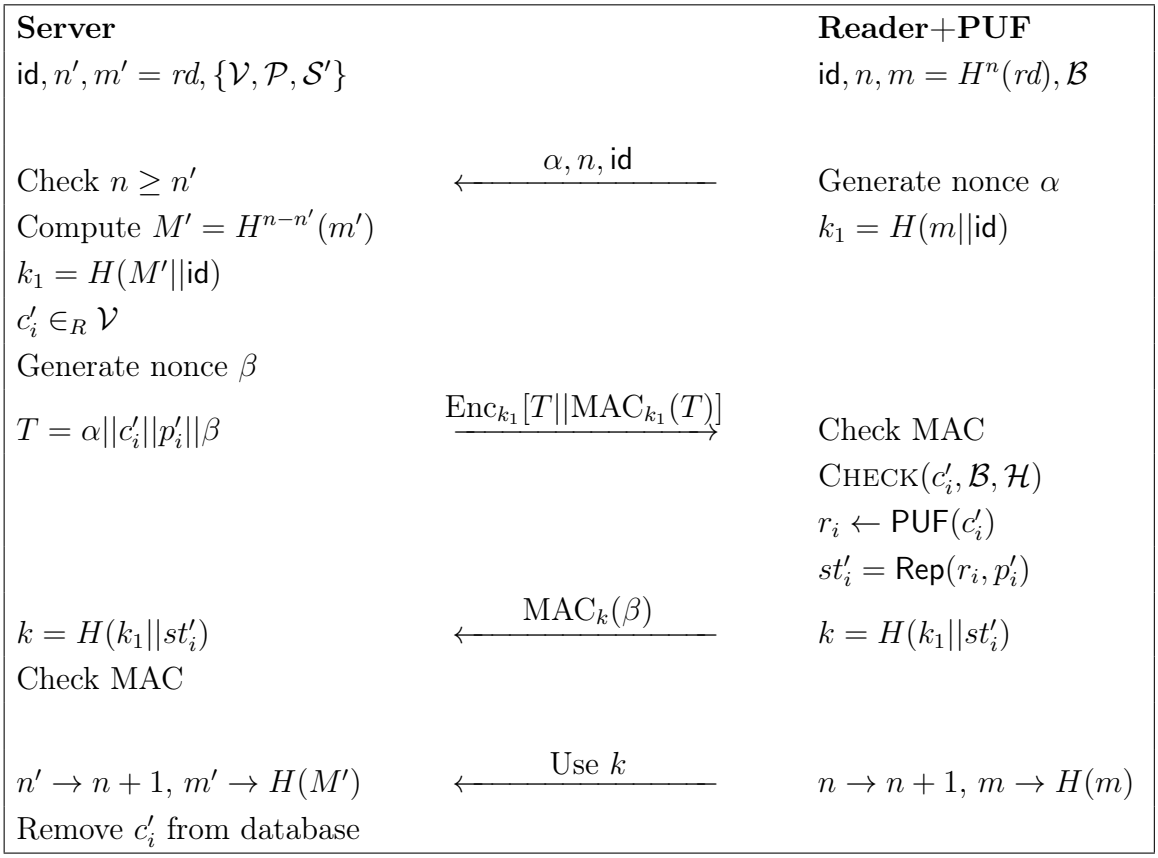


Figure 4.2: PUF-based authentication and key establishment based on Bloom Filters.

a subset of valid challenges $\mathcal{V} \subseteq \mathcal{C}$ by choosing a certain number of challenges $c'_i \in \mathcal{C}$ at random. Afterwards, the server stores the challenges c'_i of $\mathcal{V}$ with the corresponding responses in its database. Furthermore, the server computes a Bloom Filter $\mathcal{B}$ of size ℓ and stores all x challenges using b hash functions in $\mathcal{B}$. The rewritable non-volatile memory on the smart card stores the identifier id, the usage counter n, the current hash value m, and the Bloom Filter $\mathcal{B}$. If the reader receives a challenge c'_i, the reader verifies that it receives a challenge that was initially chosen by the server by checking whether $c'_i \in \mathcal{B}$, e.g., whether all array locations $B[H_j(c'_i)]$ for $1 \leq j \leq b$ are set to 1. If any

check fails, then clearly c_i' is not a member of $\mathcal{V}$ and the reader aborts. Otherwise, the reader follows the protocol steps as described in Section 4.1. This way, the reader can be sure that the server initially selected the challenge, unless an adversary succeeded in guessing a valid challenge.

4.4.1 Security Trade Off between Space and False Positives

In this section we take a closer look at the parameters of the Bloom Filter. Since Bloom Filters always have a false positive probability, which in our protocol results in the fact that the card reader accepts invalid challenges, the goal is to find a trade off between the space required to store the valid challenges on the card, the number of hash functions, and the false positive probability of the Bloom Filter. Recall that a false positive occurs if an element is accepted to be in the Bloom Filter, although it is not in the set [Bloom, 1970, Broder and Mitzenmacher, 2002, Mitzenmacher and Upfal, 2005]. The probability of a false positive f can be computed as

$$f = \left(1 - \left(1 - \frac{1}{\ell}\right)^{bx}\right)^{b} \approx (1 - e^{-bx/\ell})^{b},$$

where x is the number of challenges in $\mathcal{V}$, ℓ is the number of bits in the array of the Bloom Filter and b is the number of hash functions. We have to choose the parameters of a Bloom Filter appropriately in order to find a trade off between the computation time (corresponds to the number b of hash functions), the size (corresponds to the number ℓ of bits in the Bloom Filter array), and the probability of error (corresponds to the false positive probability f), which we will discuss in Section 4.5.2.

4.4.2 Analysis of the AKE-Protocol based on Bloom Filters

We analyze in this section the extended PUF-based protocol depicted in Figure 4.2. The main idea of the extension is to let both parties store a subset of valid challenges $\mathcal{V}$ of the challenge space $\mathcal{C}$. The reader then can check efficiently if a received challenge c is a member of $\mathcal{V}$. If $c \notin \mathcal{V}$, the reader aborts. Otherwise, if $c \in \mathcal{V}$, the reader follows the further protocol steps. Since only the server and the smart card know the subset $\mathcal{V}$, we prevent the impersonation of the server by an adversary. As a consequence, we

have a mutual authentication between the server and the holder of the smart card. Furthermore, the protocol is resistant against replay attacks because each PUF challenge is used only once. The protocol also retains backwards-security of the original TAP protocol.

Now, let us consider the subset $\mathcal{V}$ of valid challenges in more detail. Since we use a strong PUF we cannot draw conclusions about the elements of $\mathcal{V}$. Moreover, the subset $\mathcal{V}$ of valid challenges is encoded as a Bloom Filter in the read-only memory on the card. If an adversary obtains the ℓ-bit Bloom Filter and the b hash functions, it cannot deduce the x elements of $\mathcal{V}$. This follows from the fact that the hash functions are chosen independently However, there is still the probability that the adversary guesses a valid challenge, i.e., the adversary manages to find a challenge such that the testing algorithm of the Bloom Filter outputs 1. The probability that the adversary $\mathcal{A}$ guesses such an element (event win) can be upper bounded by the probability that it guesses a challenge being in the set $\mathcal{V}$ and the probability that an invalid challenge is accepted. This probability can be computed as follows:

$$\mathsf{Prob}[\mathcal{A}\,\mathsf{win}] \leq \frac{|\mathcal{V}|}{|\mathcal{C}|} + \left(1 - \left(1 - \frac{1}{\ell}\right)^{bx}\right)^{b} + \alpha.$$

Finally, the protocol is very efficient because it only requires symmetric cryptographic primitives.

4.5 PUF-based AKE-Protocol based on Hash Trees

In this section we propose an alternative solution based on hash tress. The benefit of this approach is that we only need to store a constant amount of data in the read-only memory of the smart card regardless of the number of elements of $\mathcal{V}$. Although this approach reduces storage, it induces an additional communication overhead.

The extended PUF-based authentication and key establishment protocol based on hash trees is depicted in Figure 4.3. During the secure enrollment phase, the server computes a subset of valid challenges $\mathcal{V} \subseteq \mathcal{C}$ by choosing uniformly a certain number of challenges of $\mathcal{C}$. Furthermore, let H be an one-way hash function. The server computes a hash tree $\mathcal{T}$, based on the elements of $\mathcal{V}$, as follows: Let us assume that the number of challenges

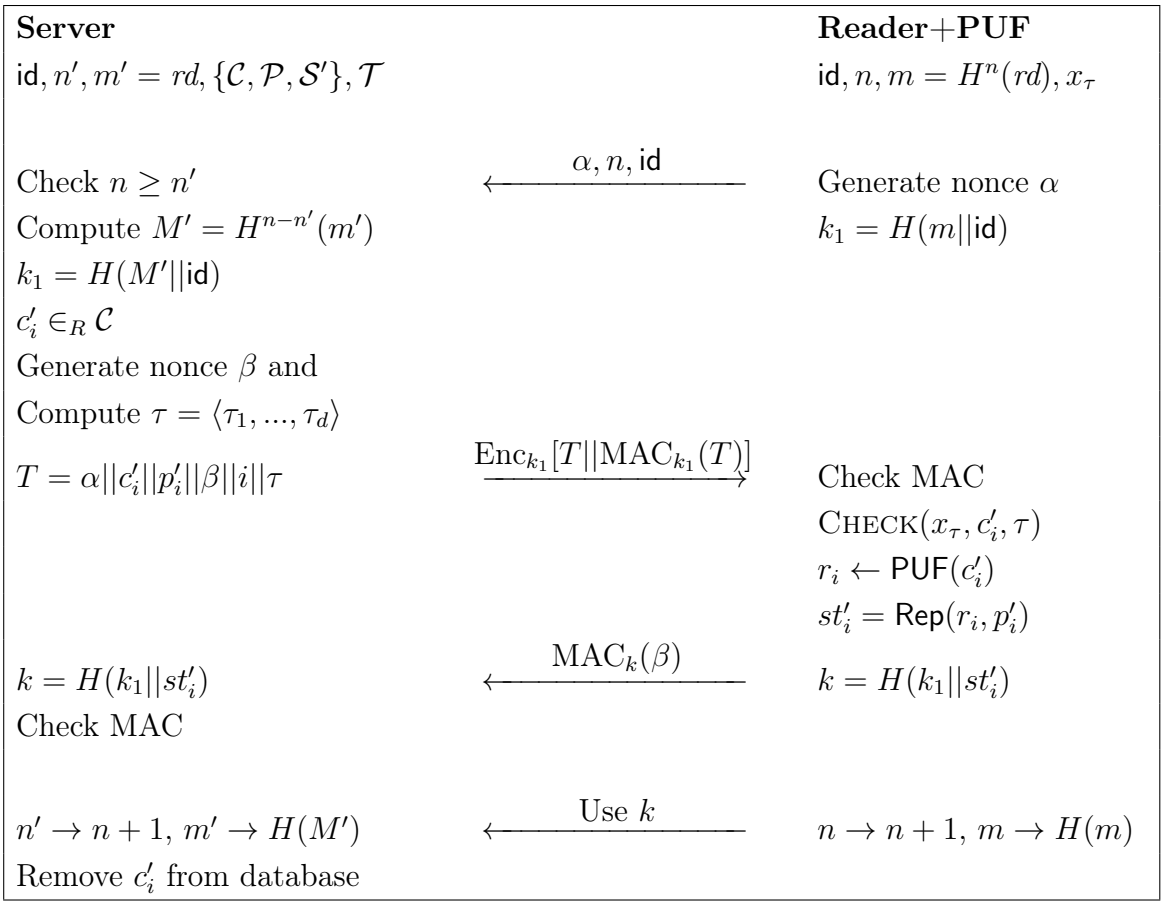


Figure 4.3: PUF-based authentication and key establishment protocol based on hash trees.

of the subset is a power of $|\mathcal{V}| = 2^\nu$. To authenticate the challenges $c_0, ..., c_{|\mathcal{V}|}$, the server places each challenge at the leaf nodes of a binary tree of depth $\mathcal{V}$. Moreover, the root node and each internal node of the binary tree are computed as hashes of its two child nodes (see Section 4.3.2). The root hash value x_τ is stored in the memory of the smart card, whereas the server stores the hash tree $\mathcal{T}$. To authenticate a challenge c'_i, the server discloses i, the corresponding path τ, between c'_i and the root node and all necessary sibling nodes, and sends the additional information to the reader. The reader now runs the algorithm $\textsc{Check}(x_\tau, c'_i, \tau)$ in order to verify the validity of the received path with its stored root value x_{root}. If the function returns 0, then the verification is not successful and the reader aborts with an error message. Otherwise, the reader knows that the challenge c'_i is a member of the set $\mathcal{V}$ of valid challenges. Subsequently, the reader follows the protocol steps as described in Section 4.1.

4.5.1 Communication Overhead of Hash Trees

The benefit of hash trees is that we only have to store the root value of the hash tree in the read-only memory of the smart card. Unfortunately, this solution involves additional communication overhead. The server has to send additional data besides the quadruple $(\alpha, c_i', p_i', \beta)$ in order to allow the reader to verify validity of c_i'. Let us assume that the hash function H maps its input c_i' to an λ-bit output v_i. Now the server has to transmit for each level of the tree the particular sibling node. Thus, the server has to send $\nu \cdot \lambda = \log_2 |\mathcal{V}| \cdot \lambda$ additional bits to the reader.

4.5.2 Analysis of the PUF-based AKE-Protocol based on Hash Trees

The security properties follow analogously to Section 4.4.2, except for the assumptions about the Bloom Filter. Here, the subset of valid challenges $\mathcal{V}$ is encoded as a hash tree. The security of hash trees relies on the one-way property and on the collision-resistance of the hash function H. Moreover, it is well known, that this data structure authenticates the elements in the hash tree, i.e., it is computationally infeasible to find a valid path given only the root node v_{root}. Finally, since only hash values are transmitted, and because the adversary is unable to invert the one-way hash function H, no information about the other challenges in the set (authenticated through the same tree) is leaked.

4.6 Bloom Filter vs. Hash Tree

Both of our above-mentioned approaches can be implemented on a smart card. Table 4.1 summarizes the storage and communication overhead of Bloom Filters and hash trees. Recall that $\mathcal{V}$ is the set of challenges, ℓ the length of the Bloom Filter, and λ the output length of the hash function.

Our solution based on Bloom Filter has the advantage that we do not need any additional communication overhead. However, in order to reduce the false-positives, the length ℓ has to be chosen appropriately.

	Storage on the Credit Card	Communication Overhead
Bloom Filter	ℓ	0
Hash tree	λ	$\lambda \cdot \log_2 \lvert\mathcal{V}\rvert$

Table 4.1: Comparison of the storage and communication overhead of Bloom Filters and hash trees.

On the other hand, if we want to optimize the storage capacity on the smart card, then the solution based on hash trees is the better choice because we only have to store the ℓ-bit hash value of the root node. Although this solution is very storage efficient, we get an addition communication overhead of $\lambda \cdot \log_2 |\mathcal{V}|$ bits.

5 Physically Uncloneable Functions in the Universally Composable Framework

In this section we formalize the notion of PUFs in the UC framework. Subsequently, we provide a UC secure oblivious transfer protocol, a commitment protocol that is based on any oblivious transfer protocol, and finally a UC secure key exchange protocol.

5.1 The UC Framework

In the following we briefly review the main concepts of Canetti's UC framework. A comprehensive description is out of scope of this dissertation and we refer the reader to [Canetti, 2001].

5.1.1 Intuition of the UC Framework

Consider a set of parties $P_1(x_1), P_2(x_2), ..., P_n(x_n)$ that wish to compute some function $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$ running a protocol Π, where y_i denotes the output of f for the $i-th$ party. Canetti's approach for defining security is based on the seminal paper of Goldreich, Micali, and Wigderson [Goldreich et al., 1987], which states: What does it mean for a given protocol Π to be secure for some task?

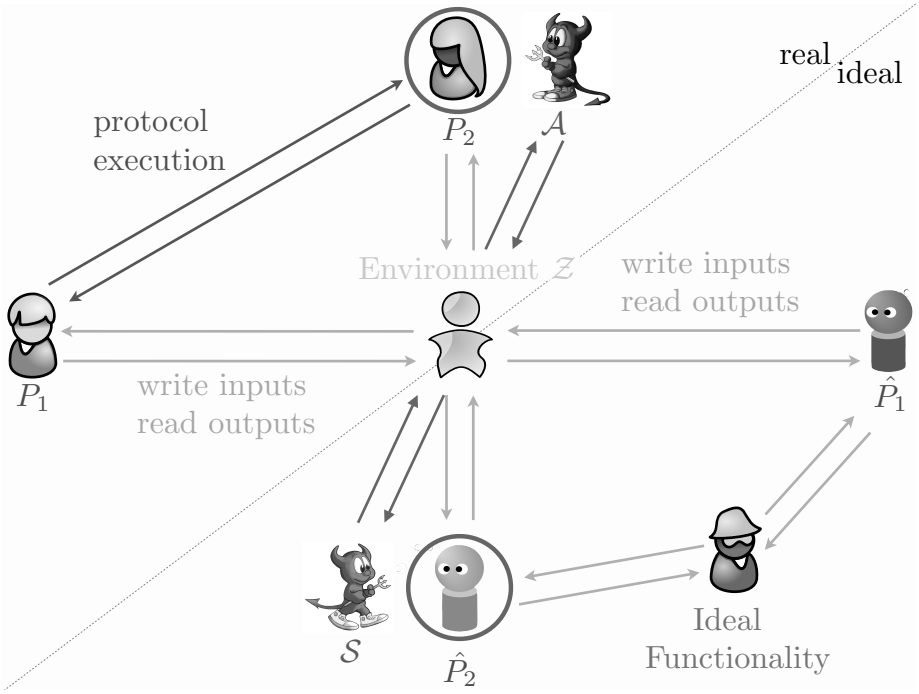
Micali and Rogaway answer this question by using the following approach: First one describes how the protocol Π should ideally computes the function f. This is done by letting the parties send their inputs $x_1, ..., x_n$ to some trusted party. This trusted party

then computes $(y_1, ..., y_n) \leftarrow f(x_1, ..., x_n)$ and sends y_i to the $i-th$ party. This is called the *ideal functionality*.

An interactive Turing Machine that interacts via authenticated communication channels with the other parties of the protocol describes this trusted entity. The second step is to show that Π realizes the ideal functionality. A protocol securely models an ideal functionality if any protocol attack in the real world (where no trusted party exists) can be carried out against the ideal functionality in the ideal world with the same outcomes. Naturally, in the ideal world an ideal adversary performs the attacks on the protocol. This ideal adversary can interact with the ideal functionality and it can corrupt honest parties by sending a corrupt command to the ideal functionality. At that time the adversary has full control over the corrupted party; note, further abilities of the adversary are explicitly specified by the ideal functionality.

5.1.2 The UC Framework

The UC framework describes two worlds, the *ideal* and the *real* world. Towards a third party, called the *environment machine* $\mathcal{Z}$, these two worlds should be indistinguishable:



Ideal World—with a Trusted Entity The ideal world consists of dummy parties, an ideal functionality $\mathcal{F}$, the environment $\mathcal{Z}$, and an ideal adversary, called the simulator $\mathcal{S}$. The parties are called dummy parties since (1) they forward any received input directly to the corresponding ideal functionality, and (2) they write every received value from the ideal functionality to their local output tape. The environment $\mathcal{Z}$ can set the parties' inputs and it can read their local output tape, but it cannot see nor tamper the communication with the ideal functionality. The simulator $\mathcal{S}$ can communicate with both the environment $\mathcal{Z}$ and the ideal functionality. Moreover, $\mathcal{S}$ can corrupt one or more of the parties and thus, it has the ability to see the party's input and any communication sent to a corrupted party and to control the output of the corrupted party.

Real World—Without any Trusted Entity The real world consists of parties, the environment $\mathcal{Z}$, and a real adversary $\mathcal{A}$. The adversary $\mathcal{A}$ can communicate with the environment $\mathcal{Z}$, and can corrupt one or more of the parties. Note that a corrupted party is completely controlled by the adversary, while an uncorrupted party proceeds according to the protocol specification. Similar to the ideal world, the environment $\mathcal{Z}$ can define the party's inputs and it can see its outputs, but it cannot see the internal communication (other than what is known to the adversary).

Now, a protocol securely realizes an ideal functionality $\mathcal{F}$ in the UC framework, if there exists an ideal-world adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ and for all adversary $\mathcal{A}$, it holds that the environment $\mathcal{Z}$ cannot distinguish between a run with the ideal functionality $\mathcal{F}$ and $\mathcal{S}$ in the ideal world or a run of the protocol with the adversary $\mathcal{A}$ in the real world.

In its original form the hybrid model supports the decomposition of cryptographic tasks into basic building blocks and to conclude security of protocols, which are composed out of such building blocks. Loosely speaking, Canetti's composition theorem—or, actually a corollary of a more general statement—says that, if a protocol $\pi^{\mathcal{F}}$ UC-securely realizes some functionality $\mathcal{G}$ in the hybrid world with efficient functionality $\mathcal{F}$, and some protocol ρ UC-securely realizes $\mathcal{F}$, then the composed protocol π^{ρ}, which invokes ρ whenever π would call $\mathcal{F}$, also UC-securely realizes $\mathcal{G}$.

5.2 Universally Composable Security and PUFs

Our ideal functionality covers different kinds of PUF technologies and comprises even PUFs with small input or output size (in which case unpredictability should be understood relative to the small output size). We note that for designing secure protocols, the intermediate access of the adversary also necessitates that the challenge space of the PUF is super-polynomial; else the adversary could clone the PUF easily. This domain requirement may currently not be true for all kinds of PUF technology; we comment on this later, see Section 6.

Furthermore, we consider a very minimalistic model, which basically says that only the party in possession of a PUF can evaluate it by sending some stimulus to the PUF and observing the output, and where learning outputs for some stimuli does not facilitate the task of predicting the function's output for other stimuli. Thus, our ideal functionality for PUFs only allows the party in possession to stimulate it in order to retrieve a response, thus ensuring restricted access. Uncloneability is enforced through unpredictability. Parties can hand over the PUF to other parties, but then we allow the adversary temporary access before the PUF reaches the recipient. This models the classical example of PUF-augmented cards, sent via postal service, which is read out before getting delivered. Moreover, one usually requires that tampering with PUFs can be detected easily, the idea being that a user does not use the PUF anymore after detecting it has been tampered with. Our UC-functionality will cover this property implicitly, as we permit the adversary black-box access to the PUF and the choice of delivering the PUF or not. Tampering with the PUF is treated as not delivering it. Furthermore, as in other works about hardware based tokens, we assume some kind of tamper-evidence in the sense that the receiver can later verify the authenticity and integrity of the PUF. We note that this need not be ensured by the PUF technology itself. One may also consider reliable delivery (in which case the adversary may have read-only access during the manufacturing process), see also [Ostrovsky et al., 2012].

Recall that the usage of hardware components in the UC context, especially of PUFs, causes several unpleasant side effects, though. At foremost, PUFs are not known to be implementable by probabilistic polynomial-time Turing machines; the manufacturing process seems to be inherently based on physical properties. Hence, while the claims in the hybrid model are technically sound, any realization in practice through actual

PUFs leaves a gap in the security claim of the composed protocol, as, strictly speaking, the composition theorem only applies to *probabilistic polynomial-time computable* functionalities $\mathcal{F}$. Fortunately, Canetti [Canetti, 2001] proves the composition theorem to hold for a broader class of interactive Turing machines, which also holds for PUFs.

The uninstantiability of PUFs through efficient algorithms causes another issue when it comes to complex cryptographic protocols. For any PUF-based protocol relying on further cryptographic assumptions like the hardness of computing discrete logarithms, the assumption would need to hold relative to the additional computational power given through PUFs. That is, the underlying problem must be hard to solve even for attackers with "more than probabilistic polynomial-time power". It is therefore advantageous to avoid additional cryptographic assumptions in protocols and provide solutions with statistical security.

5.2.1 Authenticated Communication in UC

The UC functionality $\mathcal{F}_{\mathsf{auth}}$ for authenticated message transmission [Canetti, 2001] is presented in Figure 5.1. The functionality prevents tampering and/or injecting messages. That is, a party P_j will receive a message msg from a party P_i only if P_i has sent msg to P_j. Each session will be defined by a session identifier $\mathtt{sid}$ and each execution will be accompanied by a unique sub session identifier $\mathtt{ssid}$.

$\mathcal{F}_{\mathsf{auth}}$ runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$.

- Whenever P_i writes $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ on $\mathcal{F}_{\mathsf{auth}}$'s input tape then $\mathcal{F}_{\mathsf{auth}}$ proceeds with the following actions:
  - $\star$ Store the tuple $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ and output $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ to the adversary $\mathcal{S}$.
- Whenever $\mathcal{S}$ writes $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ on $\mathcal{F}_{\mathsf{auth}}$'s input tape check if a tuple $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ is stored. If so, write $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ on P_j's communication input tape. Else, ignore the input.

Figure 5.1: The ideal functionality for message authentication adapted from [Canetti, 2001].

5.2.2 Modeling PUFs in UC

In the following we propose an ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ that will model PUFs. The functionality is presented in Figure 5.2 and handles the following operations:

1. A party P_i is allocated a PUF;
2. P_i can query the PUF;
3. P_i gives the PUF to another party P_j who can also query the device;
4. An adversary can query the PUF during transition.

The functionality $\mathcal{F}_{\mathsf{PUF}}$ maintains a list $\mathcal{L}$ of tuples $(\mathtt{sid}, P_i, \mathsf{id}, \tau)$ where $\mathtt{sid}$ is the (public) session identifier and id is the (internal) PUF-identifier, essentially describing the output distribution. Note that the PUF itself does not use $\mathtt{sid}$. The element $\tau \in \{\mathsf{trans}(P_j), \mathsf{notrans}\}$ denotes whether the PUF is in transition to P_j. For $\mathsf{trans}(P_j)$, indicating that the PUF is in transition to P_j, the adversary is able to query the PUF. In turn, if it is set to $\mathsf{notrans}$ then only the possessing party can query the PUF.

The PUF functionality $\mathcal{F}_{\mathsf{PUF}}$ is indexed by the PUF parameters $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$-PUF and gets the security parameter λ in unary encoding as additional input, see 3.3. It is required to satisfy the bounded noise property for $d_{\mathsf{noise}}(\lambda)$ and the unpredictability property for $(d_{\mathsf{min}}(\lambda), m(\lambda))$. This enforces that the outputs obey the basic entropic requirements of PUFs (analogously to the requirement for the random oracle functionality to produce random and independent outputs),see 3.3. We write $\mathcal{F}_{\mathsf{PUF}}$ and $\mathcal{F}_{\mathsf{PUF}}(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ interchangeably.

Also note that our definition requires that a PUF is somehow certified. That is, the adversary cannot replace a PUF sent to an honest party by a fake token including some "software emulation"; the adversary can only measure the PUF when in transition. The receiver can verify the constitution and authenticity of the received hardware.

Note, as explained in Section 5.2.3 we envision a non-programmable version of PUFs. The functionality above, if used in the standard way within the hybrid model, would be programmable, though, because the environment would not have direct access (even if the PUF is in possession of the adversary). One way to enforce non-programmability is to switch to the extended UC (EUC) model [Canetti et al., 2007] where all parties, including the environment, share the above functionality. The PUF could then also

$\mathcal{F}_{\mathsf{PUF}}(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$ receives as initial input a security parameter 1^λ and runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$.

- Whenever a party P_i writes $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$ then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathtt{sid}, *, *, *, *)$:
  - $\star$ If this is the case then turn into the waiting state.
  - $\star$ Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}(1^\lambda)$ from the PUF-family. The functionality $\mathcal{F}_{\mathsf{PUF}}$ puts the following tuple in $\mathcal{L}$: $(\mathtt{sid}, \mathsf{id}, P_i, *, \mathsf{notrans})$ and writes $(\mathsf{initialized}_{\mathsf{PUF}}, \mathtt{sid})$ on the communication input tape of P_i.
- Whenever a party P_i writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, P_i, c)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape then $\mathcal{F}_{\mathsf{PUF}}$ first checks, if there exists a tuple $(\mathtt{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ in $\mathcal{L}$:
  - $\star$ If this is not the case then turn into the waiting state.
  - $\star$ Else, run $r \leftarrow \mathsf{Eval}(1^\lambda, \mathsf{id}, c)$ and write $(\mathsf{eval'ed}_{\mathsf{PUF}}, \mathtt{sid}, c, r)$ on P_i's communication input tape.
- Whenever a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$ then $\mathcal{F}_{\mathsf{PUF}}$ first checks, if there exists a tuple $(\mathtt{sid}, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$:
  - $\star$ If this is not the case then turn into the waiting state.
  - $\star$ Else, modify the tuple $(\mathtt{sid}, \mathsf{id}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$. Write $\mathsf{invoke}_{\mathsf{PUF}}(\mathtt{sid}, P_i, P_j)$ on $\mathcal{S}$'s communication input tape to indicate that a $\mathsf{handover}_{\mathsf{PUF}}$ occurs between P_i and P_j.
- Whenever the adversary writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, \mathcal{S}, c)$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$ then $\mathcal{F}_{\mathsf{PUF}}$ first checks, if $\mathcal{L}$ contains a tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(*))$:
  - $\star$ If this is not the case then turn into the waiting state.
  - $\star$ Else, run $r \leftarrow \mathsf{Eval}(1^\lambda, \mathsf{id}, c)$ and return $(\mathsf{eval'ed}_{\mathsf{PUF}}, \mathtt{sid}, c, r)$ to $\mathcal{S}$.
- Whenever the adversary writes $(\mathsf{ready}_{\mathsf{PUF}}, \mathtt{sid}, \mathcal{S})$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape then $\mathcal{F}_{\mathsf{PUF}}$ searches for a tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$ in $\mathcal{L}$:
  - $\star$ If such a tuple does not exist then turn into the waiting state.
  - $\star$ Else, modify the tuple $(\mathtt{sid}, \mathsf{id}, \perp, \mathsf{trans}(P_j))$ to the updated tuple $(\mathtt{sid}, \mathsf{id}, P_i, \mathsf{notrans})$. Write the message $(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ on P_j's communication input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$.
- Whenever the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, $\mathcal{F}_{\mathsf{PUF}}$ checks if a tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathtt{sid}, P_i)$ has been stored. If so, it writes this tuple to the communication input tape of P_i. Else, $\mathcal{F}_{\mathsf{PUF}}$ turns into the waiting state.

Figure 5.2: The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for PUFs.

be evaluated by the environment in which case the simulator is informed about the challenge and response.

To simplify we linger within the basic UC framework and instead allow the environment to dispatch special PUF queries to the adversary/simulator. This query needs to be answered faithfully by forwarding it to a genuine PUF instance, and the response is handed back to the environment. Put differently, we put some restrictions on the how the simulator behaves, formally giving an UC-security proof, which would transfer to the EUC model.

5.2.3 Non-Programmability of PUFs

For PUFs, another aspect is the intrinsic *non-programmability* of these tokens: Even the manufacturer usually has no control over the functional behavior. Hence, the ability of the ideal-world simulator to adapt the outcome of a PUF measurement adaptively, as guaranteed when modeling the PUF through an ideal functionality in the hybrid world, appears to be exceedingly optimistic. A similar observation has been made by Nielsen [Nielsen, 2002] about the (non-)programmability of random oracles in the UC framework. Roughly, Nielsen takes away the ability of the simulator to program the random oracle by giving the environment direct access to the random oracle. To support the argument in favor of non-programmable PUFs we also note that for random oracles it is straightforward to program consistently given a partial view of the function for other values, namely, by providing independent random values; for PUFs this is less clear since one would need to take the (not necessarily efficiently computable) conditional distribution of the specific PUF type into account.

We adopt Nielsen's approach and augment the environment's ability by giving it also access to the concrete PUF instantiation used in a protocol. Unlike in the case of the publicly available random oracle, though, the environment can only access this PUF when it is in possession of the adversary, i.e., we assume that a PUF, once in possession of the user, can only be accessed by this user. This approach corresponds to the case that the user somewhat prevents further unauthorized access then. In a stronger version one could also allow further "uncontrolled" interaction between the environment, i.e., other protocols, and the PUF even then. This would somehow correspond to a permanently shared PUF functionality in the GUC model [Canetti et al., 2007]. However, many advantages of deploying PUFs for designing efficient protocols would then disappear. With the restriction on temporary access we can still devise efficient solutions, e.g., circumventing impossibility results for UC commitment schemes in the plain

model [Canetti and Fischlin, 2001] and for GUC commitment schemes in the common reference string model [Canetti et al., 2007].

Note, in this thesis the discussion about non-programmability of PUFs is out of scope and we refer the reader to the full version of [Brzuska et al., 2011b] for a comprehensive discussion.

5.3 PUF-based Protocols in the UC Framework

We finally exemplify the usability of our PUF modeling by presenting PUF-based protocols for three classical areas: oblivious transfer (OT), commitments, and key exchange. Our protocols are UC-secure in the hybrid world (where we grant the environment access to the PUF instantiation as described above), and typically require only a few operations besides PUF evaluations. In particular, all protocols require only sending one party a token in the first step. The protocols do not rely on additional cryptographic assumptions, except for authenticated channels.

Designing PUF-based protocols is not just a matter of adopting other token-based solutions. One reason is clearly the non-programmability property, which is usually not stipulated for other tokens (cf. [Katz, 2007, Goldwasser et al., 2008]). In fact, most protocols take advantage of the ability to adapt the token's outputs on the fly. But more importantly, the main difference between PUFs and other tokens is that PUFs are by nature even unpredictable for the manufacturer. It follows that only the party in possession of the PUF has full access to the secrets; other parties may only draw from a small set of previously sampled values. In comparison, for the wrapper tokens [Katz, 2007], for example, the creator still knows the program placed inside the token, and the token holder can fully access this program in a black-box way. Hence, both parties somehow share a complete view of the secret. For PUFs the situation is rather "asymmetric".

Designing an UC-secure commitment scheme with the help of our PUFs turns out to be quite challenging. The non-programmability of our PUFs inhibits equivocality, a property, which allows to adapt committed values appropriately, and which is usually required for such commitments [Canetti and Fischlin, 2001]. We therefore use our PUF-based oblivious transfer protocol to derive an UC-secure bit commitment scheme.

Interestingly, while the standard construction of commitment schemes out of OT, see [Crépeau, 1987] uses cut-and-choose techniques with a linear number of oblivious transfers, our transformation does not add any significant overhead. It only needs a single execution of the OT protocol and one extra message. We were not able to trace this idea back to any previous work. A noteworthy aspect is that, while our OT protocol only withstands static corruptions, our derived commitment scheme is secure in presence of adaptive corruptions. The reason is that for commitments, in contrast to OT, the receiver does not obtain any external input; the values used in the OT sub protocol are chosen internally. This facilitates the simulation of the receiver's side. Hence, if we use our transformation we derive an adaptively secure commitment protocol from a concrete statically secure OT protocol!

Finally, our key exchange protocol follows the folklore approach of using essentially the PUF to transport the key, only that our protocol is stated and formalized in the UC framework. That is, the sender samples some challenge-response pairs, sends the PUF, and later reveals a challenge to the receiver who recovers the image with the help of the PUF. Both parties use the images as the key, after applying a fuzzy extractor for error correction and smoothing the output. It is clear that for a key exchange protocol where only one party sends a PUF, some additional, one-sided authentication mechanism is required. Else the adversary with temporary access to the PUF could impersonate the honest sender.

All our protocols allow re-using the PUF for multiple executions. By the unpredictable nature of PUFs, however, it is clear that the number of executions must be fixed in advance and must be known to the parties: The sender, once having sent the PUF, cannot access the PUF anymore and must thus challenge the PUF before sufficiently often, unless the PUF is frequently exchanged or further PUF tokens are sent. An interesting feature of PUFs is that, unlike other hardware tokens (e.g., [Katz, 2007]), protocols using PUFs are automatically secure against reset attacks because they implement (noisy) functions.

5.4 Oblivious Transfer with PUFs

In a 1-out-of-2 oblivious transfer (OT) protocol the sender possesses two secrets $\mathsf{s}_0, \mathsf{s}_1$ and the receiver holds a selection bit $b \in \{0, 1\}$, thereby choosing one of the two secrets.

A 1-out-of-2 OT-protocol assures that at the end of the protocol execution, the receiver learns the secret s_b, but nothing about s_{1-b}, and the sender does not learn anything about the selection bit b.

Oblivious Transfer is a widely used cryptographic primitive for many cryptographic applications [Kilian, 1988, Crépeau, 1987, Goldreich et al., 1987]. However, in many of those applications a bottleneck of OT is the computational requirements since, for instance, several public key operations are necessary. We here show how to avoid the number of public key operations by adopting hardware. In the following, we recall the oblivious transfer ideal functionality and then provide a PUF-based oblivious transfer protocol.

As noted in the introduction, we envision a scenario in which the PUF is used multiple times. In the plain UC model, however, a fresh PUF would need to be sent for each OT execution. An alternative would be to switch to the joint-state theorem (JUC) [Canetti and Rabin, 2003] for the UC framework. However, JUC applies a transformation to the original protocol, and if a single session of a PUF protocol requires to hand over a PUF once, the JUC transformation would also require a handover per session. Nothing would be gained. Thus, we define and analyze multi-session protocols instead of the more common one-session protocols.

5.4.1 The Oblivious Transfer Ideal Functionality

1-out-of-2 oblivious transfer is an interaction between a sender P_i and a receiver P_j where the environment $\mathcal{Z}$ provides P_i with two inputs $\mathsf{s}_0, \mathsf{s}_1$ and P_j with an input bit b. As soon as both parties provided their inputs (and the simulator $\mathcal{S}$ allows delivery), the ideal functionality returns the secret s_b to the receiver. The ideal functionality for oblivious transfer $\mathcal{F}_{\mathsf{OT}}$ is given in Figure 5.3. We stress that this functionality only supports static corruption and can be used a bounded number of times, and only by the parties, which have exchanged the PUF.

5.4.2 PUF-based Oblivious Transfer Scheme

Our OT-protocol is based on a fuzzy extractor, a PUF, and an authenticated channel. The participating parties use the unpredictability of the PUF to ensure that the sender

$\mathcal{F}_{\mathsf{OT}}$ is parameterized by an integer N and receives as input a security parameter 1^λ, and runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$. The functionality initially sets $(n, S, R) = (1, \perp, \perp)$.

In the following, the functionality ignores any input if $n > N$, or if $n > 1$ and $(S, R) \neq (P_i, P_j)$ for the parties' identities (P_i, P_j) in the input. Else,

- Whenever P_i writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ with $\mathsf{s}_0, \mathsf{s}_1 \in \{0,1\}^\lambda \cup \{\perp\}$ on $\mathcal{F}_{\mathsf{OT}}$'s input tape, $\mathcal{F}_{\mathsf{OT}}$ stores $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ and writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to the communication input tape of $\mathcal{S}$. The functionality increments n to $n+1$ and stores $(S, R) = (P_i, P_j)$ if $n = 2$ now.
- Whenever P_j writes $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$, the functionality $\mathcal{F}_{\mathsf{OT}}$ stores this tuple and writes $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{S}$.
- When $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on $\mathcal{F}_{\mathsf{OT}}$'s input communication tape then $\mathcal{F}_{\mathsf{OT}}$ checks if tuples $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ and $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ have been stored. If so, write $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{s}_b)$ on the input communication tape of P_j.
- When $\mathcal{S}$ writes $(\mathsf{receipt}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on $\mathcal{F}_{\mathsf{OT}}$'s input communication tape then $\mathcal{F}_{\mathsf{OT}}$ checks if the tuple $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ has been stored. If so, write $(\mathsf{receipt}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input communication tape of P_i.

Figure 5.3: The ideal functionality for oblivious transfer adapted from [Canetti, 2001].

remains oblivious about the selection bit b and the receiver only learns the secret s_b. In a setup phase, the receiver issues a PUF, measures the responses for a set of randomly chosen challenges and stores the challenge/response pairs (c, r) in a list $\mathcal{L}$. The PUF is then given to the sender.

In Figure 5.4, we provide an oblivious transfer protocol. For simplicity of exposition, we use the following notation. For a possibly empty set $\mathcal{C}$ we let $\mathsf{dis}(c, \mathcal{C}) > d_{\mathsf{min}}$ denote the check that each element c_i in $\mathcal{C}$ satisfies the bound $\mathsf{dis}(c, c_i) > d_{\mathsf{min}}$. If not, we assume that the corresponding party aborts. Also, when interacting with the PUF (functionality), we simply write for example $r \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, c)$ to denote the fact that, for a call $(\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, c)$ the functionality has replied with $(\mathsf{eval'ed}_{\mathsf{PUF}}, \mathtt{sid}_0, c, r)$. Here, $\mathtt{sid}_0$ is the session identifier for $\mathcal{F}_{\mathsf{PUF}}$, as opposed to $\mathtt{sid}$ and $\mathtt{ssid}$ for the oblivious transfer protocol. The OT-protocol now involves the following executions:

First Message In the beginning of each protocol execution, the sender randomly generates two values x_0, x_1 and sends the values to the receiver.

Sender $\boldsymbol{P_i}$	session $\mathtt{sid}$	Receiver $\boldsymbol{P_j}$
		$(\mathsf{init}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, \lambda)$
		$k = 1, ..., N$: $c_k \leftarrow \{0,1\}^\lambda$
		$r_k \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, c_k)$
$\mathcal{C} := \emptyset$	$(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, P_j) \xleftarrow{\quad}$	$\mathcal{L} := (c_1, r_1, ..., c_\ell, r_\ell)$
		$\mathcal{C} := \emptyset$
Repeat at most N times with fresh $\mathtt{ssid}$		
Input:		Input:
$\mathsf{s}_0, \mathsf{s}_1 \in \{0,1\}^\lambda$, $\mathtt{sid}$		$b \in \{0,1\}$, $\mathtt{sid}$
$x_0, x_1 \xleftarrow{\$} \{0,1\}^\lambda$	$(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (x_0, x_1)) \xrightarrow{\quad}$	Draw $(c, r) \xleftarrow{\$} \mathcal{L}$
		$v := c \oplus x_b$
		$c' := c \oplus x_0 \oplus x_1$
$\mathsf{dis}(v \oplus x_0, \mathcal{C}) > d_{\mathsf{min}}$?	$(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, v) \xleftarrow{\quad}$	$\mathsf{dis}(c, \mathcal{C}) > d_{\mathsf{min}}$?
$\mathsf{dis}(v \oplus x_1, \mathcal{C}) > d_{\mathsf{min}}$?		$\mathsf{dis}(c', \mathcal{C}) > d_{\mathsf{min}}$?
Add $v \oplus x_0$, $v \oplus x_1$ to $\mathcal{C}$		Add c, c' to $\mathcal{C}$
$r'_0 \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, v \oplus x_0)$		Delete (c, r) from $\mathcal{L}$
$r'_1 \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}_0, P_i, v \oplus x_1)$		
$(st_0, p_0) \leftarrow \mathsf{Gen}(r'_0)$		
$(st_1, p_0) \leftarrow \mathsf{Gen}(r'_1)$		
$\mathsf{S}_0 := \mathsf{s}_0 \oplus st_0$,		
$\mathsf{S}_1 := \mathsf{s}_1 \oplus st_1$	$(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{S}_0, p_0, \mathsf{S}_1, p_1)) \xrightarrow{\quad}$	$st'_b \leftarrow \mathsf{Rep}(r, p_b)$
		$\mathsf{s}_b = \mathsf{S}_b \oplus st'_b$

Figure 5.4: Oblivious transfer Scheme with PUFs.

Second Message The receiver takes its secret bit b and picks a challenge/response pair (c, r) from $\mathcal{L}$. It sends the value $v = c \oplus x_b$ to the sender—note that the value x_b is statistically hidden, since c is a random string.

Third Message The sender generates two noisy responses $r'_0 \leftarrow \mathsf{PUF}(v \oplus x_0) = \mathsf{PUF}(c \oplus x_b \oplus x_0)$ and $r'_1 \leftarrow \mathsf{PUF}(v \oplus x_1) = \mathsf{PUF}(c \oplus x_b \oplus x_1)$ and ensures corresponding noise-free values by computing $(st_0, p_0) \leftarrow \mathsf{Gen}(r'_0)$ and $(st_1, p_1) \leftarrow \mathsf{Gen}(r'_1)$. Subsequently, the sender masks its two secrets s_0, s_1 by xoring st_0, st_1, and sends $\mathsf{S}_0 := \mathsf{s}_0 \oplus st_0, p_0, \mathsf{S}_1 := \mathsf{s}_1 \oplus st_1, p_1$ to the receiver.

Fourth Message The receiver runs $st_b \leftarrow \mathsf{Rep}(r, p_b)$ and simply computes s_b as $\mathsf{s}_b = \mathsf{S}_b \oplus st_b$. with $st = st_b$. The receiver cannot derive s_{1-b} as with overwhelming probability, he did not measure the PUF for challenge $v \oplus x_{1-b}$ or close challenge values, due to the well-spread domain property, see Section 3.5.1.

We note that the protocol does not achieve perfect completeness, i.e., executions between honest parties may fail. The probability for this is negligible, though. This follows straightforwardly again from the fact that the domain is well-spread: All (at most polynomial) challenges are independent random values such that one is within small distance of the others with negligible probability only. If all challenges are sufficiently far apart, the receiver always obtains the correct value.

We now sketch the security arguments for the OT-protocol in Figure 5.4. At the end of the OT protocol

1. a malicious sender learns nothing about the bit b and
2. a malicious receiver learns only the secret s_b and remains oblivious about s_{1-b}.

For case (1), the receiver chooses the challenge c at random. Thus, $v = c \oplus x_b$ hides x_b information-theoretically and thus also b. We now consider case (2). For simplicity, assume that $b = 0$. Then, the sender shall remain oblivious about any information about s_1. If st_1 looks uniform to the sender, then s_1 is information-theoretically hidden. If the fuzzy extractor and the PUF have matching parameters (see Definition 3.5.2), then with overwhelming probability this is the case, as—due to the well-spread domain property (see Section 3.5.1)—the probability that the receiver queried the PUF on values c_k with $\mathsf{dis}_{\mathsf{ham}}(c_k, v \oplus x_1) < d_{\mathsf{min}}$ is negligible, and the checks on the sender side about list $\mathcal{L}$ provided that the sender does not reveal PUF responses to critical challenges.

Theorem 5.4.1. *Assuming that* $(\mathsf{Gen}, \mathsf{Rep})$ *is a* (m, ℓ, t, ϵ)*-fuzzy generator, and* $\mathsf{PUF} = (\mathsf{Sample}, \mathsf{Eval})$ *is a PUF-family with matching parameters (see Definition 3.5.2), then protocol* PUFOT *securely realizes the functionality* $\mathcal{F}_{\mathsf{OT}}$ *in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model.*

Security holds in a statistical sense, i.e., the environment's views in the two worlds are statistically close. This remains true for unbounded algorithms $\mathcal{A}, \mathcal{S}$, and $\mathcal{Z}$, as long as the number of PUF evaluations is polynomial bounded.

Proof. As only static corruptions are considered, we can distinguish the simulation depending on the corrupted parties being involved. Our simulator $\mathcal{S}$ runs a black-box

simulation of adversary $\mathcal{A}$, emulating the communication of honest parties with the limited information provided in the ideal model, and such that $\mathcal{S}$ can use the adversary to reply to the environment. As explained, for non-programmability we consider restricted simulators, which faithfully initialize a PUF and allow the environment straight access when the PUF is in possession of the simulator. In the analysis below we analyze only a single execution; the argument extends straightforwardly to the at most N runs.

Both parties are honest. First, we consider the case when both parties are honest:

- Whenever $\mathcal{F}_{\mathsf{OT}}$ writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to the communication input tape of $\mathcal{S}$ in the ideal world, then this message indicates in the real world that $\mathcal{Z}$ wrote two secrets $(\mathsf{s}_0, \mathsf{s}_1)$ to the communication input tape of P_i. The simulator $\mathcal{S}$ now draws two random values $(\mathsf{s}'_0, \mathsf{s}'_1)$ as fake secrets and writes them to the local input tape of the simulated P_i in the ideal world. The simulator $\mathcal{S}$ then chooses two random values $x_0, x_1 \in \{0,1\}^\lambda$ as the real P_i and runs the P_i algorithm, which will write $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (x_0, x_1))$ to the simulated copy of $\mathcal{F}_{\mathsf{auth}}$.

- If at some point, $\mathcal{F}_{\mathsf{OT}}$ outputs $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, then $\mathcal{S}$ draws a random bit b and writes it to the local input tape of the simulated copy of the user P_j.

- All participants, i.e., the simulated parties, $\mathcal{F}_{\mathsf{PUF}}$, and $\mathcal{A}$ might write on the (simulated) input tapes of each other and are activated upon receiving input to run according to their program. As long as they produce no local output, $\mathcal{S}$ simply relays and activates its respective subroutines.

- If the receiver P_j produces the local output s_b, then $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$. If the sender P_i locally outputs $\mathsf{receipt}_{\mathsf{OT}}$, then $\mathcal{S}$ writes on $(\mathsf{receipt}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$ and is activated again to continue the simulation of the sender P_i.

In the setup phase, $\mathcal{A}$ and $\mathcal{Z}$ had access to the PUF used in the protocol and may make polynomially many measurements. We show that with high probability, for the adversary $\mathcal{A}$, the sequence of strings $(x_0, x_1, c \oplus x_b, \mathsf{s}_0 \oplus st_0, \mathsf{s}_1 \oplus st_1)$ looks like a random 5-tuple of strings. Towards this goal, we are going to show that with high probability, the five random variables corresponding to each of the entries are

almost uniformly and independently distributed. As dependence is a symmetric property, it suffices to show that no random variable depends on the previous ones: x_0 and x_1 are drawn independently at random; $c \oplus x_b$ is defined by drawing c at random and then xoring it to x_b; the distribution of $c \oplus x_b$ is independent from x_0, x_1 and b. The well-spread domain property (see Section 3.5.1) assures that with overwhelming probability, the adversary did not query the PUF on challenges closer than d_{min} from $x_0 \oplus c \oplus x_b$ or $x_1 \oplus c \oplus x_b$. Assume from now on that this is indeed the case. Then, due to the well-spread domain property (see Section 3.5.1), the first outputs of $\mathsf{Gen}(r'_0)$ and $\mathsf{Gen}(r'_1)$ are distributed almost according to U_ℓ. Even when observing the helper data, their statistical distance from U_ℓ is at most $\epsilon(\lambda)$ and, thus, their statistical distance from distributions corresponding other challenges (and other values $(x_0, x_1, c \oplus x_b)$) is at most $2\epsilon(\lambda)$ and thereby negligible, which concludes the analysis.

Receiver is corrupt. We next consider the case where the sender P_i is honest and the receiver P_j is dishonest. The simulator $\mathcal{S}$ observes the dishonest receivers' PUF queries (made by $\mathcal{A}$ and $\mathcal{Z}$) in the setup phase and stores the challenge-response pairs in a list $\mathcal{L}$. Those will later help the simulator to extract secret bits in the protocol execution. The simulator also keeps an initially empty list of challenge values $\mathcal{C}$.

Whenever $\mathcal{F}_{\mathsf{OT}}$ writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{S}$, then the simulator $\mathcal{S}$ draws a pair of random values (x_0, x_1) from $\{0,1\}^\lambda$ and sends $(\mathtt{sid}, P_i, P_j, (x_0, x_1))$ in the simulation. When $\mathcal{A}$ instructs the simulated P_j to send v to P_i as in the real world, simulator first checks if $\mathsf{dis}(v \oplus x_0, \mathcal{C}) > d_{\mathsf{min}}$ and $\mathsf{dis}(v \oplus x_1, \mathcal{C}) > d_{\mathsf{min}}$. If so, the simulator $\mathcal{S}$ checks for each $(c, r) \in \mathcal{L}$, if either the hamming distance of c and $v \oplus x_0$ is smaller than d_{min}, or if the hamming distance of c and $v \oplus x_1$ is smaller than d_{min}. With overwhelming probability, for a single challenge c, both cases cannot occur simultaneously, as $\mathsf{dis}(c, v \oplus x_0) < d_{\mathsf{min}}$ and $\mathsf{dis}(c, v \oplus x_1) < d_{\mathsf{min}}$ implies $\mathsf{dis}(v \oplus x_0, v \oplus x_1) < 2d_{\mathsf{min}}$ and $\mathsf{dis}(x_0, x_1) < 2d_{\mathsf{min}}$, but due to the well-spread domain property, the latter only occurs in negligibly many cases. We established that for a single challenge, only one of the statements can hold. We now show that with high probability, either there are no challenges in $\mathcal{L}$ close to $v \oplus x_0$, or there are none, which are close to $v \oplus x_1$. Assume, the adversary had non-negligible probability that there are c_0, c_1 in his list of previously made queries to the PUF, such that $\mathsf{dis}(c_0, v \oplus x_0) < d_{\mathsf{min}}$ and

$\mathsf{dis}(c_1, v \oplus x_1) < d_{\mathsf{min}}$. It follows that $\mathsf{dis}(c_0 \oplus x_0 \oplus x_1, c_1) < 2d_{\mathsf{min}}$, or, equivalently, that $\mathsf{dis}(x_0 \oplus x_1, c_0 \oplus c_1) < 2d_{\mathsf{min}}$. Now, $x := x_0 \oplus x_1$ is a random value, which, since $\mathcal{A}$ has at this point already made all queries, independent of any c-values, which $\mathcal{A}$ has evaluated the PUF for. Thus, if $\mathcal{A}$ has non-negligible probability of having chosen such two challenges c_0 and c_1, then this must hold for a non-negligible fraction of values $x \in \{0,1\}^\lambda$. Let $p(\lambda)$ the number of challenges, the adversary queried. Then, the sums of two different challenges c_i and c_j are at most $\binom{p(\lambda)}{2} = \frac{1}{2}p(\lambda)(p(\lambda)-1)$, which is a negligible fraction of 2^λ. If we multiply this number by the number of elements in a ball of radius $2d_{\mathsf{min}}$, then the property still holds, as $2d_{\mathsf{min}}$ is still in $o(\lambda/\log\lambda)$. Hence, $\mathcal{A}$ cannot cover a non-negligible number of values x.

If there only exist challenges c with $\mathsf{dis}_{\mathsf{ham}}(x_0, v \oplus c) < d_{\mathsf{min}}$, then set $b := 0$. If there only exist challenges c with $\mathsf{dis}_{\mathsf{ham}}(x_1, v \oplus c) < d_{\mathsf{min}}$, then set $b := 1$. If no such challenge exists, then draw $b \leftarrow \{0,1\}$ at random. The simulator $\mathcal{S}$ now sends $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_j, P_i, v)$ in the simulation. Since P_j is dishonest, $\mathcal{S}$ can now write $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, b)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$, and, upon receiving the message $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{OT}}$, which passes the message $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{s}_b)$ to the corrupt user P_j and consequently to the simulator $\mathcal{S}$, which uses this value to compute a simulated output for P_j, which is controlled by $\mathcal{A}$. The simulator $\mathcal{S}$ picks a random value $\mathsf{s}_{1-b} \leftarrow \{0,1\}^\lambda$. In response to the message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, v)$, simulator uses the PUF and Gen to determine st_0, p_0, st_1 and p_1. The simulator $\mathcal{S}$ then passes $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0 \oplus st_0, p_0, \mathsf{s}_1 \oplus st_1, p_1))$ to the simulated $\mathcal{F}_{\mathsf{auth}}$.

We now demonstrate that the joint view of $\mathcal{Z}$ and $\mathcal{A}$ in the protocol execution is indistinguishable from the joint view of $\mathcal{Z}$ and $\mathcal{S}$ (with the simulated $\mathcal{A}$) in the ideal world model. The only difference between the two executions is the final message $(\mathsf{s}_0 \oplus st_0, p_0, \mathsf{s}_1 \oplus st_1, p_1)$ received by P_j. The checks performed assure that $\mathcal{A}$ did not query the PUF on any challenge with distance smaller than d_{min} from $v \oplus x_{1-b}$. Thus, by response independence (see Section 3.5.2), st_{1-b} is quasi uniform even in the presence of p_{1-b}. Thus, from the point of view of $\mathcal{A}$, the value $(\mathsf{s}_{1-b} \oplus st_{1-b}, p_{1-b})$ is identically distributed for all values s_{1-b}.

Sender is corrupt. We finally assume that the sender P_i is dishonest and the receiver

P_j is honest. The simulator $\mathcal{S}$ will use its permanent PUF access to extract the secrets $(\mathsf{s}_0, \mathsf{s}_1)$ from the dishonest sender. We now describe the simulation in detail: The simulator $\mathcal{S}$ waits for two conditions to be satisfied in arbitrary order:

- $\mathcal{A}$ instructs the malicious party P_i to send $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, (x_0, x_1))$ in the simulation.
- The ideal functionality $\mathcal{F}_{\mathsf{OT}}$ writes $(\mathsf{choose}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{S}$.

If both conditions are fulfilled, then the simulator $\mathcal{S}$ draws a random string $v \leftarrow \{0,1\}^\lambda$ and returns it to P_i. When $\mathcal{A}$ instructs P_i to send $(\mathtt{sid}, \mathtt{ssid}, P_i, P_j, a_0, p_0, a_1, p_1)$ to P_j, then $\mathcal{S}$ extracts the secrets s_0, s_1 as follows: The simulator $\mathcal{S}$ evaluates the PUF on $v \oplus x_0$ and $v \oplus x_1$ to get responses r_0 and r_1. It then uses the helper data p_0, p_1 and Rep to obtain st_0 and st_1. If one of the two Rep evaluation fails, then the corresponding value $\mathsf{s}_0/\mathsf{s}_1$ is set to be the empty value $\perp$. Else, they are set to be $\mathsf{s}_0 := st_0 \oplus a_0$ and $\mathsf{s}_1 := st_1 \oplus a_1$. The $\mathcal{S}$ then writes $(\mathsf{send}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, (\mathsf{s}_0, \mathsf{s}_1))$ on the input tape of the ideal functionality $\mathcal{F}_{\mathsf{OT}}$ and also instructs it with the message $(\mathsf{deliver}_{\mathsf{OT}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$.

We now show that the simulation for the case that P_i is dishonest is indistinguishable from the real world execution. This is the case (1) since in both worlds the message received by P_i is a uniformly random string; and (2) since the output of P_j is the same in both worlds' executions. We elaborate on the latter: If P_j has input bit $b = 0$ then it would proceed as the simulator $\mathcal{S}$ did to compute s_0. If P_j has input bit $b = 1$ then it would compute s_1 in the same way as the simulator $\mathcal{S}$. □

5.4.3 Oblivious Transfer with Sender-PUF

Our OT-protocol requires the receiver to send a PUF to the sender. Sometimes it may be desirable to have the sender prepare the PUF, though. This can be achieved by switching the roles of the sender and the receiver via the protocol by Wolf and Wullschleger [Wolf and Wullschleger, 2006], but at the expense of having to run linear many OT executions for strings of length λ. This is unavoidable since the receiver in an OT-protocol just enters a bit such that, when acting as a sender, it can only transmit

a single bit. In this protocol the sender of the outer OT-protocol acts as a receiver in the inner OT-protocol, thus sending the PUF.

The protocol in [Wolf and Wullschleger, 2006] requires only a single round of additional communication. It is UC-secure in the $\mathcal{F}_{\mathsf{OT}}$-hybrid world and inherits the security properties (statistical vs. computational security, and adaptive vs. static corruptions). With a linear overhead [Brassard et al., 1986] and another extra round of communication one can then get an OT-protocol for strings, which is also UC-secure in the $\mathcal{F}_{\mathsf{OT}}$-hybrid world for bit-functionality $\mathcal{F}_{\mathsf{OT}}$. The final protocol is now an UC-secure OT-protocol for strings with linear many calls to $\mathcal{F}_{\mathsf{OT}}$, a few extra rounds, and inheriting all security characteristics from $\mathcal{F}_{\mathsf{OT}}$.

5.5 PUF-based Commitment Scheme

A commitment scheme is a two-party protocol between a sender and a receiver where the sender (also called committer) first sends a disguised version of the value to the receiver such that, later, only this value can be revealed. More precisely, a commitment scheme allows the committer to compute to a value msg a pair $(\mathrm{com}, \mathrm{decom})$ such that com reveals nothing about the value msg but using the pair $(\mathrm{com}, \mathrm{decom})$ one can open msg. Moreover it should be infeasible to find a value decom' such that $(\mathrm{com}, \mathrm{decom}')$ reveals $\mathsf{msg}' \neq \mathsf{msg}$.

5.5.1 The Commitment Scheme Ideal Functionality

In the UC world, the commitment scheme is realized by the (bounded) functionality $\mathcal{F}_{\mathsf{com}}$ as follows: $\mathcal{F}_{\mathsf{com}}$ receives an input $(\mathsf{commit}, \mathtt{sid}, \mathtt{ssid}, \mathsf{msg})$ from some committer P_i where msg is the value committed to. After verifying the validity of the session identifier $\mathtt{sid}$, $\mathcal{F}_{\mathsf{com}}$ records the value msg. Subsequently, the functionality lets both the receiver P_j and the adversary $\mathcal{S}$ know that the committer has committed to some value by computing a public delayed output $(\mathsf{receipt}, \mathtt{sid}, \mathtt{ssid})$ and sending it to P_j (this phase is called the commitment phase).

To initiate the decommitment phase, the committer P_i sends $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid})$ to the functionality $\mathcal{F}_{\mathsf{com}}$. Thereupon, $\mathcal{F}_{\mathsf{com}}$ checks if there exists a value msg; if so, the functionality computes a public delayed output $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid}, \mathsf{msg})$ and sends it to P_j. When the adversary corrupts the committer by sending $(\mathsf{corrupt} - \mathsf{committer}, \mathtt{sid}, \mathtt{ssid})$ to $\mathcal{F}_{\mathsf{com}}$, the functionality reveals the recorded value msg to the adversary $\mathcal{S}$. Furthermore, if the $\mathsf{receipt}$ value was not yet delivered to P_j, then $\mathcal{F}_{\mathsf{com}}$ allows the adversary modifying the committed value. This is in order to deal with adaptive corruptions. The ideal functionality for commitment schemes $\mathcal{F}_{\mathsf{com}}$ is given in Figure 5.5.

$\mathcal{F}_{\mathsf{com}}$ is parameterized by an integer N and runs with parties P_i, P_j, and adversary $\mathcal{S}$. It initially sets $(n, S, R) = (1, \perp, \perp)$.

The functionality ignores any commit-input if $n > N$, or if $n > 1$ and $(S, R) \neq (P_i, P_j)$ for the parties' identities in the input. Else,

- Upon receiving input $(\mathsf{commit}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \mathsf{msg})$ from party P_i, $\mathcal{F}_{\mathsf{com}}$ proceeds as follows:
  - ⋆ Records msg, generate a publicly delayed output $(\mathsf{receipt}, \mathtt{sid}, \mathtt{ssid})$, and send the output to P_j. Increment n to $n+1$ and store $(S, R) = (P_i, P_j)$ if now $n = 2$.
- Upon receiving input $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid})$ from party P_i, $\mathcal{F}_{\mathsf{com}}$ proceeds as follows:
  - ⋆ If a value msg is recorded, generate a publicly delayed output $(\mathsf{open}, \mathtt{sid}, \mathtt{ssid}, \mathsf{msg})$ and send it to P_j.
  - ⋆ Otherwise, do nothing.
- Upon receiving the input $(\mathsf{corrupt} - \mathsf{committer}, \mathtt{sid}, \mathtt{ssid})$ from the adversary $\mathcal{S}$, $\mathcal{F}_{\mathsf{com}}$ proceeds as follows:
  - ⋆ Send the value msg to $\mathcal{S}$.
  - ⋆ If $\mathcal{S}$ provides a value msg' and the $\mathsf{receipt}$ output was not yet written on P_j's tape, then $\mathcal{S}$ can change the recorded value to msg'.

Figure 5.5: The ideal functionality for commitment schemes adapted from [Canetti, 2001].

5.5.2 PUF-based Commitment Scheme

We now provide a *universal* transformation from OT-protocols to bit commitment schemes, which—to our knowledge—has not been considered so far. Previous transformations [Kilian, 1988, Crépeau, 1987] rely on cut-and-choose and require linear many executions of the OT-protocol. Our transformation only requires a single additional

message to be sent after executing the OT-protocol. The main idea of the protocol in Figure 5.6 is to inverse the roles of the sender and the receiver. The OT-protocol transfers two secrets, and the committer only learns one of them, namely the one corresponding to its secret bit b. This secret is then used to open the commitment.

As mentioned above the main idea is to inverse the roles of the sender and the receiver. The commitment protocol uses the OT-protocol as a building block or, more precisely, since we work in the UC framework, the corresponding ideal OT-functionality. Consider a commitment scheme with OT-sender P_i and OT-receiver P_j. Then, P_j is the committer and has a secret bit b, which it submits to the OT-functionality. The receiver P_i draws two sufficiently long random strings s_0 and s_1, which it submits to the OT-functionality. The OT-functionality then provides P_j with the secret s_b. This terminates the commitment phase.

In the opening phase, the committer P_j sends the pair (b, s_b). The receiver P_i then checks whether s_b matches the b-th secret. The protocol is binding, as the OT-functionality does not allow to modify the secret bit b and the secret s_{1-b} is statistically hidden from P_i. Thus, P_i can determine s_{1-b} only with negligible probability. The protocol is hiding, as the OT-functionality does not reveal information about the bit b to P_i.

We now provide a formal description and analysis of the above transformation:

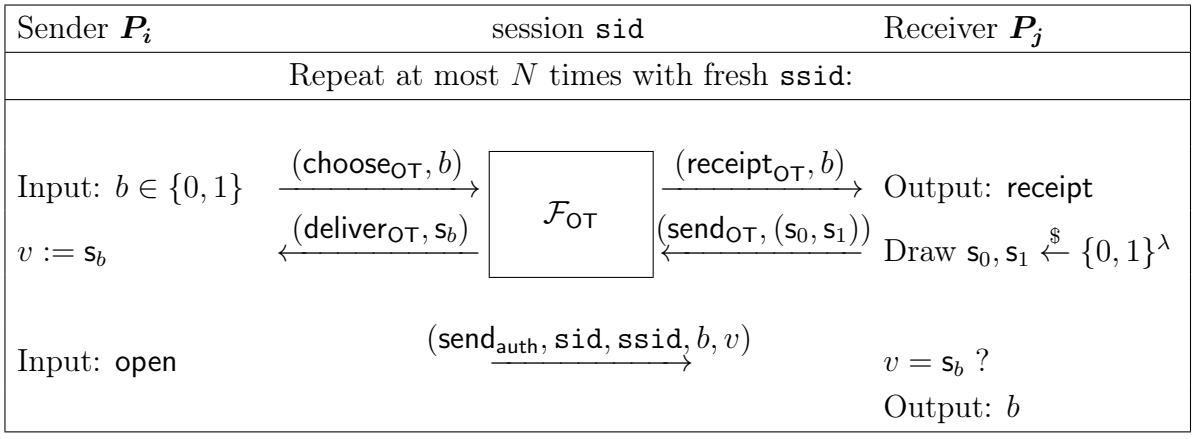


Figure 5.6: Commitment scheme with $\mathcal{F}_{\mathsf{OT}}$.

Theorem 5.5.1. *The commitment protocol in Figure 5.6 securely UC-realizes the functionality $\mathcal{F}_{\mathsf{com}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.*

If functionality $\mathcal{F}_{\mathsf{OT}}$ is replaced by some OT-protocol, then the derived commitment protocol basically inherits the characteristics of the OT-protocol. That is, it is secure against adaptive corruptions if OT is, and it is statistically secure if OT is. Remarkably, we show in the next section that our PUF-based OT-protocol, while being only statically secure, makes the commitment scheme even adaptively secure.

We merely provide a proof sketch for Theorem 5.5.1 here. Note that in the case where both users are honest, only the modeling of the final message needs to be taken into consideration. The simulator learns the secret bit b from the commitment functionality $\mathcal{F}_{\mathsf{com}}$. It then draws a random string v from $\{0,1\}^{\lambda}$ and sends $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, b, v)$. If the receiver is dishonest, then it provides $\mathcal{F}_{\mathsf{OT}}$ with two secrets $\mathsf{s}_0, \mathsf{s}_1$. The simulator lets the sender provide a random bit b' to the simulated $\mathcal{F}_{\mathsf{OT}}$. It receives back the secret $\mathsf{s}_{b'}$. In the opening phase, $\mathcal{S}$ learns the (real) secret bit b from the commitment functionality and simulates the final protocol message as $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, b, \mathsf{s}_b)$. If the sender is corrupt then it provides the simulated $\mathcal{F}_{\mathsf{OT}}$ with a secret bit b. The simulator creates two random strings $\mathsf{s}_0, \mathsf{s}_1$ and passes them to the simulated $\mathcal{F}_{\mathsf{OT}}$, which passes s_b to the receiver. The simulator commits to the sender's bit b in the ideal world. If the sender sends a message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, b, v)$ then $\mathcal{S}$ checks if $\mathsf{s}_b = v$. If so, it instructs $\mathcal{F}_{\mathsf{com}}$ to open the commitment.

All simulations are perfect.

5.5.3 Adaptively Secure Commitments

Consider the concrete commitment protocol where we plug in our OT-protocol from the previous section into the abstract scheme above (and work in the $\mathcal{F}_{\mathsf{PUF}}$-hybrid model instead of the $\mathcal{F}_{\mathsf{OT}}$-hybrid model then), then we observe the following: In the commitment phase (i.e., the OT-phase), the message sent by the OT-receiver (=commitment sender) is statistically independent from its secret input: the OT-receiver merely sends a single uniformly random message.

For the OT-sender, this is not the case: When having access to the PUF, one can extract both secrets from the mere transcript of the protocol. This enables the simulator $\mathcal{S}$ to derive both secrets from the protocol, as it accesses the PUF. It can thus provide the simulated committer with open messages for both bit values. As the remaining part

of the committer's state merely consists in challenge-response-pairs, the simulator can thus provide genuine internal state.

5.6 Key Exchange with PUFs

In a key exchange (ke) protocol two parties interact over an insecure network to establish a common secret key κ. This common secret key can then be used to build a secure channel or to ensure confidentiality of transmitted data.

5.6.1 The Key Exchange Ideal Functionality

The main idea of the key exchange ideal functionality $\mathcal{F}_{\mathsf{ke}}$ is the following: if both parties are honest, the functionality provides them with a common random value, which is invisible to the adversary. If one of them is corrupted, though, the adversary determines the session key entirely thus modeling the participation of a corrupted party. The definition of the key exchange functionality $\mathcal{F}_{\mathsf{ke}}$ is depicted in Figure 5.7 adapted from [Canetti and Krawczyk, 2002].

5.6.2 Minimal Requirements

We present a key exchange protocol in Section 5.6.3, which sends a PUF in a setup phase. Afterwards, a single message per protocol execution is sent via a unidirectional authenticated channel. As mentioned in the introduction, it is desirable to circumvent the use of complexity-theoretic assumptions. However, for practical reasons, PUF transfers should also be minimized. If only a single PUF transfer occurs, then the assumption of a unidirectional authenticated channel cannot be dropped: The sender of the PUF measured the PUF several times and sent it to the receiver. The adversary can query the PUF during its transition. If the sender does not have any further secret information for authentication, then the adversary can carry out the same computations as the sender. Thus, without an additional authenticated channel, the protocol cannot be secure against impersonation attacks. In the following, we use the standard bidirectional $\mathcal{F}_{\mathsf{auth}}$ functionality. Deriving corresponding unidirectional definitions is straightforward.

$\mathcal{F}_{\mathsf{ke}}$ is parameterized by an integer N and receives as input a security parameter 1^λ, and runs with parties $P_1, ..., P_n$ and adversary $\mathcal{S}$. $\mathcal{F}_{\mathsf{ke}}$ obtains a list of corrupt parties. It initially sets $(n, S, R) = (1, \perp, \perp)$.

Ignore any $\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}$-input if $n > N$, or if $n > 1$ and $(S, R) \neq (P_i, P_j)$ for the parties' identities in the input. Else,

- $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ is written on $\mathcal{F}_{\mathsf{ke}}$'s input tape by a party P_i. Then $\mathcal{F}_{\mathsf{ke}}$ stores the tuple $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ (and refuses if there already is a tuple $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_j, P_i)$ or a tuple $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$). $\mathcal{F}_{\mathsf{ke}}$ outputs $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to the adversary $\mathcal{S}$. If both users are honest then draw a random value κ from $\{0,1\}^\lambda$ and store the messages $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ and $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_j)$. Increment n to $n + 1$.
- When $\mathcal{S}$ writes $(\mathsf{choose} - \mathsf{value}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, \kappa)$ on $\mathcal{F}_{\mathsf{ke}}$'s input tape then check whether there is a message $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ or a message $(\mathsf{establish} - \mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ and whether at least one of the users P_i and P_j is corrupt. If so, store the messages $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ and $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_j)$.
- $\mathcal{S}$ writes $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i)$ on $\mathcal{F}_{\mathsf{ke}}$'s input communication tape. Check if a tuple $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ is stored. If so, write $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$ to P_i's input tape and delete $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, \kappa, P_i)$. Else, do nothing.

Figure 5.7: The key exchange ideal functionality adapted from [Canetti and Krawczyk, 2002].

5.6.3 PUF-based Key Exchange Scheme

Intuitively, our key exchange protocol proceeds as follows. In an enrollment phase, a server issues a PUF, measures for a set of randomly chosen challenges the corresponding responses, and finally ensures a noisy-free PUF measurement by generating for each response r a fuzzy extractor secret st from a set of random secrets as well as a corresponding helper data p. The server then sends the PUF to the client. Upon finishing the enrollment phase the server broadcasts a randomly chosen challenge c including its helper data p to the client and sets $\kappa = st$ to obtain the protocol key. The client evaluates the PUF on the challenge c, computes the corresponding fuzzy secret st due to the helper data p, and obtains the protocol key by setting $\kappa = st$. Consequently, both parties use the fuzzy extractor secret st as their common protocol key κ.

Theorem 5.6.1. *Protocol* PUFKE *securely realizes functionality* $\mathcal{F}_{\mathsf{ke}}$ *in the* $\mathcal{F}_{\mathsf{PUF}}$*-hybrid model.*

Server $\boldsymbol{P_i}$		Client $\boldsymbol{P_j}$
$(\mathsf{init}_{\mathsf{PUF}}, \mathtt{sid}, P_i, \lambda)$ Repeat N times: $r \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, P_i, c)$ $(st, p) \leftarrow \mathsf{Gen}(r)$ add (c, r, st, p) to $\mathcal{L}$	$(\mathsf{handover}_{\mathsf{PUF}}, \mathtt{sid}, P_i, P_j) \longrightarrow$	
Repeat at most N times		
pick $(c, r, st, p) \leftarrow \mathcal{L}$ remove the entry from $\mathcal{L}$ Output: $\kappa = st$	$(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, P_j, (c, p)) \longrightarrow$	$r' \leftarrow (\mathsf{eval}_{\mathsf{PUF}}, \mathtt{sid}, P_j, c)$ $st \leftarrow \mathsf{Rep}(r', p)$ Output: $\kappa = st$

Figure 5.8: PUF-based key exchange scheme.

Proof. Throughout all simulations, queries to PUFs are genuinely relayed to the $\mathcal{S}$'s PUF functionality, and the PUF's answer is honestly delivered to the querying party.

Both parties are honest. Whenever the key exchange functionality $\mathcal{F}_{\mathsf{ke}}$ sends message $(\mathsf{establish-session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ for the first time, the $\mathcal{S}$ initiates a PUF and queries it on N random challenges $c_1, ..., c_N$ to obtain responses $r_1, ..., r_N$. It then computes $(st_i, p_i) \leftarrow \mathsf{Gen}(r_i)$, and stores all tuples (c_i, r_i, st_i, p_i) in a list $\mathcal{L}$. The simulator $\mathcal{S}$ then simulates a $\mathsf{handover}_{\mathsf{PUF}}$ and lets the adversary query the PUF, until the adversary terminates the transition phase. $\mathcal{S}$ sets the counter n to 1 and continues with the simulation of the first session. This concludes the simulation of the setup phase.

On receiving a message $(\mathsf{establish-session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, the simulator $\mathcal{S}$ increases the counter n by one and sends $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i)$ to $\mathcal{F}_{\mathsf{ke}}$. The party P_i then writes the key on its local output tape and $\mathcal{S}$ is activated again. It simulates that P_i sends the message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, (c_n, p_n))$. When the adversary instructs to deliver the latter message to P_j, then the simulator $\mathcal{S}$ sends $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{ke}}$.

Analysis of the simulation: Due to the well-spread domain property, with overwhelming probability, the adversary did not query the PUF on values closer than d_{min} to one of the challenges c_i. By response independence, the message (c_n, p_n) is then statistically independent from the value st. Hence, the simulation is sound.

Receiver is corrupt. The simulation of the setup phase is as in the honest case.

On receiving a message $(\mathsf{establish}-\mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$, the simulator $\mathcal{S}$ increases the counter n by one and writes $(\mathsf{choose}-\mathsf{value}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, st_n)$ on $\mathcal{F}_{\mathsf{ke}}$'s input tape. It is activated again and writes $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i)$ to $\mathcal{F}_{\mathsf{ke}}$. The party P_i then writes the key on its local output tape and $\mathcal{S}$ is activated again. It simulates that P_i sends $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, (c_n, p_n))$. When the adversary instructs to deliver the latter message to P_j, then the simulator $\mathcal{S}$ sends $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{ke}}$. The simulation is perfect.

Sender is corrupt. The $\mathcal{S}$ allows the malicious user P_i to instantiate an arbitrary number of PUFs. The receiver only accepts a single $\mathsf{handover}_{\mathsf{PUF}}$. When the adversary instructs to deliver a message $(\mathsf{send}_{\mathsf{auth}}, \mathtt{sid}, \mathtt{ssid}, (c, p))$ to P_j, then the $\mathcal{S}$ evaluates the PUF on c to obtain response r and computes $st \leftarrow \mathsf{Rep}(r, p)$. The $\mathcal{S}$ lets the corrupt dummy user P_i write the message $(\mathsf{establish}-\mathsf{session}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j)$ on the input tape of $\mathcal{F}_{\mathsf{ke}}$ and subsequently passes the messages $(\mathsf{choose}-\mathsf{value}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_i, P_j, st)$ and $(\mathsf{deliver}_{\mathsf{ke}}, \mathtt{sid}, \mathtt{ssid}, P_j)$ to $\mathcal{F}_{\mathsf{ke}}$. The simulation is perfect. □

6 Conclusion

In this thesis, we presented different approaches to build cryptographic primitives and protocols based on Physically Uncloneable Functions. After analyzing existing PUF implementations and applications, we cryptographically formalized PUFs by proposing an appropriate PUF model.

Moreover, we investigated the security of PUF-based protocols in the stand-alone setting. Our results showed that known constructions are not fully secure if attackers have raw access to the PUF for a short period of time. We therefore proposed a new, stronger, and more realistic attacker model. Subsequently, we suggested two constructions of authentication protocols, which are secure against physical attackers in the new model and which only need symmetric primitives.

Finally, we analyzed PUFs in the UC framework for the first time. It turns out that designing PUF-based protocols is fundamentally different from designing protocols for other hardware tokens: One reason is the non-programmability of PUFs. Another reason is that the functional behavior a PUF is unpredictable even for its creator, which causes an asymmetric situation in which only the party in possession of the PUF has full access to the secrets. Using our UC notion of PUFs, we then devised efficient UC-secure protocols for basic tasks like oblivious transfer, commitments, and key exchange.

Future Directions Although the idea of Physically Uncloneable Functions has already existed for a couple of years (since 2001), it is still a young research field in the context of physical cryptography and security. This dissertation provides a solid foundation for future work in the field of PUFs. One open problem in the area is the following:

Our modeling of PUFs in the UC case is comprehensive enough to cover different PUF types which are based on different technologies. However, our UC protocols withstanding temporary adversarial access require that the challenge space of the PUF is super-polynomial. If this would be not the case the adversary could perform a full readout of the PUF. From a technological point of view, currently only optical PUFs satisfy this property. To broaden the set of admissible PUFs one could develop methods to enlarge the challenge space for PUFs, either by technological means or algorithmically via domain extensions. In this case, our ideas can be applied as well, but at the cost of efficiency, essentially increasing the number of PUF tokens to be used.

Bibliography

[Armknecht et al., 2011] Armknecht, F., Maes, R., Sadeghi, A.-R., Standaert, F.-X., and Wachsmann, C. (2011). A formal foundation for the security features of physical functions. In *IEEE Symposium on Security and Privacy*, pages 397–412. IEEE Computer Society.

[Armknecht et al., 2009] Armknecht, F., Maes, R., Sadeghi, A.-R., Sunar, B., and Tuyls, P. (2009). Memory leakage-resilient encryption based on physically unclonable functions. In Matsui, M., editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer.

[Atallah et al., 2008] Atallah, M. J., Bryant, E., Korb, J. T., and Rice, J. R. (2008). Binding software to specific native hardware in a VM environment: The PUF challenge and opportunity. In *VMSec*, pages 45–48. ACM.

[Barker and Kelsey, 2012] Barker, E. and Kelsey, J. (2012). Recommendation for random bit generator (RBG) constructions. *DRAFT NIST Special Publication 800-90C*.

[Batina et al., 2007] Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., and Verbauwhede, I. (2007). Public-key cryptography for RFID-tags. In *PerCom Workshops*, pages 217–222. IEEE Computer Society.

[Beckmann and Potkonjak, 2009] Beckmann, N. and Potkonjak, M. (2009). Hardware-based public-key cryptography with public physically unclonable functions. In *Information Hiding*, volume 5806 of *Lecture Notes in Computer Science*, pages 206–220. Springer.

[Bloom, 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

[Brassard et al., 1986] Brassard, G., Crépeau, C., and Robert, J.-M. (1986). Information theoretic reductions among disclosure problems. In *FOCS*, pages 168–173. IEEE Computer Society.

[Broder and Mitzenmacher, 2002] Broder, A. and Mitzenmacher, M. (2002). Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646.

[Brzuska et al., 2011a] Brzuska, C., Fischlin, M., Schröder, H., and Katzenbeisser, S. (2011a). Physically uncloneable functions in the universal composition framework. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70. Springer.

[Brzuska et al., 2011b] Brzuska, C., Fischlin, M., Schröder, H., and Katzenbeisser, S. (2011b). Physically uncloneable functions in the universal composition framework. *IACR Cryptology ePrint Archive*, 2011:681.

[Busch et al., 2009] Busch, H., Katzenbeisser, S., and Baecher, P. (2009). PUF-based authentication protocols - revisited. In *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 296–308. Springer.

[Busch et al., 2010] Busch, H., Sotáková, M., Katzenbeisser, S., and Sion, R. (2010). The PUF promise. In *Trust and Trustworthy Computing*, volume 6101 of *Lecture Notes in Computer Science*, pages 290–297. Springer.

[Canetti, 2001] Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society.

[Canetti et al., 2007] Canetti, R., Dodis, Y., Pass, R., and Walfish, S. (2007). Universally composable security with global setup. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer.

[Canetti and Fischlin, 2001] Canetti, R. and Fischlin, M. (2001). Universally composable commitments. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer.

[Canetti and Krawczyk, 2002] Canetti, R. and Krawczyk, H. (2002). Universally composable notions of key exchange and secure channels. In *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer.

[Canetti et al., 2006] Canetti, R., Kushilevitz, E., and Lindell, Y. (2006). On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167.

[Canetti et al., 2002] Canetti, R., Lindell, Y., Ostrovsky, R., and Sahai, A. (2002). Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM.

[Canetti and Rabin, 2003] Canetti, R. and Rabin, T. (2003). Universal composition with joint state. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer.

[Crépeau, 1987] Crépeau, C. (1987). Equivalence between two flavours of oblivious transfers. In *CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 350–354. Springer.

[DeJean and Kirovski, 2007] DeJean, G. and Kirovski, D. (2007). RF-DNA: Radio-frequence certificates of authenticity. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 346–363. Springer.

[Dodis et al., 2008] Dodis, Y., Ostrovsky, R., Reyzin, L., and Smith, A. (2008). Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139.

[Frikken et al., 2009] Frikken, K. B., Blanton, M., and Atallah, M. J. (2009). Robust authentication using physically unclonable functions. In *ISC*, volume 5735 of *Lecture Notes in Computer Science*, pages 262–277. Springer.

[Gassend, 2003] Gassend, B. (2003). Physical random functions. Master thesis, Massachusetts Institute of Technology, Massachusetts Institut of Technology.

[Gassend et al., 2002a] Gassend, B., Clarke, D. E., van Dijk, M., and Devadas, S. (2002a). Controlled physical random functions. In *ACSAC*, pages 149–160. IEEE Computer Society.

[Gassend et al., 2002b] Gassend, B., Clarke, D. E., van Dijk, M., and Devadas, S. (2002b). Silicon physical random functions. In *ACM Conference on Computer and Communications Security*, pages 148–160. ACM.

[Gassend et al., 2003] Gassend, B., Clarke, D. E., van Dijk, M., and Devadas, S. (2003). Delay-based circuit authentication and applications. In *SAC*, pages 294–301. ACM.

[Gassend et al., 2004] Gassend, B., Lim, D., Clarke, D., Devadas, S., and van Dijk, M. (2004). Identification and authentication of integrated circuits. *Concurrency - Practice and Experience*, 16(11):1077–1098.

[Gassend et al., 2008] Gassend, B., van Dijk, M., Clarke, D., Torlak, E., Tuyls, P., and Devadas, S. (2008). Controlled physical random functions and applications. *ACM Trans. Inf. Syst. Secur.*, 10(4).

[Goldreich et al., 1987] Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM.

[Goldwasser et al., 2008] Goldwasser, S., Kalai, Y. T., and Rothblum, G. N. (2008). One-time programs. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 39–56. Springer.

[Gora et al., 2009] Gora, M. A., Maiti, A., and Schaumont, P. (2009). A flexible design flow for software IP binding in commodity FPGA. In *SIES*, pages 211–218. IEEE.

[Guajardo et al., 2007] Guajardo, J., Kumar, S. S., Schrijen, G. J., and Tuyls, P. (2007). FPGA intrinsic PUFs and their use for IP protection. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer.

[Guajardo et al., 2008] Guajardo, J., Kumar, S. S., Schrijen, G. J., and Tuyls, P. (2008). Brand and IP protection with physical unclonable functions. In *ISCAS*, pages 3186–3189. IEEE Computer Society.

[Hammouri and Sunar, 2008] Hammouri, G. and Sunar, B. (2008). PUF-HB: A tamper-resilient HB based authentication protocol. In *ACNS*, volume 5037 of *Lecture Notes in Computer Science*, pages 346–365. Springer.

[Håstad et al., 1999] Håstad, J., Impagliazzo, R., Levin, L. A., and Luby, M. (1999). A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396.

[Hazay and Lindell, 2008] Hazay, C. and Lindell, Y. (2008). Constructions of truly practical secure protocols using standard smartcards. In *ACM Conference on Computer and Communications Security*, pages 491–500. ACM.

[Hofheinz et al., 2005] Hofheinz, D., Müller-Quade, J., and Unruh, D. (2005). Universally composable zero-knowledge arguments and commitments from signature cards. In *Proc. of the 5th Central European Conference on Cryptology MoraviaCrypt 2005*.

[ID, 2011] ID, I. (2011). http://www.intrinsic-id.com.

[Ignatenko et al., 2006] Ignatenko, T., Schrijen, G.-J., Škorić, B., Tuyls, P., and Willems, F. M. J. (2006). Estimating the secrecy rate of physical uncloneable functions with the context-tree weighting method. In *Proceedings of the IEEE International Symposium on Information Theory 2006*, pages 499–503. IEEE Computer Society.

[Katz, 2007] Katz, J. (2007). Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128. Springer.

[Katzenbeisser et al., 2012] Katzenbeisser, S., Koçabas, Ü., Rozic, V., Sadeghi, A.-R., Verbauwhede, I., and Wachsmann, C. (2012). PUFs: Myth, fact or busted? a security evaluation of physically unclonable functions (PUFs) cast in silicon. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 283–301. Springer.

[Katzenbeisser et al., 2011] Katzenbeisser, S., Koçabas, Ü., van der Leest, V., Sadeghi, A.-R., Schrijen, G. J., Schröder, H., and Wachsmann, C. (2011). Recyclable PUFs: Logically reconfigurable PUFs. In *CHES*, volume 6917 of *Lecture Notes in Computer Science*, pages 374–389. Springer.

[Kilian, 1988] Kilian, J. (1988). Founding cryptography on oblivious transfer. In *STOC*, pages 20–31. ACM.

[Kumar et al., 2008] Kumar, S. S., Guajardo, J., Maes, R., Schrijen, G. J., and Tuyls, P. (2008). The butterfly PUF: Protecting IP on every FPGA. In *HOST*, pages 67–70. IEEE Computer Society.

[Kursawe et al., 2009] Kursawe, K., Sadeghi, A.-R., Schellekens, D., Škorić, B., and Tuyls, P. (2009). Reconfigurable physical unclonable functions – enabling technology for tamper-resistant storage. In *HOST*, pages 22–29. IEEE Computer Society.

[Lee et al., 2004] Lee, J. W., Lim, D., Gassend, B., Suh, G. E., van Dijk, M., and Devadas, S. (2004). A technique to build a secret key in integrated circuits for identification and authentication applications. In *In Proceedings of the IEEE VLSI Circuits Symposium*, pages 176–179. IEEE Computer Society.

[Lim, 2004] Lim, D. (2004). Extracting secret keys from integrated circuits. Master thesis, Massachusetts Institute of Technology, Massachusetts Institute of Technology.

[Lim et al., 2005] Lim, D., Lee, J. W., Gassend, B., Suh, G. E., van Dijk, M., and Devadas, S. (2005). Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205.

[Lindell, 2009] Lindell, Y. (2009). General composition and universal composability in secure multiparty computation. volume 22, pages 395–428. Springer.

[Maes et al., 2012] Maes, R., Herrewege, A. V., and Verbauwhede, I. (2012). PUFKY: A fully functional PUF-based cryptographic key generator. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 302–319. Springer.

[Maes et al., 2008] Maes, R., Tuyls, P., and Verbauwhede, I. (2008). Intrinsic PUFs from flip-flops on reconfigurable devices. In *Workshop on Information and System Security (WISSec)*, page 17.

[Maes and Verbauwhede, 2010] Maes, R. and Verbauwhede, I. (2010). *Physically Unclonable Functions: a Study on the State of the Art and Future Research Directions*, chapter 1. Towards Hardware-Intrinsic Security. Springer.

[Merkle, 1980] Merkle, R. C. (1980). Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134. IEEE Computer Society.

[Micali and Reyzin, 2001] Micali, S. and Reyzin, L. (2001). Min-round resettable zero knowledge in the public key model. In *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 373–393. Springer.

[Mitzenmacher and Upfal, 2005] Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[Nielsen, 2002] Nielsen, J. B. (2002). Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer.

[O'Donnell et al., 2004] O'Donnell, C. W., Suh, G. E., and Devadas, S. (2004). PUF-based random number generation. CSAIL CSG Technical Memo 481, Massachusetts Institute of Technology.

[Ostrovsky et al., 2012] Ostrovsky, R., Scafuro, A., Visconti, I., and Wadia, A. (2012). Universally composable secure computation with (malicious) physically uncloneable functions. *IACR Cryptology ePrint Archive*, 2012:143.

[Pappu, 2001] Pappu, R. S. (2001). *Physical One-Way Functions*. Phd thesis, Massachusetts Institut of Technology, Massachusetts Institut of Technology.

[Pappu, 2003] Pappu, R. S. (2003). Physical one-way functions. *RSA Laboratories Cryptobytes*, 6(2):21–32.

[Pappu et al., 2002] Pappu, R. S., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030.

[Posch, 1998] Posch, R. (1998). Protecting devices by active coating. *Journal of Universal Computer Science*, 4(7):652–668.

[Rührmair, 2009] Rührmair, U. (2009). On a public key variant of physical unclonable functions. *IACR Cryptology ePrint Archive*, 2009:255.

[Rührmair, 2010] Rührmair, U. (2010). Oblivious transfer based on physical unclonable functions. In *TRUST*, volume 6101 of *Lecture Notes in Computer Science*, pages 430–440. Springer.

[Rührmair et al., 2010] Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., and Schmidhuber, J. (2010). Modeling attacks on physical unclonable functions. In *ACM Conference on Computer and Communications Security*, pages 237–249. ACM.

[Rührmair et al., 2009] Rührmair, U., Sölter, J., and Sehnke, F. (2009). On the foundations of physical unclonable functions. *IACR Cryptology ePrint Archive*, 2009:277.

[Rührmair and van Dijk, 2012] Rührmair, U. and van Dijk, M. (2012). Practical security analysis of PUF-based two-player protocols. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 251–267. Springer.

[Sadeghi et al., 2010] Sadeghi, A.-R., Visconti, I., and Wachsmann, C. (2010). *Enhancing RFID Security and Privacy by Physically Unclonable Functions*, chapter 5, pages 281–305. Towards Hardware-Intrinsic Security. Springer.

[Simpson and Schaumont, 2006] Simpson, E. and Schaumont, P. (2006). Offline hardware/software authentication for reconfigurable platforms. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 311–323. Springer.

[Suh and Devadas, 2007] Suh, G. E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. In *DAC*, pages 9–14. IEEE.

[Suh et al., 2005a] Suh, G. E., O'Donnell, C. W., and Devadas, S. (2005a). AEGIS: A single-chip secure processor. *Information Security Technical Report*, 10(2):63–73.

[Suh et al., 2007] Suh, G. E., O'Donnell, C. W., and Devadas, S. (2007). AEGIS: A single-chip secure processor. *IEEE Design & Test of Computers*, 24(6):570–580.

[Suh et al., 2005b] Suh, G. E., O'Donnell, C. W., Sachdev, I., and Devadas, S. (2005b). Design and implementation of the AEGIS single-chip secure processor using physical random functions. In *ISCA*, pages 25–36. IEEE Computer Society.

[Thompson, 1996] Thompson, A. (1996). An evolved circuit, intrinsic in silicon, entwined with physics. In *ICES*, volume 1259 of *Lecture Notes in Computer Science*, pages 390–405. Springer.

[Tuyls et al., 2006] Tuyls, P., Schrijen, G. J., Škorić, B., van Geloven, J., Verhaegh, N., and Wolters, R. (2006). Read-proof hardware from protective coatings. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 369–383. Springer.

[Tuyls et al., 2007a] Tuyls, P., Škorić, B., Ignatenko, T., Willems, F., and Schrijen, G.-J. (2007a). Entropy estimation for optical PUFs based on context-tree weighting methods. In *Security with Noisy Data*, pages 217–233. Springer.

[Tuyls et al., 2007b] Tuyls, P., Škorić, B., and Kevenaar, T. A. M. (2007b). *Security with Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting*. Springer.

[Tuyls et al., 2007c] Tuyls, P., Škorić, B., Petković, M., and Jonker, W. (2007c). *Strong Authentication with Physical Unclonable Functions*, chapter 10, pages 133–148. Security, Privacy, and Trust in Modern Data Management. Springer.

[Tuyls et al., 2005] Tuyls, P., Škorić, B., Stallinga, S., Akkermans, A., and Ophey, W. (2005). Information-theoretic security analysis of physical uncloneable functions. In *Financial Cryptography*, volume 3570 of *Lecture Notes in Computer Science*, pages 141–155. Springer.

[van Dijk and Rührmair, 2012] van Dijk, M. and Rührmair, U. (2012). Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results. *IACR Cryptology ePrint Archive*, 2012:228.

[Verayo, 2010] Verayo (2010). http://www.verayo.com.

[Škorić et al., 2006a] Škorić, B., Maubach, S., Kevenaar, T., and Tuyls, P. (2006a). Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied Physics*, 100.

[Škorić et al., 2006b] Škorić, B., Maubach, S., Kevenaar, T. A. M., and Tuyls, P. (2006b). Information-theoretic analysis of coating pufs. *IACR Cryptology ePrint Archive*, 2006:101.

[Škorić and Tuyls, 2007] Škorić, B. and Tuyls, P. (2007). *Security, Privacy and Trust in Modern Data Management*, chapter Strong Authentication with Physical Unclonable Functions, pages 133–148. Springer.

[Škorić et al., 2005] Škorić, B., Tuyls, P., and Ophey, W. (2005). Robust key extraction from physical unclonable functions. In *ACNS*, volume 3531 of *Lecture Notes in Computer Science*, pages 407–422. Springer.

[Wolf and Wullschleger, 2006] Wolf, S. and Wullschleger, J. (2006). Oblivious transfer is symmetric. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 222–232. Springer.