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

Marten van Dijk Ulrich Rührmair

April 25, 2012

Abstract

We investigate the power of physical unclonable functions (PUFs) as a new primitive in cryptographic protocols. Our contributions split into three parts. Firstly, we focus on the realizability of PUF-protocols in a special type of stand-alone setting (the "stand-alone, good PUF setting") under minimal assumptions. We provide new PUF definitions that require only weak average security properties of the PUF, and prove that these definitions suffice to realize secure PUF-based oblivious transfer (OT), bit commitment (BC) and key exchange (KE) in said setting. Our protocols for OT, BC and KE are partly new, and have certain practicality and security advantages compared to existing schemes.

In the second part of the paper, we formally prove that there are very sharp limits on the usability of PUFs for OT and KE *beyond* the above stand-alone, good PUF scenario. We introduce two new and realistic attack models, the so-called posterior access model (PAM) and the bad PUF model, and prove several impossibility results in these models. First, OT and KE protocols whose security is solely based on PUFs are generally impossible in the PAM. More precisely, one-time access of an adversary to the PUF after the end of a single protocol (sub-)session makes all previous (sub-)sessions provably insecure. Second, OT whose security is solely based on PUFs is impossible in the bad PUF model, even if only a stand alone execution of the protocol is considered (i.e., even if no adversarial PUF access after the protocol is allowed). Our impossibility proofs do not only hold for the weak PUF definition of the first part of the paper, but even apply if ideal randomness and unpredictability is assumed in the PUF, i.e., if the PUF is modeled as a random permutation oracle.

In the third part, we investigate the feasibility of PUF-based bit commitment beyond the stand-alone, good PUF setting. For a number of reasons, this case is more complicated than OT and KE. We first prove that BC is impossible in the bad PUF model if players have got access to the PUF between the commit and the reveal phase. Again, this result holds even if the PUF is "ideal" and modeled as a random permutation oracle. Secondly, we sketch (without proof) two new BC-protocols, which can deal with bad PUFs *or* with adversarial access between the commit and reveal phase, but not with both.

We hope that our results can contribute to a clarification of the usability of PUFs in cryptographic protocols. They show that new hardware properties such as offline certifiability and the erasure of PUF responses would be required in order to make PUFs a broadly applicable cryptographic tool. These features have not yet been realized in practical PUF-implementations and generally seem hard to achieve at low costs. Our findings also show that the question how PUFs can be modeled comprehensively in a UC-setting must be considered at least partly open.

1 Introduction

Since the time of Kerckhoff's principle [1], most cryptographic schemes have been built on the concept of a secret key. This forces modern security hardware to contain a piece of digital information that is, and must remain, unknown to the adversary. It is long known that this requirement can be difficult to uphold in practice. Physical attacks like invasive, semi-invasive or side-channel attacks, as well as software attacks like malware, can lead to key exposure and full security breaks. As Ron Rivest emphasized in his keynote talk at CRYPTO 2011 [36], merely calling a bit string a "secret key" does not make it secret, but rather identifies it as an interesting target for the adversary.

Indeed, one of the initial motivations in the development of *Physical Unclonable Functions (PUFs)* was their promise to better protect secret digital keys in vulnerable hardware. A PUF is an (at least partly) disordered physical system P that can be excited with external stimuli or so-called challenges c. It reacts with corresponding responses r, which depend on the challenge and on the micro- or nanoscale structural disorder that is present in

the PUF. It is assumed that this disorder cannot be cloned or reproduced exactly, not even by the PUF's original manufacturer, and that it is unique to each PUF. Each PUF P thus implements a unique and individual function g that maps challenges c from an admissible challenge set to responses $r = g(c)$. The tuples (c, r) are usually called the challenge-response pairs (CRPs) of the PUF. Due to its complex internal structure, a PUF can avoid some of the shortcomings of classical digital keys. It is usually harder to read out, predict, or derive PUF-responses than to obtain digital keys that are stored in non-volatile memory. The PUF-responses are only generated when needed, which means that no secret keys are present permanently in an easily accessible digital form. These facts have been exploited in the past for different security protocols. Prominent examples include schemes for identification [33, 17] or various forms of (tamper sensitive) key storage and applications thereof, for example intellectual property protection or read-proof memory [21, 26, 48].

In recent years, however, the use of PUFs in more advanced cryptographic protocols has been investigated. Not their application as a key storage mechanism is in the focus of these approaches, but their usability as a novel, and possibly very powerful, cryptographic primitive. The use of the PUF in these protocols is similar to a "physical random oracle". It is physically transferred between the parties, and can be queried for responses *only* by the very party who currently holds physical possession of it. Its challenge-response behavior is so complex that its response to a randomly chosen challenge cannot be predicted purely numerically and without physical measurement of the PUF, not even by a person who held the PUF at earlier points in time. In 2010, Rührmair [38] showed that oblivious transfer (OT) can be realized by physically transferring a PUF between two parties in this setting. In the same year, the first formal security proof for a PUF-protocol was provided by Rührmair, Busch and Katzenbeisser [39]. At CRYPTO 2011, Brzuska et al. [7] presented a way to adapt Canetti's universal composition (UC) framework [9] to include PUFs, an approach that was followed up in 2012 by very recent work of Ostrovsky et al. [31].

In this paper, we continue this line of research, and try to clarify the power of PUFs in cryptographic protocols. We approach the problem from two opposite ends: At the one end of the spectrum, we prove the security of several PUF protocols in a stand-alone scenario under a very weak PUF definition, and under the assumption that all parties faithfully generate and use non-manipulated PUFs. This ideal scenario is called the stand-alone, good PUF model. At the other end of the spectrum, we introduce two new and very realistic attack models: The first is the so-called *posterior access model (PAM)*, in which it is assumed that the adversary has got physical access to the PUF at least once after at least one protocol session or subsession (if there are any subsessions). Among other situations, the PAM applies in practice whenever the same PUF is used multiple sequential protocol sessions or subsessions. The second new model is the *bad PUF model*, which allows malicious players to fabricate and use malicious hardware that looks like a PUF from the outside, exhibiting the same CRP behavior etc., but which possesses hidden extra properties that enable fraud. We show that in these two models, several PUF-protocols are provably impossible. This includes PUF-based OT and KE in the PAM, and OT in a stand alone, bad PUF model. Our impossibility results even apply if we assume a very strong randomness and unpredictability of the PUF, i.e., if the PUF is modeled as a random permutation oracle. Finally, we consider the case of PUF-based BC, providing one impossibility result and two new constructions (without proof) that can withstand adversarial access between the commit and the reveal phase or the use of bad PUFs, but not both.

Related Work. Our paper relates to existing literature as follows. Rührmair [38] was the first to give (without proof) an oblivious transfer protocol for PUFs, but his protocol is more complicated than ours, and is based on an interactive hashing step with a linear number of rounds. Our modified OT-scheme is still based on interactive hashing, but has constant rounds.

Rührmair, Busch and Katzenbeisser [39] were the first to provide PUF-definitions together with formal security proofs, but only considered schemes for identification. They do not treat more advanced protocols like OT and BC or consider more sophisticated attacks like bad PUFs. Furthermore, their PUF-definition involves a relatively large number of parameters.

Rührmair, Jaeger and Algasinger [41] provided an attack on a session key exchange protocol by Tuyls and Skoric [47], in which they implicitly assume access to the PUF after a protocol execution. Their attack motivated our posterior access model (see Section 2). One difference to our work is that they do not lead general impossibility proofs, but focus on attacking one specific protocol.

Brzuska, Fischlin, Schröder and Katzenbeisser presented at Crypto 2011 one (of several possible) ways to adapt Canetti's UC framework to PUFs [7, 8]. They give PUF-schemes for OT, BC and KE, and prove them secure in their setting. Their work differs from our work in a number of aspects: First, quadratic attacks exist on their OT- and BC-protocols, whence they cannot be used safely in practice with optical PUFs or with PUFs of

	OT	**KE**	**BC**
Stand-Alone Good PUF	✓ Prot 4	✓ Prot 9	✓ Prot 8
Stand-Alone Bad PUF	× Thm 22	✓ Prot 9	ⓥ Prot 24
Posterior Access Good PUF	× Thm 18	× Thm 19	✓ Prot 8
Posterior Access Bad PUF	× Thm 18	× Thm 19	ⓥ Prot 24
Access Before Reveal Phase Good PUF	—	—	ⓥ Prot 25
Access Before Reveal Phase Bad PUF	—	—	× Thm 23

Table 1: A taxonomy of protocols whose security is solely based on PUFs in various scenarios (see Section 2 for detailed definitions). A checkmark indicates that we provide a protocol together with a formal security argument. A cross means we lead a formal impossibility proof. A circled checkmark says that we give a protocol without a formal security argument, possibly making additional security assumptions. A hyphen shows that the listed scenario does not apply to the respective protocol. All impossibility results in the bad PUF models hold already if the malicious parties use no other than simulatable and challenge-logging PUFs (Section 2.4). Pointers to the respective theorems and protocols are provided, too.

challenge lengths 64 bits, not even in the stand-alone, good PUF setting (see [40] and Appendix B). Second, their PUF-based BC-protocol is a generic reduction to OT, while our BC-protocol is the first direct PUF-construction. Finally, and perhaps most importantly, their adaption of the UC-framework to PUFs does not deal with the cases that adversaries have repeated access to the same PUF in multiple protocol (sub-)sessions, or that players actively use manipulated, malicious PUFs. We introduce these two new and realistic attack models in this manuscript, and show that several protocols provably cannot solely be based on PUFs under these attacks.

Just recently, Ostrovsky, Scafuro, Visconti and Wadia re-considered PUFs in the UC framework, and presented new protocols and security proofs in an ePrint paper [31]. They introduce a new class of PUFs termed malicious PUFs that is related, but not equal to our bad PUF model. However, they do not give formal impossibility proofs for PUF protocols, and do not consider specific bad PUFs (such as the simulatable bad PUFs or challenge-logging bad PUFs introduced in Section 2). Another difference to our work is that they do not focus on protocols that are solely based on PUFs, but combine PUFs with classical computational assumptions. Contrarily, all protocols that we provide are based on PUFs only, and the impossibility proofs which we lead refer to protocols that are solely based on PUFs. We will further comment on the exact relationship between the work of Ostrovsky et al. and our efforts in future versions of this paper.

Organization of this Paper. Section 2 describes different communication models and attack scenarios. Section 3 gives protocols for OT, BC and KE, and security proofs in the stand-alone, good PUF setting. Section 4 lays the mathematical foundations for our impossibility results, which are proven in Section 5. Section 6 deals with PUF-based bit commitment protocols. Section 7 summarizes our work.

2 Communication Models and Attack Scenarios for PUF-Protocols

Let us overview and motivate the four communication models and attack scenarios that are relevant for this paper.

2.1 The Stand-Alone, Good PUF Model

The stand-alone, good PUF model is the vanilla model among the considered scenarios. We assume that there is only one single, isolated execution of a PUF-protocol (or (sub-)session, if there are any (sub-)sessions). The PUF

cannot be accessed any more or communicated with after the end of the protocol (or (sub-)session). [1]

The parties and also any external adversaries may only use faithfully generated, non-manipulated, "good" PUFs. They are not allowed to change or manipulate existing PUF hardware when they have access to it. Apart from these restrictions, all parties may be actively malicious, and can deviate from the protocol arbitrarily (including the exchange of one good PUF against another good PUF).

The stand-alone, good PUF model will not be realistic in many practical applications, but makes a clean first setting for studying the security of PUF-protocols.

2.2 The UC-Model of Brzuska et al.

Brzuska et al. [7, 8] proposed one possible method how Canetti's UC-framework can be adapted to PUFs. For a detailed treatment we refer the readers to the original papers [7, 8], but summarize the features of their model that are most relevant for us in the sequel:

1. It is assumed that all used PUFs are drawn faithfully from a previously specified PUF-family. Not only external adversaries, but also the players themselves are not allowed to use malicious hardware instead of a PUF, physically manipulate a PUF, or add malicious hardware to an existing PUF.

2. Only one PUF can be used per protocol session `sid`. The PUF is bound to this protocol session and cannot be used in another session.

3. The adversary does not have physical access to the PUF between the different subsessions `ssid` of a protocol.

For completeness we indicate where the above features are specified in [8]: Features 1 and 2 directly follow from the specification of the ideal PUF-functionality $\mathcal{F}_{\mathsf{PUF}}$, in particular the first and third dotted item of Fig. 2 of [8]. Regarding feature 2, the functionality $\mathsf{init}_{\mathsf{PUF}}$ specifies that $\mathcal{F}_{\mathsf{PUF}}$ turns into the waiting state if the session `sid` already contains a PUF. And the functionality $\mathsf{handover}_{\mathsf{PUF}}$ specifies that `sid` remains unchanged in the handover, i.e., the PUF remains in the same session `sid` after the handover process. Feature 3 follows from the treatment of the subsessions `ssid` throughout their paper. Examples include Figs. 3 to 8, the protocols given in Figs. 3 and 7, or the proof of Theorem 7.1, where the adversary is only allowed to access the PUF in the set-up phase, but not during or between the different subsessions.

Please note that the above features are not rudimentary aspects of the model of [7, 8], but that they are central to the security of their protocols and the validity of their security proofs.

2.3 The Posterior Access Model

The UC model of Brzuska et al. and their protocols assume that the adversary cannot access the PUF between different (sub-)sessions of the protocol. However, this requirement cannot be guaranteed in many natural PUF-applications. To see this, consider a well-established application scenario of PUFs: Their use on a smart-card (or bank card) that has been issued by a central authority CA, and which is subsequently used in different terminals by a user [33, 32]. To be more concrete, let us assume that the PUF is repeatedly employed in different terminals for a session key exchange between the CA on the one hand, and the smart-card/terminals on the other hand. Since an adversary could set up fake terminals, add fake readers to the card slots of terminals, or gain temporary possession of the bank card when the user employs it in different contexts (for example when he is paying with it), a straightforward and very realistic assumption is that an adversary will have temporary physical access to the PUF between the different key exchange (sub-)sessions.

This natural scenario — and the multiple adversarial access inherent to it — is hard to express in the framework of Brzuska et al. The reason is twofold (see Section 2.2): First, their model does not allow the same PUF to be used in different sessions. Second, while the same PUF could well be used in different subsessions, the adversary has no PUF-access between these different subsessions, for example in the OT or KE protocol of [7, 8].

This motivates the introduction and investigation of new and more realistic attack models. One straightforward possibility would be to assume that the adversary has access to the PUF between *each* session or subsession. However, for our upcoming impossibility results even a weaker and more general assumption suffices. We will show in Section 5 that whenever the adversary gains access to a PUF only once after the end of a (sub-)session, all

[1] One (costly) possibility to realize this in pratice would be to physically destroy the PUF directly at the end of the protocol (or (sub-)session).

previous (sub-)sessions that have been carried out by using this PUF become insecure. This leads to the following model:

The Posterior Access Model (PAM). In the PAM, we assume that the adversary and malicious players can access the PUF at least one time after at least one completed protocol session or subsession (if there are any subsessions).

Security of Existing PUF-Protocols in the PAM. We observe that many existing PUF-protocols are naturally no longer secure in the PAM. This applies to the OT-protocol of Rührmair [38] and to the OT and KE protocol of Brzuska et al. [7]. Since this is not of central importance to this paper, we provide an example attack on the OT protocol of Brzuska et al. in the posterior access model in Appendix C. Interested readers may perhaps use this example attack to become familiar with the PAM.

2.4 The Bad PUF Model

One other central assumption in the UC-model of Brzuska et al. is that the players will not use malicious PUF-hardware with properties beyond the expected PUF functionality. Consider again the above smart-card example for illustration purposes. Let us assume that the CA issues the card that carries the PUF, and that the CA and the smart-card/terminals want to run an OT protocol in this setting. We must assume that the CA is not fully trusted by the smart-card/terminals (note that if the CA was fully trusted, then the smart-card/terminals would not require an OT implementation). However, a malicious CA can cheat easily in this scenario by putting a malicious PUF-hardware (a "bad PUF") instead of a normal PUF on the smart card. To name one example, the CA could replace the normal PUF by a pseudo random number generator (PRNG) with a seed s known to the CA. This enables the CA to simulate and predict all responses of this "bad PUF" without being in physical possession of it, and breaks one of the essential security features of the purported "PUF" on the bankcard, namely its unpredictability. It is not too difficult to see that under the assumptions that the CA replaces the PUF by a PRNG the currently known OT protocols of Rührmair [38] and Brzuska et al. [7] break down and are no longer secure. If the CA acts as OT-receiver, for example, it can learn both bits of the OT-sender.

This motivates a systematic study of bad PUF attacks. Generally, we denote by the term *"bad PUF"* a hardware system that looks like a proper PUF from the outside, showing an input-output behavior indistinguishable from a proper PUF, but which possesses secret, additional properties that allow cheating. The assumed similar input-output behavior makes it impossible to distinguish a bad PUF from a proper PUF by mere challenge-response measurements. In order to detect bad PUFs, an honest party would need to physically open the PUF-hardware and to inspect it thoroughly (as a regular and dedicated step of the protocol), a task that is beyond the capabilities of an average user. While detection of a bad PUF would not even be guaranteed with certainty by such a step (adversaries would presumably develop obfuscation techniques for the bad PUF hardware), it would surely destroy the opened PUF, even if it was "good" and non-manipulated. Overall, this makes bad PUFs a very simple and effective way to cheat.

From an abstract perspective, bad PUFs exploit the fact that PUFs are real physical objects. Unlike the clean binary strings transferred in classical cryptographic protocols, these objects may bring about unwanted properties. They can act as real, physical "Trojans". The two types of bad PUFs that we focus on in this paper are the PUFs that are numerically simulatable by their manufacturer (but by no one else), and bad PUFs that "log" or record all challenges that have been applied to them. Both are particularly easy to implement, but suffice for our formal impossibility results in the upcoming sections.

Simulatable Bad PUFs (SIM-PUFs). The concept of a simulatable PUF (or SIM-PUF, for short) is relatively simple: It is a hardware system that looks like a PUF, having a challenge-response interface etc., but which possesses a simulation algorithm Sim. Sim takes as input any challenge c, and computes in polynomial time the corresponding response r. It is assumed that Sim has been derived during the fabrication of the simulatable PUF via the special construction of the PUF. External parties who merely have access to the simulatable PUF after fabrication are not able to derive a simulation model.

In practice there are several possibilities for implementing simulatable PUFs. A straightforward and very efficient way is to use a trapdoor one-way permutation or pseudo random function g_s based on a short digital seed

s. The hardware of the simulatable PUF simply implements g_s. Whenever the PUF is interrogated over the digital interface with a challenge c, the hardware outputs the response $r = g_s(c)$.

The party who manufactured the PUF knows both g as well as seed s and can easily simulate the input-output behavior of the PUF. Furthermore, if a cryptographically hard pseudo-random function is used, it is practically infeasible for the honest parties to distinguish the bad PUF from a proper PUF with a real, random output. Two other, more involved examples of simulatable PUFs are described in Appendix E. They are not strictly necessary for the exposition of this paper, but they add interesting extra information.

Challenge-Logging Bad PUFs (CL-PUFs). A second feature that bad PUFs may possess is challenge-logging. A challenge-logging PUF (CL-PUF for short) with secret challenge c^*, also called the access challenge, is a malicious piece of hardware that looks like a proper PUF from the outside (with a challenge-response interface etc.), but which possesses the following properties:

1. Except for one input challenge c^*, the challenge-response behavior of a CL-PUF is exactly like that of the underlying PUF. Whenever a challenge c unequal to c^* is applied to the CL-PUF via its interface, the challenge is passed on to the underlying PUF. The corresponding response r is obtained from the latter, and the CL-PUF uses this response r as its output.

2. The CL-PUF has a non-volatile memory (NVM) module in which it automatically records all challenges that have been applied to it.

3. When challenge c^* is applied to the CL-PUF, it does not pass on this challenge to the underlying PUF as usual. Instead, the CL-PUF outputs the entire content of the non-volatile memory module (i.e., all challenges that have previously been applied to it) via the challenge-response interface, and erases the content of the NVM module.

If the PUF has a large, preferably exponential challenge set, then the probability that someone by chance inputs c^* and detects the challenge-logging feature is negligibly small.

Countermeasures? At first sight, a seemingly simple countermeasure against bad PUFs would be to "authenticate" or "certify" the PUF in some way to detect bad PUFs. For example, a trusted authority (TA) could send a list of CRPs as a "fingerprint" of a genuine PUF to the players before any protocol execution. On closer inspection, however, this countermeasure turns out to be problematic and ineffective.

First of all, the use of a TA that needs to be called in every single protocol session would make the use of PUFs in security protocols obsolete. The aspired functionalities could then be implemented in a much simpler fashion directly via the TA, avoiding the significant effort of physically transferring a PUF during the protocol. Secondly, CRP-based authentication does not rule out externally added malicious hardware, such as external challenge loggers. The latter do not affect the CRP-behavior of an existing (and previously certified) PUF.

Meaningful "certification" of a PUF hence requires not only to "identify" a PUF. It also must (i) exclude that external parts have been added to the PUF and that the PUF-hardware has been manipulated; and (ii) it should work offline, i.e., it must avoid calling a central TA in every execution of the protocol. Currently, no protocols or PUF implementations that realize these two properties have been considered in the literature. Given the current state of the field, it seems hard to design such methods, even more so at low costs. Once more, this makes bad PUFs a realistic and efficient method to cheat.

Brzuska et al. indeed assume certification of the PUF, but do not give protocols or methods how it can be achieved. For the above reasons, we believe that efficient certification is currently infeasible in practice. This holds even more if malicious players (not only external adversaries) generate and use manipulated PUFs.

Natural Limits on Bad PUFs and Super-Bad PUFs. How "bad" can a PUF be? When do protocols based on bad PUFs become straightforwardly impossible? Perhaps the most extreme type would be a PUF that has a real-time wireless connection to the malicious party. The party could use the wireless connection (i) to passively learn which challenges are applied to and/or which responses are obtained from the PUF by the honest parties, or (ii) to actively influence and alter the challenge-response behavior of the PUF. In the worst case, the malicious party receives in real-time any challenge that is applied to the PUF, and returns in real-time a personally selected

response, which the PUF then outputs. A less laborious possibility was that the malicious party sends one single signal to the PUF that flips some selected PUF-CRPs for good.

We call bad PUFs of the above type (where there is a wireless connection between the PUF and the malicious party) "super-bad PUFs". Super-bad PUFs are not our central topic in this publication. There are two reasons: Firstly, it is straightforward that many protocols cannot be based solely on PUFs (without making additional complexity assumptions) if malicious parties can use super-bad PUFs.[2] Secondly, communication between the super-bad PUF and the malicious player during a protocol can be prevented in several natural PUF-applications (for example bank cards) by shielding the PUF/bank card for the time in which the protocol is run. This allows at least protocols that are secure in a single execution, stand-alone setting. Such measures are already common today in Automated Teller Machines.

We comment that all of our formal impossibility results in the bad PUF model hold already if only two realistic and simple types of bad PUFs are used, namely simulatable PUFs and challenge-logging PUFs (see the last Section 2.4). We leave PUF-protocols that remain secure during multiple protocol executions in versions of the super-bad PUF model (possibly under additional computational assumptions) as a future research topic.

Security of Existing PUF-Protocols in the Bad PUF Model. Again, it is relatively easy to see that many existing PUF-protocols are no longer secure if the adversary can use bad PUFs, for example simulatable PUFs. This applies to the OT-protocol of Rührmair [38] and to the OT and BC protocol of Brzuska et al. [7]. Since this is not of central importance to this paper, we describe example attacks on the OT protocol of Brzuska et al. under the use of simulatable PUFs in Appendix D. Interested readers may perhaps use this example attack in order to become familiar with the bad PUF model.

2.5 Combinations of Model Features

As indicated in Table 1, we consider different combinations of attack models, such as "Posterior Access, Bad PUFs", or "Stand Alone, Bad PUFs". These combinations have the expectable properties that follow from our above discussion: For example, "Posterior Access, Bad PUFs" means that the adversary and malicious players are allowed to (i) have posterior access to the PUF as described in Section 2.3, and (ii) that they are allowed to use bad PUFs as described in Section 2.4. Similar statements hold for the other combinations of model features that we examine.

3 Protocols and Security Proofs in the Stand Alone, Good PUF Model

We now turn to the first part of the paper as announced in the abstract. We provide protocols for OT, BC and KE, and prove their security in the stand-alone, good PUF setting. The protocols that we provide are at least partly new; for example, we give the first direct BC construction that rests on the unpredictability of the PUF alone. The exact motivation for each protocol is described in the respective subsections. We start by giving a new PUF definition that is relatively simple, avoiding an asymptotic treatment and min-entropy conditions.

3.1 Yet Another PUF Definition (YAP)

The question about an intuitive security definition for PUFs has been open for some time. Early suggestions captured the intuition about PUFs well, but partly suffered from formal problems [43]. Recent suggestions by Rührmair, Busch and Katzenbeisser [39] and by Brzuska et al. [7] can be used in formal security proofs, but are relatively complicated. The framework of Armknecht et al. [2] mainly applies to so-called Physically Obfuscated Keys or POKs (sometimes also termed weak PUFs), for example SRAM PUFs, which are not relevant for this paper. Finding an intuitive PUF-definition that appeals to hardware designers and theoretical cryptographers alike seems at least partly open. Existing work indicates that some small concessions between formal correctness and simplicity might be inevitable.

[2] There are some parallels to another well-established example from classical cryptography here, namely the condition that the two provers in Multi-Prover Interactive Proof Systems must not communicate with each other. This is a necessary requirement for exploiting the extra power of two provers over one single prover. This is somewhat similar to the situation with super-bad PUFs: If real-time communication between the (super-bad) PUF and a malicious party is allowed, certain security features break down naturally.

In the following, we present an extremely simple definition, which still suffices for certain security proofs. It focuses on a single PUF, and does not require worst-case security (or min-entropy conditions), such as the definition of Brzuska et al. [7]. The reason is that many existing PUF-candidates do not fulfill such worst-case conditions, since they have strong correlations between certain selected CRPs. [3] Nevertheless, such correlations of a few specific CRPs do not hinder a PUF's applicability in typical protocols, in particular if it is used with randomly selected challenges. This motivates to merely require the weaker feature of average-case unpredictability in definitions.

Definition 1 (PUFs and Associated Functions)**.** *A PUF P is a physical object that can be stimulated by challenges c from a challenge space $\mathcal{C}_P$, by which it reacts with corresponding responses r from a response space $\mathcal{R}_P$. We model P by a associated function $g_P : \mathcal{C}_P \to \mathcal{R}_P$ that maps the challenges c to responses $r = g_P(c)$. The pairs $(c, g_P(c))$ are called challenge-response pairs (CRPs) of the PUF.*

Whenever the PUF P is clear from the context, we will often drop the index and write $\mathcal{C}, \mathcal{R}$ or g. Definition 1 assumes that suitable error correction techniques have been applied to the PUF, leading to a stable PUF output. The definition can easily be adjusted to the case of noisy PUF outputs by replacing $g_P(c)$ by a random variable, but we will not follow this route in this work.

In this paper, we will almost exclusively consider the case that $\mathcal{C} = \mathcal{R} = \{0,1\}^\lambda$, with λ being the security parameter. In order to achieve output length of λ in practice, fuzzy extraction of several consecutive responses can be applied, or the concatenation of the λ responses of λ independent hardware instantiations of a PUF to the same challenge can be used (as discussed in [7, 8]).

Definition 2 (ϵ-Unpredictability with respect to Parties)**.** *We model the ability of a party $\mathcal{A}$ to predict the output of a PUF P by a random variable A_P, which maps challenges $c \in \mathcal{C}$ to responses $r \in \mathcal{R}$ according to some probability distribution $\mathcal{D}_{\mathcal{A},P}$. We call a PUF P ϵ-unpredictable with respect to a party $\mathcal{A}$ if*

$$\mathsf{Prob}_{c \leftarrow \mathcal{C}}[A_P(c) = g_P(c)] \leq \epsilon.$$

Thereby the probability is taken over the random variable A and the uniformly random choice of $c \in \mathcal{C}$.

Since we consider PUFs with challenge and response length λ in this paper, please note that the probability ϵ might (for a well-designed or ideal PUF) be as low as $2^{-\lambda}$. Again, we will sometimes drop the index P if it is clear from the context.

Definition 2 is astonishingly simple, but suffices for the upcoming security arguments of Protocols 4, 8 and 9. Our framework is to some extent inspired by the work of Pavlovic [34]. Instead of quantifying over all possible (and infinitely many) Turing machines or adversaries that could attack a PUF, we focus on the concrete capabilities of a single adversary. We use the assumption that a given adversary cannot predict a PUF as a premise in our security proofs, which are then led relative to this adversary, if you like.

3.2 Oblivious Transfer

3.2.1 Interactive Hashing with Constant Rounds in a PUF-Setting

One basic tool in our upcoming constructions is interactive hashing [44]. Ding, Harnik, Rosen and Shaltiel [4] showed how to achieve secure protocols with only four rounds:

Lemma 3 (Interactive Hashing [4])**.** *Let $s \geq 2 + \log \lambda$. There exists a 4 message* interactive hashing (IH) *protocol between Alice with no input and Bob with input string $W \in \{0,1\}^\lambda$ that outputs to both players $(W_0, W_1) \in \{0,1\}^\lambda \times \{0,1\}^\lambda$, satisfying the following:*

Correctness: *If both players are honest, then $W_0 \neq W_1$ and there exists a $D \in \{0,1\}$ such that $W_D = W$. Furthermore, the distribution of W_{1-D} is $2^{-\lambda}$-close to uniform on all strings $\{0,1\}^\lambda \backslash \{W\}$ not equal to W.*

[3] As an example, consider the well-known Arbiter PUF [24, 45]: Flipping the first (leftmost) input bit will not change the output with a probability close to 1, as the resulting delay change is dominated by the accumulated delays in the rest of the structure. At the same time, flipping the rightmost input bit will almost certainly change the PUF's output, as the two signal paths are exchanged. This means that there are CRPs with strong positive or negative correlation. Related considerations hold for the Arbiter PUF variants Feed-Forward Arbiter PUF and XOR Arbiter PUF.

Security for Bob: *If Bob is honest, then (for every unbounded strategy by Alice)* $W_0 \neq W_1$ *and there exists a* $D \in \{0,1\}$ *such that* $W_D = W$*. If Bob chooses* W *uniformly at random, then* D *is uniform and independent of Alice's view.*

Security for Alice: *If Alice is honest, then (for every unbounded strategy by Bob) for every subset* $\mathcal{S} \subseteq \{0,1\}^\lambda$*,*

$$|\mathcal{S}| \leq 2^s \;\Rightarrow\; \Pr[W_0 \in \mathcal{S} \text{ and } W_1 \in \mathcal{S}] \leq 10 \cdot \frac{\lambda 2^s}{2^\lambda}$$

Note that interactive hashing is unconditionally secure in the sense that it does not require additional set-up or computational assumptions. The above IH protocol by Ding et al. uses a so-called "μ*-almost* t*-wise independent permutation space*" from which Alice uniformly selects a member. Since the selection only affects Alice's own security, interactive hashing does not need any pre-protocol agreement by Alice and Bob. I.e., no set-up assumptions are needed; the interactive hashing protocol is unconditionally secure. We will use the lemma to construct constant round OT and BC protocols solely based on PUFs in the sequel.

3.2.2 Oblivious Transfer Protocol

Following the PUF-based OT-protocols by Rührmair [38] and Brzuska et al. [7], we provide another PUF-based OT-protocol in this section. Our motivation for giving another protocol is as follows: Compared to the original protocol of Rührmair, the protocol below has been simplified, and also has a reduced round complexity due to its new interactive hashing step. In comparison to the OT-protocol of Brzuska et al., our approach does not allow the same type of quadratic attack that is described in Appendix B. Due to the interactive hashing step, the security proof Protocol 4 does not require conditions on the mutual (information-theoretic or computational) independence of more than one CRP of the PUF, such as in [7]. The average unpredictability of single CRPs (see Def. 2) suffices.

Let P be a PUF with $g_P : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Let the employed interactive hashing scheme be the four message IH scheme of Ding et al. [4], whose security is described in Lemma 3. We assume that the sender's input are two strings $s_0, s_1 \in \{0,1\}^\lambda$ and the receiver's input is a choice bit $b \in \{0,1\}$. The receiver intially holds the PUF.

Protocol 4: PUF-BASED 1-OUT-OF-2 OBLIVIOUS TRANSFER WITH INTERACTIVE HASHING

1. The receiver chooses a challenge c uniformly at random. He applies c to the PUF, and obtains a response r.
2. The receiver transfers the PUF to sender.
3. The sender and receiver execute an IH protocol, where the receiver has input c. Both get output c_0, c_1. Let i be the value where $c_i = c$.
4. The receiver sends $b' := b \oplus i$ to the sender.
5. The sender applies the challenges c_0 and c_1 to the PUF, obtaining responses r_0 and r_1.
6. The sender sends $S_0 := s_0 \oplus r_{b'}$ and $S_1 := s_1 \oplus r_{1-b'}$ to the receiver.
7. The receiver recovers the string s_b that depends on his choice bit b as $S_b \oplus r = s_b \oplus r_{b \oplus b'} \oplus r = s_b \oplus r_i \oplus r = s_b$.

We will now prove the security of Protocol 4 in the stand-alone, good PUF model using Lemma 3.

Lemma 5. *Protocol 4 is secure for the receiver, i.e., the sender does not learn the receiver's choice bit* b*.*

Proof. From Lemma 3 it follows that the value i is uniform and unknown to the sender. So b' does not give any information about whether $c = c_0$ or $c = c_1$. Therefore, the sender has no idea whether the receiver knows r_0 or r_1, which means that the sender does not know whether s_0 or s_1 has been revealed. □

Lemma 6. *Suppose that* $2^\lambda \geq 160\lambda^3$ *and let* $2^{-\lambda} \leq \epsilon \leq 1/(10\lambda)$*. If after step 2 in Protocol 4, the used PUF is* ϵ*-unpredictable with respect to the receiver and if the IH is based on parameter* $s = \lambda + \log\sqrt{\epsilon/(10\lambda)}$*, then Protocol 4 is secure for the sender in that the receiver is able to correctly guess both bit strings* s_0 *and* s_1 *with probability at most* $\sqrt{40\lambda\epsilon}$*.*

Proof. Let S_p be the set of challenges for which the receiver is able to correctly guess responses with probability at least p (notice that $c_i \in S_p$). By Definition 2, a lower bound on ϵ is given by $|S_p|p/2^\lambda \leq \epsilon$. This yields an upper bound on the cardinality of S_p,

$$|S_p| \leq \epsilon 2^\lambda / p. \tag{1}$$

In a standard run of Protocol 4, the receiver reconstructs the string s_b by using his knowledge of $r = r_i$. In order to also reconstruct s_{1-b}, he needs to guess r_{1-i}. Let q be the probability that the receiver is able to guess r_{i-1}. By the definition of S_p,

$$q \leq Prob(c_{1-i} \in S_p) + Prob(c_{i-1} \notin S_p)p \leq Prob(c_{1-i} \in S_p) + p. \tag{2}$$

Notice that the receiver is able to predict r_i with probability 1, therefore $\epsilon \geq 2^{-\lambda}$. Together with $2^\lambda \geq 160\lambda^3$ this implies $s \geq 2 + \log \lambda$ for s defined in the lemma. This means that Lemma 3 is applicable: The probability that r_{1-i} corresponds to a challenge c_{1-i} in S_p is by Lemma 3 at most $10 \cdot \lambda 2^{s-\lambda}$ if $|S_p| \leq 2^s$. Together with (1) this proves

$$\epsilon 2^\lambda / p \leq 2^s \;\Rightarrow\; Prob(c_{1-i} \in S_p) \leq 10\lambda 2^{s-\lambda}$$

Combined with (2) this gives

$$\epsilon 2^\lambda / p \leq 2^s \;\Rightarrow\; q \leq 10\lambda 2^{s-\lambda} + p.$$

Let $p = \epsilon 2^{\lambda - s}$, which is ≤ 1 if $10\lambda\epsilon \leq 1$ for s defined in the lemma. Then, for $s = \lambda + \log \sqrt{\epsilon/(10\lambda)}$, $q \leq 10\lambda 2^{s-\lambda} + \epsilon 2^{\lambda - s} = \sqrt{40\lambda\epsilon}$. □

Comments and Discussion. We remark once more that depending on the PUF and the adversary, ϵ may be as small as $2^{-\lambda}$. Furthermore, please note that $\log \sqrt{\epsilon/(10\lambda)} < 0$, whence $s < \lambda$ for the parameter choice of Lemma 6.

The security of Protocol 4 can be amplified by using a well-known result by Damgard, Kilian and Savail (see Lemma 3 of [11]):

Theorem 7 (OT-Amplification [11])**.** *Let (p, q)-WOT be a 1-2-OT protocol where the sender with probability p learns the choice bit c and the receiver with probability q learns the other bit b_{1-c}. Assume that $p + q < 1$. Then the probabilities p and q can be reduced by running k (p, q)-WOT-protocols to obtain a $(1 - (1 - p)^k, q^k)$-WOT protocol.*

In the case of our OT-Protocol 4 it holds that $p = 0$, whence the technique of Damgard et al. leads to an efficient security amplification, and to a $(0, q^k)$-WOT protocol. The PUF does not need to be transferred k times, but one PUF-transfer suffices.

Please note that the security guarantee of Lemma 6 contains a square root, but is otherwise very different from the quadratic attack described in Appendix B. This quadratic attack breaks the OT-protocol of Brzuska et al. with probability 1 if an adversary is able to read out $2^{\lambda/2}$ CRPs. The attack is independent of the cryptographic hardness and unpredictability of the PUF, and even holds for an ideal, perfectly random PUF. No probability amplification by the technique of Damgard et al. is possible any more after the attack. In addition to the quadratic attack, also "normal" attacks on the employed PUF (for example modeling attacks [42]) can be mounted.

To the contrary, our protocol only allows modeling attacks on the employed PUF. Depending on the cryptographic hardness of the PUF, our protocol can remain secure if $2^{\lambda/2}$ CRPs (or even more) have been read out, as long as a large fraction of the remaining $2^\lambda - 2^{\lambda/2} \approx 2^\lambda$ other CRPs remains relatively hard to predict. The protocol security can then be amplified exponentially by applying the technique of Damgard et al.

Security in the PAM and the bad PUF model. Protocol 8 is not secure in the PAM or the bad PUF model. A malicious receiver with posterior access to the PUF can learn both strings s_0 and s_1, and the same holds for a malicious receiver employing a simulatable PUF. The attacks are very similar to the attacks in Appendices C and D. The details are straightforward and omitted for space reasons.

3.3 Bit Commitment

It is also possible to devise a BC protocol based on PUFs and interactive hashing. Our upcoming protocol is the first direct BC-construction that relies on the unpredictability of the PUF alone. Earlier approaches utilized the non-invertability of Physical One-Way Functions [32], or reduced BC to OT [7, 8].

Let P be a PUF with $g_P : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Let the employed interactive hashing scheme be the four message IH scheme of Ding et al. [4], whose security is described in Lemma 3. We assume that the sender (=committer) initially holds a bit b and the PUF. Our protocol works as follows.

Protocol 8: PUF-BASED BIT COMMITMENT IN THE STAND-ALONE, GOOD PUF SETTING

Commit Phase:

1. The sender uniformly chooses a random challenge c and applies it to the PUF, obtaining the response r.
2. The sender transfers the PUF to the receiver.
3. The sender and receiver execute an IH protocol, where the sender has input c. Both get output c_0, c_1. Let i be the value where $c_i = c$.
4. The sender sends $b' = b \oplus i$ to the receiver.

Reveal Phase:

1. The sender sends i, r to receiver.
2. The receiver challenges the PUF with c_i and verifies if the response he obtains is equal to r.

Security in the Stand-Alone, Good PUF Model. The security analysis of the BC-Protocol 8 in the stand-alone, good PUF model is very similar to Protocol 4, whence we only sketch it. The perfect concealing property follows from a proof similar to that of Lemma 5: Lemma 3 implies that value i is uniform and unknown to the receiver. So, b' does not give away any information about whether $b = 0$ or $b = 1$.

With respect to the binding property: If the sender wants to commit to both $b = 0$ and $b = 1$, then he must be able to guess both r_0 and r_1. However, the PUF is not in the sender's possession in the IH step of the protocol. Therefore, the proof of Lemma 6 is applicable and the binding property holds with probability $\geq 1 - \sqrt{40\lambda\epsilon}$ for the parameter selection in Lemma 6.

Security in the PAM and the bad PUF model. We observe that Protocol 4 is still secure in the PAM, i.e., if the sender and receiver can only access the PUF after the protocol's end (=after the end of the reveal phase). The reason is that nothing needs to remain secret in a BC protocol after the reveal phase. The protocol is no longer secure if the sender can access the PUF before the reveal phase (the binding property gets lost), but this access *during* the protocol is not allowed in the PAM. Furthermore, the binding property vanishes if the sender uses a simulatable PUF. The details of the attacks are relatively similar to the attacks in Appendices C and D, and are omitted for space reasons. General impossibility proofs that include the insecurity of the above protocols in the PAM and bad PUF model are presented in Section 5.

3.4 Key Exchange

Let us finally give a protocol for PUF-based key exchange in the stand-alone setting. Generally, PUF-based KE protocols have been around as folklore in the community for quite some time. The earliest mentioning of "key establishment" as a PUF-application to our knowledge was made by Pappu et al. in 2002 [33]. The first concrete protocol for PUF-based KE was probably given by van Dijk in 2004 [12]. Brzuska et al. describe a similar KE protocol in a UC-setting in 2011 [7]. The protocol that we provide below slightly deviates from earlier approaches, as it does not assume an authenticated physical channel. More precisely, we assume the following communication channels between Alice and Bob:

1. A binary channel, which is authenticated.

2. A physical channel, over which the PUF is transferred. It is assumed to be insecure [4] in the following sense:

   — In the good PUF model, the PUF be accessed by the adversary for CRP measurements or exchanged against another good PUF by him.

   — In the bad PUF model, the adversary is potentially allowed not only to access the PUF, but also to manipulate it arbitrarily or to exchange it against a bad PUF.

Now, let P be a PUF with $g_P : \{0,1\}^\lambda \to \{0,1\}^\lambda$, which is originally held by Alice. Our key exchange protocol works as follows.

Protocol 9: PUF-BASED KEY EXCHANGE IN THE STAND-ALONE, GOOD PUF SETTING

1. Alice chooses two challenges c and c^* uniformly at random, and measures the two corresponding responses r and r^*.
2. Alice sends the PUF P to Bob.
3. When Bob receives a PUF, he sends a message "Got it!" over the binary channel.
4. Upon receipt of this message, Alice sends $(c, r), c^*$ to Bob.
5. Bob applies the challenges c to the PUF he received in Step 3. If the obtained response is unequal to r, he aborts.
6. *(Applies only in the super-bad PUF model):* Bob shields the PUF from any external communication for the rest of the protocol.
7. Bob applies the challenge c^* to the PUF, obtaining response r^*.
8. Alice and Bob derive a key from the response r^*, which is now known to both of them.

The security in the stand alone, good PUF model follows almost immediately: The adversary can only predict r^* with probability ϵ. Furthermore, he can only exchange the PUF against another good PUF without being detected with equally small probability.

It is relatively easy to see that the protocol is no longer secure in the PAM: An adversary eavesdrops the binary communication in the protocol and learns c^*. He applies c^* in his posterior access phase to the PUF, obtains r^*, and derives the same key.

Interestingly, the protocol is still secure in the stand-alone, bad PUF model. To see this, convince yourself that Alice and Bob will not benefit from using bad PUFs in this setting: They fight a joint adversary, and the PUF will not be re-used by other parties in the stand-alone setting. The adversary cannot replace the PUF against a new, possibly bad PUF due to the authentication step. Please note also that a standard challenge-logging PUF will not help the adversary, since he will not have access any more to the PUF and hence cannot read out the challenge logger (recall that we are in the stand alone model). If super-bad PUFs are allowed, the adversary could attempt the following strategy: He might add a PUF-response transmitter, which does not change the input-output behavior of the PUF, but transmits wirelessly all responses obtained from the PUF by other parties to the malicious party. However, this is prevented by the shielding step 6.

4 Formal Foundations for the Upcoming Impossibility Proofs

We now turn to the second part of the paper, in which we prove a number of impossibility results for protocols whose security is solely based on PUFs. Before we can lead these proofs in Section 5, we will lay the mathematical foundations in this Section 4. The main observation behind our impossibility proofs is that ideal PUFs with multi-bit outputs (such as the PUFs used in [7, 8] and in this paper) bear some similarity with random oracles. It is long

[4] Please note that insecure physical channel, for example the possibility to exchange PUFs, marks one difference between our protocol and the protocol of Brzuska et al. [7]. In the latter, it is assumed that also the physical channel is authenticated, and that the PUF is somehow certified.

known that the power of the latter for implementing cryptographic protocols is limited, as stated in the well-known result of Impagliazzo and Rudich [22].

Before this observation can lead to a formal impossibility proof, several non-trivial problems need to be solved, however. First, the mathematical concepts relevant to the work of Impagliazzo and Rudich need to be adapted to PUFs, which is a subtle and tricky task. It involves a clarification of questions like: What is the view of a player in a PUF-protocol? While the knowledge of players in binary protocols continuously increases (they can record and store all bit strings they have ever seen), the knowledge or ability of players in PUF protocols may decrease once they give away the PUFs they previously held in possession. How can this property of the view in PUF-protocols be modeled?

Another central problem in the application of Impagliazzo-Rudich to PUFs is that a random oracle can be accessed by all parties continuously throughout a protocol, while a PUF can only be accessed by the player who currently holds possession of it. This issue seems so severe at first sight that it prevents the application of Impagliazzo-Rudich to PUFs at all. We circumvent it by proving the impossibility of PUF-protocols in a semi-honest setting [5], and by exploiting the specific properties of our two attack models: Both in the bad PUF model and in the PAM, we eventually arrive in a situation where only the accumulated knowledge and access to the PUFs *at the end of the protocol* is relevant, and where the adversary has got free access to this accumulated knowledge (details follow below). Nevertheless, this proof strategy still requires a formal definition of semi-honest behavior in the context of PUFs and of several other formal notions, which is the purpose of this section.

4.1 Physical Unclonable Functions and Random Oracles

Meaningful cryptographic protocols that use PUFs should at least be secure if the employed PUFs have ideal input-output complexity and randomness. This case occurs if the input and output behavior of the PUF is perfectly random, or similar to a random permutation oracle. Any impossibility proofs which hold for such idealized PUFs (as the proofs in the upcoming sections) carry over to all reasonable and possibly weaker PUF-definitions, since these definitions will include PUFs with perfect randomness as special cases. This motivates the following mathematical model for "ideal" PUFs.

Definition 10 (PUF-Families)**.** *We say that $\mathscr{P} = (M_\lambda)_{\lambda\in\mathbb{N}}$ is a PUF-family if each $M_\lambda = \{P_1^\lambda, \ldots, P_{k_\lambda}^\lambda\}$ is a finite set of PUFs, each of which has challenge set $\{0,1\}^\lambda$.*

Definition 11 (Ideal PUF Model)**.** *A family of PUFs $\mathscr{P} = (M_\lambda)_{\lambda\in\mathbb{N}}$ is called an ideal family of PUFs if:*

1. *For all $\lambda \in \mathbb{N}$, each PUF in M_λ has challenge and response set $\{0,1\}^\lambda$.*
2. *For any probabilistic polynomial time (in λ) algorithm D, and for sufficiently large λ, the advantage by which D can distinguish between a random permutation oracle and an oracle for the function g_{P^λ}, where P^λ is uniformly drawn from all PUFs in M_λ, is negligible in λ.*

We would like to comment that it is necessary to work with an asymptotic treatment here, since Impagliazzo-Rudich is formulated in such a manner. This poses no restriction to our results.

4.2 Physical Unclonable Functions in Protocols

In this section, we need to clarify a few notions that are related to the use of PUFs in cryptographic protocols. They include the concept of a protocol that is solely based on PUFs, the views of the parties in PUF protocols, and the meaning of semi-honest behavior in a PUF-protocol.

Definition 12 (PUF-Protocols)**.** *A two-party protocol Π is called a two-party PUF-protocol if the parties have a binary channel and a physical channel (over which they can exchange physical objects) at their disposal, and if at least one of the parties at least once has a PUF in his possession during the protocol.*

A two-party protocol Π is solely based on a family of PUFs $\mathscr{P} = (M_\lambda)_{\lambda\in\mathbb{N}}$ (for the securiy parameter λ) if

(i) It is a PUF-protocol,

[5] Please note that this is no restriction: If a PUF-protocol is already provably impossible when the parties behave semi-honestly, it is even more so when the parties may act fully dishonestly.

(ii) the parties can draw PUFs uniformly at random from the set M_λ, where λ is the security parameter, and

(iii) no other ideal functionality or computational assumption (like, e.g., trapdoor one-way permutations) and no other set-up assumptions (such as an initial common reference string) are used.

We note that we will often leave away the explicit reference to the security parameter λ, and will simply write that a protocol *"is solely based on a family of PUFs $\mathscr{P}$"*.

As discussed at the beginning of Section 4, we require a definition of semi-honest behavior in the context of PUFs for our impossibility proofs. It is perhaps interesting to start by having a look at the standard definition of semi-honest behavior in the deterministic case from Goldreich [19], which is given in Appendix A. In this standard semi-honest model without transfer of physical objects (where the adversary follows the protocol with the exception that it keeps the values of all its intermediate computations), the joint views of the adversary and the honest player must be close to a simulated view in the ideal model. Therefore, it is not important at what time the adversary knew a value, but only that he knows it in the end.

In PUF-protocols, however, we are not only interested in a party's knowledge of (digital) values, but also in indirect knowledge of values that a party can obtain by querying PUFs while they are in the party's physical possession. Therefore, a party's view does not only contain all known digital values but also knowledge on the PUFs in his possession. Since PUFs are transfered between parties during a protocol, the possible knowledge that can be acquired by the adversary at a given point in time (as represented by his view) may decrease. This is in strong opposition to the semi-honest model without PUFs, since a party can always keep a copy of any binary string that is transfered. This implies that in protocols where physical objects such as PUFs are transfered, the adversary's final view of what he could have computed using the PUFs during the times they were in his possession is most relevant.

The following Definition 13 stipulates a notation for recording which physical objects (PUFs) each party possesses at each point in time. This leads to an extended definition of the view of each party in the definition. By using the extended definition of views, the knowledge each party could possibly acquire while in possession of physical objects can then be put down in Definition 14.

Definition 13 (Re-defining the view in PUF-protocols)**.** *Let Π be a PUF-protocol solely based on a familiy of PUFs $\mathscr{P}$. The distribution of PUFs during the execution of Π on input (x, y) is represented as follows: Let t_0 be the time at which the execution of Π on (x, y) starts. We partition the time axis for the first (resp., second) party in intervals $[t_0, t_1]$, $[t_1, t_2]$, ..., $[t_{n-1}, t_n = \infty)$, where t_i corresponds to the time of the i-th transmission of a message by the first (resp., second) party to the second (resp., first) party. Let $S_i \subseteq \{1, 2, \ldots, k_\lambda\}$, $1 \leq i \leq n$, represent the indices of the PUFs in $M_\lambda = \{P_1^\lambda, \ldots, P_{k_\lambda}^\lambda\}$ that were at one time or another available to the first (resp., second) party during the time interval $[t_{i-1}, t_i]$.*[6]

Let set T_i, $1 \leq i \leq n$, represent the challenge-response pairs collected during the protocol execution in interval $[t_{i-1}, t_i]$ by the first (resp., second) party . Set T_i contains triples $(j, c, g_j(c))$, where we write for simplicity g_j instead of $g_{P_j^\lambda}$.

Without loss of generality, we assume that the i-th message m_i, $1 \leq i \leq n$, received by the first (resp., second) party is in interval $[t_{i-1}, t_i]$ (here m_1 or m_n may equal the empty message). The final `view` *of the first (resp., second) party is redefined as* $\texttt{view}_1^\Pi = (x, r, m_1, S_1, T_1, \ldots, m_n, S_n, T_n)$ *(resp.,* $\texttt{view}_2^\Pi = (y, r, m_1, S_1, T_1, \ldots, m_n, S_n, T_n)$*), where r represents the outcome of the first (resp., second) party's internal coin tosses.*[7]

By $S^\Pi(x, y)$ we denote the set of indices corresponding to the actual physical objects used in the execution of Π on (x, y).[8]

We will now define into what extent parties can extract knowledge from the physical objects that are in their possession. We say that a semi-honest party still complies with a protocol Π that involves transfer of physical objects if he queries physical objects that are in his possession in order to gain extra knowledge. Since a

[6] Since one protocol execution may interfere (be executed in parallel) with other protocol executions by the same and other parties, a PUF may at one time or another be in possession of both the first and the second party during an interval $[t_{i-1}, t_i]$. If there is no interference with other protocol executions, then S_i for the first party is disjoint from S_i for the second party.

[7] The final view needs to record all the challenge-responses that were queried before, since a physical object that was queried may not be accessible after the end of the protocol execution.

[8] Since each party may at one time or another have possession of physical objects that are not used during the execution of Π, sets S_i are not necessarily subsets of $S^\Pi(x, y)$. We notice that PUFs in the PAM are accessible by both parties at the end of the protocol execution in which case $S^\Pi(x, y) \subseteq S_n$ for both parties.

party's internal coin tosses can be computed before the start of protocol Π, the first (resp., second) party can use the physical objects in S_i to compute new challenge response pairs based on $(x, r, m_1, T_1, \ldots, m_i, T_i)$ (resp., $(y, r, m_1, T_1, \ldots, m_i, T_i)$). For $1 \leq i \leq n$, let z_i denote the set of challenge response pairs gathered up to the time right before m_i has been received. Let $z_0 = \emptyset$. Notice that $z_{i-1} \subseteq z_i$ and $T_i \subseteq z_i$. Based on $(x, r, m_1, \ldots, m_i, S_i, z_{i-1})$ (resp., $(y, r, m_1, \ldots, m_i, S_i, z_{i-1})$) the first (resp., second) party may use a probabilistic polynomial time algorithm A_i with oracle access to $\{g_j\}_{j \in S_i}$ to compute and add new challenge-response pairs to z_i. This leads to the following definition.

Definition 14 (Knowledge extraction)**.** *Let $M_\lambda = \{P_1^\lambda, \ldots, P_{k_\lambda}^\lambda\}$ be a set of PUFs with set of associated functions $\mathcal{G} = \{g_1, \ldots, g_{k_\lambda}\}$.* [9] *A probabilistic polynomial time algorithm K_1 with oracle access to $\mathcal{G}$, denoted $K_1^{\mathcal{G}}$, is called a knowledge extraction algorithm for the first party if it is composed of algorithms A_1, A_2, etc., such that on input $\mathtt{view}_1^\Pi(x, y) = (x, r, m_1, S_1, T_1, \ldots, m_n, S_n, T_n, \mathcal{S})$ it iterates, for $1 \leq i \leq n$ and $z_0 = \emptyset$,*

$$z_i \leftarrow A_i^{\{g_j\}_{j \in S_i}}(x, r, m_1, \ldots, m_i, S_i, z_{i-1}) \cup z_{i-1} \cup T_i$$

and outputs $(x, r, m_1, \ldots, m_n, z_n)$. In a similar way, we define knowledge extraction algorithms K_2 for the second party.

Based on knowledge extraction, we are now able to define semi-honest behavior with transfer of physical objects. The definition is subtle in that it almost resembles the definition of privacy in the semi-honest model without transfer of physical objects for oracle-aided protocols (see Appendix A). The difference is that computational indistinguishability is now defined with respect to how knowledge can be extracted from a view: A machine D that distinguishes a simulated view from a real view does not have access to all functionalities in $\mathcal{G}$, it may only use knowledge extractors. We model this by first explicitly transforming a real view by using a knowledge extractor after which D proceeds its computation without access to $\mathcal{G}$. (The simulator algorithms do have access to $\mathcal{G}$ in order to obtain challenge-response pairs that cannot be distinguished from the ones recorded in the views.)

Definition 15 (Privacy w.r.t. semi honest behavior in the deterministic case with PUFs)**.** *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a deterministic functionality, and denote the first (resp., second) output of $f(x, y)$ by $f_1(x, y)$ (resp., $f_2(x, y)$). Let Π be a two-party protocol for computing f during which PUFs from a set M_λ with associated functions in a set $\mathcal{G}$ are transferred.*

We say that Π privately computes f if there exist probabilistic polynomial time algorithms, denoted S_1 and S_2, such that, for all knowledge extraction algorithms for the first and second party[10]*, denoted K_1 and K_2,*

$$\begin{aligned} \{S_1^{\mathcal{G}}(x, f_1(x, y))\}_{x,y \in \{0,1\}^*} &\stackrel{c}{\equiv} \{K_1^{\mathcal{G}}(\mathtt{view}_1^\Pi(x, y))\}_{x,y \in \{0,1\}^*} \\ \{S_2^{\mathcal{G}}(x, f_2(x, y))\}_{x,y \in \{0,1\}^*} &\stackrel{c}{\equiv} \{K_2^{\mathcal{G}}(\mathtt{view}_2^\Pi(x, y))\}_{x,y \in \{0,1\}^*} \end{aligned}$$

where $|x| = |y|$.

Based on these definitions, we have laid the foundation for the impossibility results in the next section.

5 Impossibility Results

In this section, we will prove impossibility results for certain two-party protocols in the posterior access and bad PUF models. They apply to a semi-honest behavior in these two models. Please note that this is no restriction: If a protocol is impossible already if the parties behave semi-honestly, it is even more so in the case the parties are fully malicious.

5.1 Impossibility of OT and KE in the Posterior Access Model

In the semi-honest model, we model posterior access by giving each party access to all the used PUFs *after* each execution of a protocol (or protocol (sub-)session, if there are any):

[9] As in Definition 13, we write g_i instead of $g_{P_i^\lambda}$.

[10] Here, computational indistinguishability is defined with respect to machines that do not have oracle access to $\mathcal{G}$.

Definition 16 (Privacy w.r.t. the posterior access model)**.** *Let* Π *be a two-party protocol solely based on a family of PUFs* $\mathscr{P}$ *for privately computing* f*. Let* Π' *be the protocol which on input* (x, y) *executes* Π *after which all the PUFs corresponding to* $S^{\Pi}(x, y)$ *are made physically accessible to the first party and the second party. We say* Π *privately computes* f *solely based on* $\mathscr{P}$ *in the posterior access model if protocol* Π' *privately computes* f*.*

The following lemma reduces protocols that use PUFs in the PAM to protocols that use a random oracle.

Lemma 17. *Let* Π *be a two-party protocol solely based on a family of PUFs* $\mathscr{P}$ *for privately computing* f*. If* Π *privately computes* f *in the posterior access model, then protocol* Π*, where each query to a PUF is replaced by a call to a random oracle, privately computes* f*.*

Proof. Let λ be the security paremeter, and let $\mathcal{G} = \{g_1, \ldots, g_{k_\lambda}\}$ be the set of associated functions to the set of PUFs M_λ of $\mathscr{P}$. Let A_1, A_2, ..., define a knowledge extraction algorithm K. Since the PUFs that are used in Π are uniformly drawn from M_λ, and since outputs of different PUFs are uncorrelated, we may restrict A_i's oracle access to $\{g_j\}_{j \in S_i}$ to oracle access to the subset $\{g_j\}_{j \in S_i^*}$, where S_i^* represents the PUFs in S_i that were actually used in the protocol execution.

Since Π privately computes f with respect to PUFs in the PAM, we assume that, without loss of generality, for all inputs (x, y), a protocol execution Π on (x, y) has the property that $S^{\Pi}(x, y)$ is a subset of S_n in the view of the first party as well as a subset of S_n in the view of the second party, hence, $S_i^* \subseteq S_n$. Therefore, without loss of generality, the knowledge extraction algorithm K may postpone all the computations in A_1, A_2, to A_{n-1} till the very end in A_n. In other words, K effectively takes as input a non-extended view $(x, r, m_1, \ldots, m_n)$ (resp., $(y, r, m_1, \ldots, m_n)$, re-computes all the challenge-response pairs T_1, ..., T_n that were used during the protocol execution, and computes the new challenge-response pairs in z_n by using oracle access to $S_n^* = S^{\Pi}(x, y)$. Since outputs of different PUFs are uncorrelated, we may as well give K oracle access to all of $\mathcal{G}$ (since this will not help a machine that attempts to distinguish simulated views from real views).

The non-extended view $(x, r, m_1, \ldots, m_n)$ (resp., $(y, r, m_1, \ldots, m_n)$) are the views of a protocol Π' that proceeds as in Π but without transfer of physical objects and where queries to physical objects are replaced by oracle access to $\mathcal{G}$. So, Definition 15 holds for any probabilistic polynomial time algorithms K_1 and K_2 with oracle access to $\mathcal{G}$ where $\mathtt{view}_1^{\Pi}(x, y)$ and $\mathtt{view}_2^{\Pi}(x, y)$ are replaced by $\mathtt{view}_1^{\Pi'}(x, y)$ and $\mathtt{view}_2^{\Pi'}(x, y)$. Now, we may discard K_1 and K_2 if we allow the distinguisher oracle access to $\mathcal{G}$. This corresponds to Definition 13 for an oracle-aided protocol with access to oracle $\mathcal{G}$. According to the PUF model, $\mathcal{G}$ cannot be distinguished from the random oracle, which proves the lemma. □

The Impagliazzo-Rudich result says that there are no black-box implementations of OT and KE from the random permutation oracle[11] [22, 14]. Together with the previous lemma this proves:

Theorem 18. *There does not exist a two-party protocol for privately computing oblivious transfer (OT) solely based on an ideal family of PUFs in the posterior access model.*

We notice that the parties in KE collaborate in order to obtain a shared key; they will execute the KE protocol without cheating. Therefore, since this section assumes semi-honest behavior by the parties that execute protocols between themselves, the analysis in this section does not apply to KE.

In KE we consider a third party: the adversary, who is intercepting and resending the communication between the two honest parties. In the PAM the adversary has access to the used PUFs after the protocol execution. So, in the semi-honest model with transfer of physical objects (generalized in the natural way to multiple parties) the views of the honest parties and the adversary are the same as in the KE protocol where all queries to PUFs are replaced by calls to a random oracle (a detailed proof of this statement is similar to the proof of Lemma 17). Since this is impossible by Impagliazzo-Rudich, we obtain

Theorem 19. *There is no secure two-party key exchange (KE) protocol solely based on an ideal family of PUFs in the posterior access model.*

We notice without proof that KE based on so-called erasable PUFs [41] in the posterior access model is possible: An erasable PUF [41] is a PUF that allows its owner to selectively erase responses to single challenges without affecting the responses to the other challenges. [12] Since an erasable PUF changes its input-output behavior over

[11] See [14, 35, 6] for KE protocols based on OT.

[12] Again note the similarity with an established concept in classical cryptography here, namely with the idea of a reusable common reference string with erasing parties [10].

time, it does not satisfy Definition 11 and is therefore not covered by the theorem. KE based on an erasable PUF P can be constructed in a straightforward way as follows. For a random challenge c, Alice obtains a response r from P. Alice transmits P to Bob, who acknowledges receipt of P. Once the acknowledgement is received, Alice transmits c to Bob, who obtains r from P and erases the challenge-response pair (c, r) from P. An adversary only knows c at the end of the protocol when P is not in his possession, after the protocol execution the adversary may gain access to P, however, r has been erased in the meantime. Similarly, OT based on erasable PUFs in the posterior access model is possible, too.

5.2 Impossibility of OT in the Stand-Alone, Bad PUF Model

We continue by showing a general impossibility result on OT in the stand-alone, bad PUF model: If malicious parties are allowed to use bad PUFs that are simulatable and challenge-logging at the same time, then there are no protocols that securely implement OT by using at most $O(\log \lambda)$ different PUFs. The proof even applies to the stand-alone setting, which is strictly weaker than a standard UC-setting.

Our argument works as follows. We say that a two-party protocol Π privately computes f even if it is based on a family of bad PUFs if (1) Π can be solely based on a family of (proper) PUFs and privately compute f, and (2) Π still privately computes f if the parties use bad PUFs. More specifically, we consider adversarial behavior of the following form: The adversary always follows Π as the honest player; but instead of honestly producing PUFs, he always chooses a random one-way permutation g and produces a bad PUF that implements g and has at the same time a challenge-logger as described in Section 2. Whenever he gets a PUF back, he reads out all the challenges from the logger. Whenever the PUF is not in his possession, he still has access to the input-output functionality of the PUF, as he knows g. Let the set of all adversaries of this form be $\mathcal{A}$.

We now change protocol Π in the following way:

a) If a PUF always ends up on the side of the party who created the PUF, then we replace the PUF by queries. So instead of sending the PUF to the other party, this other party sends the challenges he wants to learn to the creator of the PUF, who responds with the corresponding PUF-response.

b) If a PUF always ends up on the party's side who did not create it, we replace the PUF by a random oracle. This means that both parties have an oracle implementing the same function as the PUF already at the beginning of the protocol, and do not need to transfer the PUF.

Let this modified protocol be Π'.

Lemma 20. *Let Π be a two-party protocol solely based on an ideal family of PUFs $\mathscr{P}$ for privately computing f. If Π privately computes f solely based on $\mathscr{P}$ in the bad PUF model and if every PUF used in Π always ends up on the same side, then Π' privately reduces[13] f to the random oracle.*

A detailed proof is given below. We first provide a sketch: A semi-honest adversary in the random permutation oracle model follows the protocol, but may ask some additional queries to the permutation oracle before outputting his view. We want to show that for any such adversary A, there exists an adversary $A' \in \mathcal{A}$ for the protocol Π such that the joint views of the adversary and the honest party are identical in both settings.

- For the PUFs replaced in step a): For all PUFs where the adversary is not the creator, the two settings are identical. For the PUFs where the adversary is the creator, A will learn the queries of the other party to the PUF. But since we assume that he always gets the PUF back and that he installed a logger, A' will eventually also learn the queries.
- For the PUFs replaced in step b): A may ask the permutation oracle some extra queries during or at the end of the protocol. But since A' always has a copy of the PUF at the end of the protocol, he may at the end of the protocol ask the same queries as A.

Therefore, the views of the two parties at the end of the protocol are the same in both settings, which implies the statement.

[13] We refer to Appendix A for a formal definition of privately reducing a function to an oracle.

Proof. In order to prove that Π' privately reduces f to the random permutation oracle, we show that its views are identical to those of Π in which each PUF is replaced by a bad PUF. Then, since Π privately computes f even if based on a family of bad PUFs, the lemma follows.

Let P^1 (resp., P^2) be the set of bad PUFs created by the first (resp., second) party in Π. Let $D^1 \subseteq P^1$ (resp., $D^2 \subseteq P^2$) be the subset of bad PUFs that ends up at the second (resp., first) party. Notice that, since the PUFs are simulatable, both parties have access to the functionalities of $D^1 \cup D^2$ at the end of protocol Π.

The bad PUFs in $P^1 \setminus D^1$ end up at the first party and are replaced by queries in protocol Π', i.e., the second party replaces the PUF queries by queries to the first party. The view of the second party does not change due to this substitution; for any knowledge extraction algorithm K_2,

$$K_2(\mathtt{view}_2^{\Pi}) = K_2(\mathtt{view}_2^{\Pi"}), \tag{3}$$

where Π" is the protocol where the second party replaces the PUF queries to $P^1 \setminus D^1$ by queries to the first party.

The view of the first party does change. However, since the first party has access to $P^1 \setminus D^1$, a knowledge extraction algorithm can access the corresponding challenge loggers at the end of the protocol by using the secret bit strings (access challenges) for accessing the loggers (this is possible because, these secret bit strings are recorded in the view of the first party). This means that such a knowledge extraction algorithm reproduces the queries made by the second party. So, for such key extraction algorithms K_1,

$$K_1(\mathtt{view}_1^{\Pi}) = K_1(\mathtt{view}_1^{\Pi"}). \tag{4}$$

By a similar argument, for all knowledge extraction algorithms K_1 and K_2, that access the loggers in $P^1 \setminus D^1$ and $P^2 \setminus D^2$ resp., equations (3-4) hold for Π" defined as protocol Π where the second party replaces the PUF queries to $P^1 \setminus D^1$ by queries to the first party and where the first party replaces the PUF queries to $P^2 \setminus D^2$ by queries to the second party. Therefore, since Π privately computes f even if based on a family of bad PUFs, also Π" privately computes f even if based on a family of bad PUFs.

Notice that, at the end of protocol Π", $S^{\Pi"}(x, y) = D^1 \cup D^2$ since the PUFs in $P^1 \setminus D^1$ and $P^2 \setminus D^2$ are replaced by queries. Since both parties have access to the functionalities of $D^1 \cup D^2$ at the end of protocol Π", Π" also privately computes f if solely based on PUFs in the PAM. Hence, we may apply Lemma 17 from which the lemma follows. □

Lemma 20 does not cover a third kind of PUFs, however: PUFs for which it is not known at the beginning of the protocol on which side the PUF will end up. In order to close this gap, we change protocol Π in the following way:

Suppose that Π uses a fixed number of $O(\log \lambda)$ PUFs. For simplicity, we assume that both the first and second party draw/create m PUFs each.[14] Then, for all inputs (x, y) and every protocol execution Π on (x, y), $|S^{\Pi}(x, y)| = 2m = O(\log \lambda)$. We use Π to obtain a protocol Π^* whose aim is to evaluate f for some random input (hence, Π^* does not take any input itself, the random input is constructed by the two parties during the execution of Π^*):

a) By coin tossing, the two parties create $n = poly(\lambda)$ random inputs $\{(x_i, y_i)\}_{1 \le i \le n}$. For each random input they execute protocol Π where each PUF challenge is replaced by a call to a random oracle. During the protocol execution they track for the first (resp., second) party where the i-th PUF drawn/created by the first (resp., second) party would have ended up. Since the parties follow the protocol honestly and since proper PUFs cannot be distinguished from random permutations, the distribution among the two parties of the $2m$ "tracked PUFs" at the end of the execution has the same statistics as the distribution of PUFs in Π for random inputs. Since $2m = O(\log \lambda)$, the $n = poly(\lambda)$ executions are sufficient to estimate the most likely distribution; at least a distribution D that occurs with probability $p(D) \ge (1/2^{2m})/2 = 1/O(poly(\lambda))$ can be estimated.

b) Let Π' be the protocol as defined before for Π by assuming D, the likely distribution of PUFs computed as a result of the previous phase. Again, the two parties create $n = poly(\lambda)$ random inputs $\{(x_i, y_i)\}_{1 \le i \le n}$. For each random input they now execute protocol Π'. During the execution of Π' they again track for the first (resp., second) party where the i-th PUF drawn/created by the first (resp., second) party would

[14] If for some protocol executions less than m PUFs are drawn/created by one of the parties, then we simply append to the protocol additional steps during which the parties draw/create extra PUFs.

have ended up. If the resulting distribution equals D for some input (x_i, y_i), then the parties conclude the protocol with output $(x_i, f_1(x_i, y_i))$ for the first party and $(y_i, f_2(x_i, y_i))$ for the second party. Since $p(D) = 1/O(poly(\lambda))$, the protocol will find such an input (x_i, y_i) with high probability.

The above protocol Π^* is a probabilistic $poly(\lambda)$ time protocol using a random oracle (and no PUFs) that evaluates f for some random input, in other words, Π^* computes *randomized* f. Since the parties follow Π^* honestly, Lemma 20 proves:

Lemma 21. *Let Π be a two-party protocol solely based on an ideal family of PUFs $\mathscr{P}$ for privately computing f. If Π privately computes f solely based on $\mathscr{P}$ in the bad PUF model by using at most $O(\log \lambda)$ different PUFs during each protocol execution, then Π^* privately reduces randomized f to the random permutation oracle.*

We notice that [5] proves: if Π^* privately reduces randomized OT to the random permutation oracle, then there exists a protocol that privately reduces OT to the random permutation oracle. The Impagliazzo-Rudich result says that there are no black-box implementations of OT from the random permutation oracle [22, 14]. Together with the previous lemma this proves:

Theorem 22. *There does not exist a two-party protocol solely based on an ideal family of PUFs in the bad PUF model for privately computing oblivious transfer (OT) by using at most $O(\log \lambda)$ different PUFs in each protocol execution.*

We notice that randomized KE is equivalent to KE (since the output of randomized KE can be used as the agreed upon key). The parties in KE collaborate in order to obtain a shared key, therefore, they will execute the KE protocol without cheating. It does not make sense for the two parties to create and use bad PUFs, and if they do, they will use the challenge-loggers to achieve KE more directly. Therefore, since this section assumes semi-honest behavior by the parties that execute protocols between themselves, the analysis in this section does not apply to KE. In KE we consider a third party: the adversary, who is intercepting and resending the communication between the two parties. KE in a stand-alone, bad PUF model thus remains secure, since the parties have no incentive to use bad PUFs. KE with posterior access (i.e., in multiple sequential protocol executions) and with bad PUFs is impossible, since KE is already impossible in the posterior access model alone (as shown in the previous section).

6 The Case of Bit Commitment

We now turn to the third part of the paper, in which we analyze PUF-based bit commitment in several scenarios. It turns out that BC is special in a number of aspects. One reason is that at the end of a BC protocol (i.e., at the end of the reveal phase), nothing needs to remain secret. In OT and KE protocols, to the opposite, certain information must be kept secret forever: The "other" string s_{1-b} and the choice bit b in OT, and the exchanged key in KE. This allows secure BC protocols in circumstances where secure OT and KE are provably impossible.

For space reasons, we merely sketch our constructions and security arguments in this section. In particular, we would like to stress that we assume that a random one-way function can be derived from the unpredictability of the PUF without giving an explicit construction to this end. Methods how to achieve this may be given in future versions of this paper.

6.1 Impossibility of BC in the Bad PUF Model with Access before the Reveal Phase

We start by showing that there are also scenarios in which BC based solely on PUFs is provably impossible. This holds, for example, if malicious parties are allowed to use simulatable and challenge-logging PUFs, and if they have access to all employed PUFs before the reveal phase. Similar as in the proofs in Section 5, we may then replace each challenge to a PUF by a challenge to the party who created the PUF. [15] The resulting protocol achieves BC without access to a random oracle, without any set-up assumption and without access to any other ideal functionality. This means that the resulting protocol achieves BC unconditionally, which is not possible according to [13].

[15] Since the PUF is accessible after each message transmission, the party who created the PUF is able to read out the challenge-logger. So, replacing each challenge to a PUF by a challenge to the party who created the PUF keeps the views of both parties identical to that of Π.

Theorem 23. *There does not exist a two-party protocol for privately computing bit commitment solely based on an ideal family of PUFs $\mathscr{P}$ in the bad PUF model if the malicious party has got access to the used PUFs before the reveal phase.*

6.2 Secure Bit Commitment in the Posterior Access, Bad PUF Model

We present below a BC scheme that is secure in the posterior access, bad PUF model. More precisely, it is secure under the following presumptions:

- Bad PUFs (including SIM- and CL-PUFs) may be used be by the malicious players, but no super-bad PUFs are allowed. That is, the input-output behavior of the PUFs cannot be remotely accessed and altered, the challenges applied to the PUF are not communicated remotely (i.e., to parties without physical access to the PUF), and the CRP-behavior of the PUF does not change automatically after some time interval.
- The malicious party can physically access the PUF after the reveal phase (i.e., after the end of a protocol (sub-)session), as standard in the PAM), but not before the start of the reveal phase.

The scheme is the first PUF protocol in literature which explicitly uses two PUFs. The scheme shows that we are not helplessly extradited to bad PUFs. If careful protocol design and reasonable hardware assumptions come together, security can still be guaranteed. Due to the dual PUF transfer, it mainly has theoretical value; we leave it as an open question if there are protocols whose security is solely based on PUFs, which are secure in the bad PUF model, and which require only one PUF transfer.

We assume that the sender holds a PUF S, and the receiver a PUF R at the beginning of the protocol, and denote by $S(\cdot)$ and $R(\cdot)$ the one-way permutations corresponding to these two PUFs (see remark at the beginning of Section 6). We assume that the sender wishes to commit a bit b.

Protocol 24: BIT COMMITMENT IN THE POSTERIOR ACCESS, BAD PUF MODEL

Commit Phase:

1. The sender chooses at random a challenge c and obtains $S(c)$ by measurement.
2. The sender transmits the PUF S to the receiver. The receiver keeps S in his possession until the reveal phase has been completed.
3. The receiver chooses challenges $c_1, \ldots, c_k$ at random and uses R to measure corresponding responses $r_1, \ldots, r_k$. He transmits R to the sender.
4. The sender obtains $R(c)$ by measurement and chooses a random $y \in \{0,1\}^\lambda$. The sender computes $(y, S(c) \oplus R(c), b \oplus \langle y, c \rangle)$ and transmits this triple to the receiver. The sender keeps R in his possession until the start of the reveal phase.

Reveal phase:

1. The sender transfers (y, c) and R to the receiver.
2. The receiver applies the challenges $c_1, \ldots, c_k$ to R and obtains responses $r_1, \ldots, r_k$. He compares these responses to those measured in step 3 of the commit phase in order to verify that R has not been exchanged against a bad PUF with collisions.
3. The receiver obtains by measurement $S(c) \oplus R(c)$ and checks this against what he received before. The receiver extracts bit b.

With respect to the concealing property of Protocol 24, note that R is not in the receiver's possession before the reveal phase, so any potential challenge-logging functionality cannot yet be used. Since we are in the bad PUF model, we need to assume that the receiver did design the functionality of R such that he can easily compute inverses as well, however.

Since $S(\cdot)$ is a random function, $S(c) \oplus R(c)$ randomizes $R(c)$ completely. This means that even if $R(\cdot)$ is an easy to invert function constructed maliciously by the receiver, $S(c) \oplus R(c)$ does not reveal any information about c. Therefore $\langle y, c \rangle$ randomizes b, since c is random, and y is with overwhelming probability unequal to 0. The concealing property of the scheme follows.

Interestingly, the concealing property does even hold if the parties are allowed "super-bad PUFs", i.e., if they may use "PUFs" whose challenge-response behavior can be fully determined in real-time by wireless communication by the PUF-creator and the PUF. Such super-bad PUFs do not even need to possess response consistency; their responses can be adjusted on the fly by their creator via wireless communication. Nevertheless, Protocol 24 maintains its concealing property even in the presence of super-bad PUFs. The reason is that the sender will always use a random permutation S to guarantee the concealing property in his own interest. The fact that S is a random permutation alone already suffices to make the scheme concealing.

With respect to the binding property of the scheme: If the sender wants to be able to commit to both 0 or 1, then he must be able to generate two different challenges c_1 and c_2 such that:

$$S(c_1) \oplus R(c_1) = S(c_2) \oplus R(c_2),$$

that is,

$$S(c_1) \oplus S(c_2) = R(c_1) \oplus R(c_2).$$

The sender committed himself to the functionality of S before having possession of R. So, we may assume without harming the security of the scheme that by using a bad PUF (which is simulatable) the sender can easily invert S. We may even assume that, for all vectors $v \in \{0,1\}^\lambda$, the sender can easily compute two values c_1 and c_2 such that $S(c_1) \oplus S(c_2) = v$.

However, this does not help the sender: Let $s(x) = s(x_1, x_2) = S(x_1) \oplus S(x_2)$ and $r(x) = r(x_1, x_2) = R(x_1) \oplus R(x_2)$, both be functions from $\{0,1\}^{2\lambda}$ to $\{0,1\}^\lambda$.

Since $r(x)$ looks random in the sender's view, computing a collision $s(x) = r(x)$ is as hard as choosing a vector $y = s(x)$ in $\{0,1\}^\lambda$ and then choosing a random vector z (representing $r(x)$) in $\{0,1\}^\lambda$ and repeat this process until $y = z$. Since z is random, $y = z$ with probability $2^{-\lambda}$. Hence, the sender cannot open the commitment to both 0 and 1. The binding property of the scheme follows.

We notice that the above analysis only holds if PUF S does not change its challenge-response behavior over time. If S is super-bad, then the sender could open the commitment in both ways. Furthermore, if S would automatically (without wireless communication as in a super-bad PUF) change its input-output behavior after some time interval, then the sender could flip his commitment by introducing some delay to the start of the reveal phase.

6.3 Bit Commitment in the Good PUF Model with Access between Commit and Reveal Phase

We conclude with a construction for a PUF-based bit commitment scheme which remains secure even if the sender gains access to the PUF before the reveal phase. We assume that the sender holds a PUF S with challenges and responses of length λ, and denote by $S : \{0,1\}^\lambda \to \{0,1\}^\lambda$ the one-way permutation derived from S (see remark at the beginning of Section 6). Our construction is reminiscent of the well-known Goldreich-Levin approach [20], and of an earlier bit commitment protocol by Pappu (see section 9.3 of [32]).

Protocol 25: BIT COMMITMENT WITH ACCESS BETWEEN COMMIT AND REVEAL PHASE

Commit Phase:

1. The sender holds the PUF S and a bit b. He chooses a random vector $y \in \{0,1\}^\lambda$ and a random challenge $c \in \{0,1\}^\lambda$.

2. The sender transmits the PUF S together with the information $(y, S(c), b \oplus \langle y, c \rangle)$ to Bob (where $\langle \cdot, \cdot \rangle$ denotes the scalar product).

Reveal Phase:

1. The sender sends c to the receiver.

2. The receiver verifies that the information sent to him in step 2 of the the commit phase was correct: Denoting the tuple which he received by (d, e, f), he verifies that $e = S(c)$ by measurement on the PUF, and extracts $b = f \oplus \langle d, c \rangle$. If this is the case, he accepts the reveal phase.

Protocol 25 is concealing even in the PAM, because a receiver who has got access to the PUF before the reveal phase and who wants to recover b must recover $\langle y, c \rangle$ given y and $S(c)$. This means that recovering b with probability greater than $1/2$ is as hard as inverting the random function $S(\cdot)$ derived from the PUF. The protocol is binding since it is hard to produce two different c_0 and c_1 such that $S(c_1) = S(c_2)$. This once more follows from the assumption that $S(\cdot)$ is a random function.

The protocol is not secure in the bad PUF model, however. The sender could create a simulatable PUF such that he knows a pair c_0, c_1 for which the responses r_0 and r_1 collide. As an alternative scheme, the next subsection discusses the security of Protocol 24 which is secure in the bad PUF model.

7 Summary and Conclusions

We examined the use of Physical Unclonable Functions (PUFs) in advanced cryptographic protocols, considering the same type of PUFs that was investigated most earlier work in this topic (e.g. [33, 32, 39, 7, 8]): Namely PUFs with a large (preferably exponential) number of CRPs, whose challenge-response interface is accessible by everyone who holds physical possession of the PUF. Such PUFs have sometimes been referred to as Strong PUFs or Physical Random Functions in the literature.

In the first part of the paper, we presented partly new protocols for OT, KE and BC, which have certain practicality and security advantages over existing schemes. For example, our OT-protocol has constant rounds due to a new interactive hashing step compared to an earlier scheme of Rührmair [38], or does not allow quadratic attacks such as the scheme of Brzuska et al. [7]. We gave a new and relatively simple PUF-definition, which focuses on single PUFs, average case security, and unpredictability with respect to one CRP only. We showed that this definition is useful in leading security proofs for our protocols.

In the second part of the paper, we introduced two new and realistic attack models, the so-called posterior access model (PAM) and the bad PUF model. Both models constitute viable and hard-to-prevent attack strategies in practice, so we argued, and are close to practical PUF usage. We observed that the recently suggested PUF-protocols of Rührmair and Brzuska et al. [7, 8, 38] for oblivious transfer (OT), bit commitment (BC) and key exchange (KE) can be attacked in these two new and realistic models. This posed the question if there might be other PUF-protocols that can withstand the PAM and bad PUF attacks. Our main contribution here is a collection of impossibility results. First, no secure protocols for KE and OT exist in the posterior access model. In a nutshell, the reason is that the responses of the employed PUFs remain accessible in unaltered form and can be read out at later points in time. This highlights an important difference between PUFs and other alternative approaches like the bounded storage model (BSM) [29, 3] or noise-based cryptography [49, 15, 30]. The latter also exploit natural and uncontrollable randomness for cryptographic protocols, but differ from PUFs as some form of information loss is implicit (e.g., part of the broadcast bitstream in BSM is inevitably lost forever). Our results indicate that such information loss is essential for achieving security. Secondly, we moved away from the PAM and considered a stand alone model for protocol execution. Are there secure protocols for PUF-based OT at least in this restricted setting? We proved that this is not the case if the adversary is allowed to use "bad PUFs" that are simulatable and challenge-logging at the same time. Our findings in the bad PUF model stress that PUFs are not easily controllable binary strings, but complex hardware that may possess unwanted additional properties. In a typical two-party protocol such as OT, a PUF originating from a malicious party is nothing else than an untrusted piece of hardware that stems from the adversary.

In the third and final part of the paper, we dealt with PUF-based BC, which is special in a number of aspects. One reasons is that in opposition to OT and KE, at the end of a BC-protocol, nothing needs to remain secret We showed that no secure protocol for BC exist if challenge logging and simulatable bad PUFs are allowed, and if the malicious party has got access to the PUF between the commit and the reveal phase. Furthermore, we provided a construction for BC that is secure in the PAM even if bad PUFs are allowed, and a construction for BC that is

secure in the good PUF model, even if the adversary has got access to the PUF between the commit and the reveal phase.

Our impossibility results illustrate that in order to be applicable as a general cryptographic tool, PUFs require new hardware properties: The responses of the PUFs must be selectively erasable, which is a concept introduced under the name "erasable PUF" in [41]. Secondly, mechanisms for PUF-certification must be developed which should work offline and detect bad PUFs, including bad PUFs that have been created by maliciously adding extra hardware to a proper PUF. This consequence of our results poses new challenges to the PUF hardware community.

Acknowledgements

We would like to thank Jürg Wullschleger and Stefan Wolf for several substantial suggestions and enjoyable discussions on this manuscript.

References

[1] A. Kerckhoff: *La cryptographie militaire*. Journal des sciences militaires, Vol. IX, pp. 5-38, 1883.

[2] Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, François-Xavier Standaert, Christian Wachsmann: *A Formalization of the Security Features of Physical Functions.* IEEE Symposium on Security and Privacy 2011: 397-412

[3] Y. Aumann, Y. Z. Ding, M. O. Rabin: *Everlasting security in the bounded storage model*. IEEE Transactions on Information Theory, 48(6):1668-1680, 2002.

[4] Yan Zong Ding, Danny Harnik, Alon Rosen, Ronen Shaltiel: *Constant-Round Oblivious Transfer in the Bounded Storage Model*. J. Cryptology 20(2): 165-202 (2007)

[5] Donald Beaver: *Correlated Pseudorandomness and the Complexity of Private Computations.* STOC 1996: 479-488

[6] Manuel Blum: *How to Exchange (Secret) Keys (Extended Abstract)* STOC 1983: 440-447.

[7] C. Bruzska, M. Fischlin, H. Schröder, S. Katzenbeisser: *Physical Unclonable Functions in the Universal Composition Framework.* CRYPTO 2011.

[8] C. Bruzska, M. Fischlin, H. Schröder, S. Katzenbeisser: *Physical Unclonable Functions in the Universal Composition Framework.*. Full version of the paper. Available from Cryptology ePrint Archive, 2011.

[9] Ran Canetti: *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. FOCS 2001: 136-145. Full and updated version available from Cryptology ePrint Archive.

[10] Ran Canetti, Marc Fischlin: *Universally Composable Commitments.* CRYPTO 2001: 19-40

[11] Ivan Damgard, Joe Kilian, Louis Salvail: *On the (Im)possibility of Basing Oblivious Transfer and Bit Commitment on Weakened Security Assumptions*. EUROCRYPT 1999: 56-73

[12] Marten van Dijk: *System and method of reliable forward secret key sharing with physical random functions.* US Patent No. 7,653,197, October 2004.

[13] Ivan Damgård, Joe Kilian, Louis Salvail: *On the (Im)possibility of Basing Oblivious Transfer and Bit Commitment on Weakened Security Assumptions.* EUROCRYPT 1999: 56-73

[14] Yael Gertner, Sampath Kannan, Tal Malkin, Omer Reingold, Mahesh Viswanathan: *The Relationship between Public Key Encryption and Oblivious Transfer.* FOCS 2000: 325-335

[15] I. Csiszar, J. Körner: *Broadcast channels with confidential messages.* IEEE Transactions on Information Theory, Vol. 24 (3), 1978.

[16] S. Even, O. Goldreich, and A. Lempel: *A Randomized Protocol for Signing Contracts*. Communications of the ACM, Volume 28, Issue 6, pg. 637-647, 1985.

[17] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, Srinivas Devadas: *Silicon physical random functions*. ACM Conference on Computer and Communications Security 2002: 148-160

[18] B. Gassend, D. Lim, D. Clarke, M. v. Dijk, S. Devadas: *Identification and authentication of integrated circuits.* Concurrency and Computation: Practice & Experience, pp. 1077 - 1098, Volume 16, Issue 11, September 2004.

[19] O. Goldreich: *Foundations of Cryptography: Volume II (Basic Applications).* Cambridge University Press, 2004.

[20] O. Goldreich and L.A. Levin: *A Hard-Core Predicate for all One-Way Functions.* STOC 1989: 25-32.

[21] Jorge Guajardo, Sandeep S. Kumar, Geert Jan Schrijen, Pim Tuyls: *FPGA Intrinsic PUFs and Their Use for IP Protection*. CHES 2007: 63-80

[22] Russell Impagliazzo, Steven Rudich: *Limits on the Provable Consequences of One-Way Permutations.* STOC 1989: 44-61

[23] Kilian, J.: *Founding cryptography on oblivious transfer*. STOC (1988)

[24] J.-W. Lee, D. Lim, B. Gassend, G. E. Suh, M. van Dijk, and S. Devadas. *A technique to build a secret key in integrated circuits with identification and authentication applications.* In Proceedings of the IEEE VLSI Circuits Symposium, June 2004.

[25] Daihyun Lim: *Extracting Secret Keys from Integrated Circuits*. MSc Thesis, MIT, 2004.

[26] Sandeep S. Kumar, Jorge Guajardo, Roel Maes, Geert Jan Schrijen, Pim Tuyls: *The Butterfly PUF: Protecting IP on every FPGA.* HOST 2008: 67-70

[27] M. Majzoobi, F. Koushanfar, M. Potkonjak: *Lightweight Secure PUFs.* IC-CAD 2008: 607-673.

[28] M. Majzoobi, F. Koushanfar, M. Potkonjak: *Testing techniques for hardware security.* In

[29] U. Maurer: *Conditionally-perfect secrecy and a provably-secure randomized cipher*. Journal of Cryptology, 5(1):53-66, 1992.

[30] Ueli M. Maurer: *Protocols for Secret Key Agreement by Public Discussion Based on Common Information*. CRYPTO 1992: 461-470

[31] R. Ostrovsky, A. Scaffuro, I. Visconti, A. Wadia: *Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions.* Cryptology ePrint Archive, March 16, 2012.

[32] R. Pappu: *Physical One-Way Functions*. PhD Thesis, Massachusetts Institute of Technology, 2001.

[33] R. Pappu, B. Recht, J. Taylor, N. Gershenfeld: *Physical One-Way Functions*, Science, vol. 297, pp. 2026-2030, 20 September 2002.

[34] Dusko Pavlovic: *Gaming security by obscurity.* CoRR abs/1109.5542: (2011)

[35] M.O. Rabin: *Digitalized signatures and public-key functions as intractable as factorization.* MIT/LCS/TR-212, 1979.

[36] Ron Rivest: *Illegitimi non carborundum*. Invited keynote talk, CRYPTO 2011.

[37] U. Rührmair: *SIMPL Systems, Or: Can we build cryptographic hardware without secret key information?*. SOFSEM 2011, Springer LNCS, 2011.

[38] U. Rührmair: *Oblivious Transfer based on Physical Unclonable Functions (Extended Abstract)*. TRUST Workshop on Secure Hardware, Berlin (Germany), June 22, 2010. Lecture Notes in Computer Science, Volume 6101, pp. 430 - 440. Springer, 2010.

[39] U. Rührmair, H. Busch, S. Katzenbeisser: *Strong PUFs: Models, Constructions and Security Proofs.* In A.-R. Sadeghi, P. Tuyls (Editors): Towards Hardware Intrinsic Security: Foundation and Practice. Springer, 2010.

[40] U. Rührmair, M. van Dijk: *Practical Security Analysis of PUF-based Two-Player Protocols.* In submission, 2012.

[41] U. Rührmair, C. Jaeger, M. Algasiner: *An Attack on PUF-based Session Key Exchange, and a Hardware-based Countermeasure: Erasable PUFs.* Financial Cryptography and Data Security 2011.

[42] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, J. Schmidhuber: *Modeling Attacks on Physical Unclonable Functions*. ACM Conference on Computer and Communications Security, 2010.

[43] U. Rührmair, J. Sölter, F. Sehnke: *On the Foundations of Physical Unclonable Functions.* Cryptology e-Print Archive, June 2009.

[44] George Savvides: *Interactive Hashing and reductions between Oblivious Transfer variants*. PhD Thesis, McGill University, 2007.

[45] G. E. Suh, S. Devadas: *Physical Unclonable Functions for Device Authentication and Secret Key Generation.* DAC 2007: 9-14

[46] Pim Tuyls, Boris Skoric, S. Stallinga, Anton H. M. Akkermans, W. Ophey: *Information-Theoretic Security Analysis of Physical Uncloneable Functions*. Financial Cryptography 2005: 141-155

[47] P. Tuyls, B. Skoric: *Strong Authentication with Physical Unclonable Functions.* In: Security, Privacy and Trust in Modern Data Management, M. Petkovic, W. Jonker (Eds.), Springer, 2007.

[48] Pim Tuyls, Geert Jan Schrijen, Boris Skoric, Jan van Geloven, Nynke Verhaegh, Rob Wolters *Read-Proof Hardware from Protective Coatings.* CHES 2006: 369-383

[49] A. D. Wyner: *The wire-tap channel.* Bell Systems Technical Journal, 54:1355-1387, 1975.

A The Semi-Honest Model Without Transfer of Physical Objects

We quote the standard definition of semi-honest behavior in the deterministic case (without the transfer of physical objects) from [19]:

Definition 26 (Privacy w.r.t. semi-honest behavior in the deterministic case [19])**.** *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a deterministic functionality, and denote the first (resp., second) output of $f(x,y)$ by $f_1(x,y)$ (resp., $f_2(x,y)$). Let Π be a two-party protocol for computing f. The* `view` *of the first (resp., second) party during execution of Π on (x,y), denoted $\texttt{view}_1^\Pi(x,y)$ (resp., $\texttt{view}_2^\Pi(x,y)$), is $(x, r, m_1, \ldots, m_t)$ (resp., $(y, r, m_1, \ldots, m_t)$), where r represents the outcome of the first (resp., second) party's internal coin tosses, and m_i represents the i-th message it has received.*

We say that Π privately computes f if there exist probabilistic polynomial time algorithms, denoted S_1 and S_2, such that[16]

$$\{S_1(x, f_1(x,y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{\texttt{view}_1^\Pi(x,y)\}_{x,y\in\{0,1\}^*} \quad (5)$$

$$\{S_2(x, f_2(x,y))\}_{x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{\texttt{view}_2^\Pi(x,y)\}_{x,y\in\{0,1\}^*} \quad (6)$$

where $|x| = |y|$.

An oracle-aided protocol Π using oracle-functionality O is said to privately compute f if there exist probabilistic polynomial time algorithms S_1 and S_2 satisfying (5-6), where the corresponding views of the execution of the

[16] Here, $\stackrel{c}{\equiv}$ denotes computational indistinguishability by non-uniform families of polynomial-size circuits such that indistinguishability also holds with respect to probabilistic polynomial-time machines that obtain (non-uniform) auxiliary inputs.

oracle-aided protocol are defined in the natural manner.[17] *An oracle-aided protocol* Π *is said to privately reduce* f *to* O*, if it privately computes* f *when using the oracle-functionality* O *(this is also called a black-box reduction).*

B A Quadratic Attack

In this appendix, we will describe a quadratic attack on the OT- and BC-protocols of Brzuska et al. [7]. It works fully in their own communication model, i.e., it does not assume new attack models such as the PAM or the bad PUF model. As discussed in detail in the upcoming Section B.3, it has the practical effect that the protocol is insecure when used with optical PUFs a la Pappu [33, 32] and with electrical PUFs that have medium challenge length of 64 bit, say. This is of particular relevance, since Brzuska et al. had explicitly suggested the use of optical PUFs in connection with their protocols (Section 8 of [8]). Our attack is regardless of the cryptographic hardness and unpredictability of the PUF, and only relates to the number of possible challenges a PUF possesses.

The attack has been described for the first time by Rührmair and van Dijk in [40]. Since this document is in submission and currently not publicly available, we excerpt from it in this appendix.

B.1 The OT- and BC-Protocol of Brzuska et al.

We start by describing the two protocols by Brzuska et al. in order to achieve a self contained treatment. To keep our exposition simple, we will not use the full UC-notation of [7], and will present the schemes mostly without error correction mechanisms, since the latter play no role in the context of our attack.

The protocols use two communication channels between the communication partners: A binary channel, over which all digital communication is handled. It is assumed that this channel is non-confidential, but authenticated. And secondly an insecure physical channel, over which the PUF is sent. It is assumed that adversaries can measure adaptively selected CRPs of the PUF while it is in transition over this channel.

B.1.1 Oblivious Transfer

The OT protocol of [7] implements one-out-of-two string oblivious transfer. It is assumed that in each subsession the sender P_i initially holds two (fresh) bitstrings $s_0, s_1 \in \{0,1\}^\lambda$, and that the receiver P_j holds a (fresh) choice bit b.

Brzuska et al. generally assume in their treatment that after error correction and the application of fuzzy extractors, a PUF can be modeled as a function $\mathsf{PUF} : \{0,1\}^\lambda \to \{0,1\}^{rg(\lambda)}$. In the subsequent protocol of Brzuska et al., it is furthermore assumed that $rg(\lambda) = \lambda$, i.e., that the PUF implements a function $\mathsf{PUF} : \{0,1\}^\lambda \to \{0,1\}^\lambda$ [7, 8]. Please note that we used this model throughout this paper, too.

Protocol 27: PUF-BASED OBLIVIOUS TRANSFER ([7], SLIGHTLY SIMPLIFIED DESCRIPTION)

External Parameters: The protocol has a number of external parameters, including the security parameter λ, the session identifier `sid`, a number N that specifies how many subsessions are allowed, and a pre-specified PUF-family $\mathcal{P}$, from which all PUFs which are used in the protocol must be drawn.

Initialization Phase: Execute once with fixed session identifier `sid`:

1. The receiver holds a PUF which has been drawn from the family $\mathcal{P}$.
2. The receiver measures l randomly chosen CRPs $c_1, r_1, \ldots, c_l, r_l$ from the PUF, and puts them in a list $\mathcal{L} := (c_1, r_1, \ldots, c_l, r_l)$.
3. The receiver sends the PUF to the sender.

[17] I.e., the view of the first (resp., second) party also records all queries by the first (resp., second) party to O. Computational indistinguishability is now defined with respect to machines that have oracle access to O. This allows such machines to extract from the first (resp., second) view information about x and y (more than what can be constructed from the input x (resp., y) and output $(x, f_1(x,y))$ (resp., $(y, f_2(x,y))$) alone) that needs oracle access to O in order to interpret the view. Also the simulator algorithms S_1 and S_2 have access to O in order to simulate the recorded oracle queries by the views. Summarizing, O is treated as a black-box accessible to all algorithms and protocols.

Subsession Phase: Repeat at most N times with fresh subsession identifier `ssid`:

1. The sender's input are two strings $s_0, s_1 \in \{0,1\}^\lambda$, and the receiver's input is a bit $b \in \{0,1\}$.
2. The receiver chooses a CRP (c, r) from the list $\mathcal{L}$ at random.
3. The sender chooses two random bitstrings $x_0, x_1 \in \{0,1\}^\lambda$ and sends x_0, x_1 to the receiver.
4. The receiver returns the value $v := c \oplus x_b$ to the sender.
5. The sender measures the responses r_0 and r_1 of the PUF that correspond to the challenges $c_0 := v \oplus x_0$ and $c_1 := v \oplus x_1$.
6. The sender sets the values $S_0 := s_0 \oplus r_0$ and $S_1 := s_1 \oplus r_1$, and sends S_0, S_1 to the receiver.
7. The receiver recovers the string s_b that depends on his choice bit b as $s_b = S_b \oplus r$. He erases the pair (c, r) from the list $\mathcal{L}$.

Comments. The protocol implicitly assumes that the sender and receiver can interrogate the PUF whenever they have access to it, i.e., that the PUF's challenge-response interface is publicly accessible and not protected. This implies that the employed PUF must possess a large number of CRPs. Using a PUF with just a few challenges does not make sense: The receiver could then create a full look-up table for all CRPs of such a PUF before sending it away in Step 3 of the Initialization Phase. This would subsequently allow him to recover both strings s_0 and s_1 in Step 6 of the protocol subsession, as he could obtain r_0 and r_1 from his look-up table. Similar observations hold for the upcoming protocol 28. Indeed, all protocols discussed in this paper require PUFs with a large number of challenges and publicly accessible challenge-response interfaces. These PUFs have sometimes been referred to as *Physical Random Functions* or also as *Strong PUFs* in the literature [21, 43, 42].

Furthermore, please note that the PUF is not transferred during the subsessions. According to the model of Brzuska et al., an adversary only has access to it during the initialization phase, but not between the subsessions. This protocol use has some similarities with a stand-alone usage of the PUF, in which exactly one PUF-transfer occurs between the parties.

B.1.2 Bit Commitment

The second protocol of [7] implements PUF-based Bit Commitment (BC) by a generic reduction to PUF-based OT. The BC-sender initially holds a bit b. When the OT-Protocol is called as a subprotocol, the roles of the sender and receiver are reversed: The BC-sender acts as the OT-receiver, and the BC-receiver as the OT-sender. The details are as follows.

Protocol 28: PUF-BASED BIT COMMITMENT VIA PUF-BASED OBLIVIOUS TRANSFER ([7], SLIGHTLY SIMPLIFIED DESCRIPTION)

Commit Phase:

1. The BC-sender and the BC-receiver jointly run an OT-protocol (for example Protocol 27).
   (a) In this OT-protocol, the BC-sender acts as OT-receiver and uses his bit b as the choice bit of the OT-protocol.
   (b) The BC-receiver acts as OT-sender. He chooses two strings $s_0, s_1 \in \{0,1\}^\lambda$ at random, and uses them as his input s_0, s_1 to the OT-protocol.
2. When the OT-protocol is completed, The BC-sender has learned the string $v := s_b$. This closes the commit phase.

Reveal Phase:

1. In order to reveal bit b, the BC-sender sends the string (b, v) (with $v = s_b$) to the BC-receiver.

Comments. The security of the BC-protocol is inherited from the underlying OT-protocol. Once this protocol is broken, also the security of the BC-protocol is lost. This will be relevant in the upcoming sections.

B.2 A Quadratic Attack on Protocols 27 and 28

We will now discuss a cheating strategy in Protocols 27 and 28. Compared to an attacker who exhaustively queries the PUF for all of its m possible challenges, we describe an attack on Protocols 27 and 28 which reduces this number to $\sqrt{m}$. As we will argue later in Section B.3, this has a particularly strong effect on the protocol's security if an optical PUF is used (as has been explicitly suggested by [8]), or if electrical PUFs with medium challenge lengths of 64 bits are used.

Our attack rests on the following lemma.

Lemma 29. *Consider the vector space $(\{0,1\}^\lambda, \oplus)$, $\lambda \geq 2$, with basis $\mathcal{B} = \{a_1, \ldots, a_{\lfloor \lambda/2 \rfloor}, b_1, \ldots, b_{\lceil \lambda/2 \rceil}\}$. Let A be equal to the linear subspace generated by the vectors in $\mathcal{B}_A = \{a_1, \ldots, a_{\lfloor \lambda/2 \rfloor}\}$ and let B the linear subspace generated by the vectors in $\mathcal{B}_B = \{b_1, \ldots, b_{\lceil \lambda/2 \rceil}\}$. Define $S := A \cup B$. Then it holds that:*

(i) *Any vector $z \in \{0,1\}^\lambda$ can be expressed as $z = a \oplus b$ with $a, b \in S$, and this expression (i.e., the vectors a and b) can be found efficiently (i.e., in at most $poly(\lambda)$ steps).*

(ii) *For all distinct vectors $x_0, x_1, v \in \{0,1\}^\lambda$ there is an equal number of combinations of linear subspaces A and B as defined above for which $x_0 \oplus v \in A$ and $x_1 \oplus v \in B$.*

(iii) *S has cardinality $|S| \leq 2 \cdot 2^{\lceil \lambda/2 \rceil}$.*

Proof. (i) Notice that any vector $z \in \{0,1\}^\lambda$ can be expressed as a linear combination of all basis vectors: $z = \sum u_i a_i + \sum v_j b_j$, i.e., $z = a \oplus b$ with $a \in A$ and $b \in B$. This expression is found efficiently by using Gaussian elimination.

(ii) Without loss of generality, since x_0, x_1 and v are distinct vectors, we may choose $a_1 = x_0 \oplus v \neq 0$ and $b_1 = x_1 \oplus v \neq 0$. The number of combinations of linear subspaces A and B is independent of the choice a_1 and b_1. (Notice that if $x_0 \neq x_1$ but $v = x_0$, then the number of combinations is twice as large.)

(iii) The bound follows from the construction of S and the cardinalities of A and B, which are $|A| = 2^{\lfloor \lambda/2 \rfloor}$ and $|B| = 2^{\lceil \lambda/2 \rceil}$. □

An Example. Let us give an example in order to illustrate the principle of Lemma 29. Consider the vector space $(\{0,1\}^\lambda, \oplus)$ for an even λ, and choose as subbases $\mathcal{B}_{A_0} = \{e_1, \ldots, e_{\lambda/2}\}$ and $\mathcal{B}_{B_0} = \{e_{\lambda/2+1}, \ldots, e_\lambda\}$, where e_i is the unit vector of length λ that has a one in position i and zeros in all other positions. Then the basis $\mathcal{B}_{A_0}$ spans the subspace A_0 that contains all vectors of length λ whose second half is all zero, and $\mathcal{B}_{B_0}$ spans the subspace B_0 that comprises all vectors of length λ whose first half is all zero. It then follows immediately that every vector $z \in \{0,1\}^\lambda$ can be expressed as $z = a \oplus b$ with $a \in A_0$ and $b \in B_0$, or, saying this differently, with $a, b \in S$ and $S := A_0 \cup B_0$. It also immediate that S has cardinality $|S| \leq 2 \cdot 2^{\lambda/2}$.

Relevance for PUFs. The lemma translates into a PUF context as follows. Suppose that a malicious and an honest player play the following game. The malicious player gets access to a PUF with challenge length λ in an initialization period, in which he can query CRPs of his choice from the PUF. After that, the PUF is taken away from him. Then, the honest player chooses a vector $z \in \{0,1\}^\lambda$ and sends it to the malicious player. The malicious player wins the game if he can present the correct PUF-responses r_0 and r_1 to two arbitrary challenges c_0 and c_1 which have the property that $c_0 \oplus c_1 = z$. Our lemma shows that in order to win the game with certainty, the malicious player does not need to read out the entire CRP space of the PUF in the initialization phase; he merely needs to know the responses to all challenges in the set S of Lemma 29, which has a quadratically reduced size compared to the entire CRP space. This observation is at the heart of the attack described below.

In order to make the attack hard to detect for the honest player, it is necessary that the attacker chooses random subspaces A and B, and does not use the above trivial choices A_0 and B_0 all the time. This fact motivates the random choice of A and B in Lemma 29. The further details are as follows.

The Attack. As in [7, 8], we assume that the PUF has got a challenge set of $\{0,1\}^\lambda$. Given Lemma 29, the OT-receiver (who initially holds the PUF) can achieve a quadratic advantage in Protocol 27 as described below.

First, he chooses uniformly random linear subspaces A and B, and constructs the set S, as described in Lemma 29. While he holds possession of the PUF before the start of the protocol, he reads out the responses to all challenges in S. Since $|S| \leq 2 \cdot 2^{\lceil \lambda/2 \rceil}$, this is a quadratic improvement over reading out all responses of the PUF.

Next, he starts the protocol as normal. When he receives the two values x_0 and x_1 in Step 3 of the protocol, he computes two challenges c_0^* and c_1^* both in set S such that

$$x_1 \oplus x_2 = c_0^* \oplus c_1^*.$$

According to Lemma 29(i), this can be done efficiently (i.e., in $poly(\lambda)$ operations). Notice that, since the receiver knows all the responses corresponding to challenges in S, he in particular knows the two responses r_0^* and r_1^* that correspond to the challenges c_0^* and c_1^*.

Next, the receiver deviates from the protocol and sends the value $v := c_0^* \oplus x_0$ in Step 4. For this choice of v, the two challenges c_0 and c_1 that the sender uses in Step 5 satisfy

$$c_0 := c_0^* \oplus x_0 \oplus x_0 = c_0^*$$

and

$$c_1 := c_0^* \oplus x_0 \oplus x_1 = c_0^* \oplus c_0^* \oplus c_1 = c_1^*.$$

By Lemma 29(ii), Alice cannot distinguish the received value v in Step 4 from any random vector v. In other words, Alice cannot distinguish Bob's malicious behavior (i.e., fabricating a special v with suitable properties) from honest behavior. As a consequence, Alice continues with Step 6 and transmits $S_0 = s_0 \oplus r_0^*$ and $S_1 = s_1 \oplus r_1^*$. Since Bob knows both r_0^* and r_1^*, he can recover both s_0 and s_1. This breaks the security of the protocol.

Please note the presented attack is simple and effective: It fully works within the original communication model of Brzuska et al. [7, 8]. Furthermore, it does not require laborious computations of many days on the side of the attacker (as certain modeling attacks on PUFs do [42]). Finally, due to the special construction we proposed, the honest players will not notice the special choice of the value v, as the latter shows no difference from a randomly chosen value.

Effect on Bit Commitment (Protocol 28). Due to the reductionist construction of Protocol 28, our attack on the oblivious transfer scheme of Protocol 27 directly carries over to the bit commitment scheme of Protocol 28 if Protocol 27 is used in it as a subprotocol. By using the attack, a malicious sender can open the commitment in both ways by reading out only $2 \cdot 2^{\lceil \lambda/2 \rceil}$ responses (instead of all 2^λ responses) of the PUF. On the other hand it can be observed easily that the hiding property of the BC-Protocol 28 is unconditional, and is not affected by our attack.

B.3 Practical Consequences of the Attack

What are the practical consequences of our quadratic attack, and how relevant is it in real-world applications? The situation can perhaps be illustrated via a comparison to classical cryptography. What effect would a quadratic attack have on schemes like RSA, DES and SHA-1? To start with RSA, the effect of a quadratic attack here is rather mild: The length of the modulus must be doubled. This will lead to longer computation times, but restore security without further ado. In the case of single-round DES, however, a quadratic attack would destroy its security, and the same holds for SHA-1. The actual effect of our attack on PUF-based OT and BC has some similarities with DES or SHA-1: PUFs are finite objects, which cannot be scaled in size indefinitely due to area requirements, arising costs, and stability problems. This will also become apparent in our subsequent discussion.

B.3.1 Electrical Integrated PUFs

We start our dicussion by electrical integrated PUFs, and take the well-known Arbiter PUF as an example. It has been discussed in theory and realized in silicon mainly for challenge lengths of 64 bits up to this date [17, 18, 24, 45]. Our attack on such a 64-bit implementation requires the read-out of $2 \cdot 2^{32} = 8.58 \cdot 10^9$ CRPs by the receiver. This read-out can be executed before the protocol, not during the protocol, and will hence not be noticed by the sender. Assuming a MHz CRP read-out rate [24] of the Arbiter PUF, the read-out takes $8.58 \cdot 10^3\,\mathrm{sec}$, or less than $144\,\mathrm{min}$.

Please note that the attack is independent of the cryptographic hardness of the PUF, such as its resilience against machine learning attacks. For example, a 64-bit, 8-XOR-Arbiter PUF (i.e., an Arbiter PUF with eight parallel standard 64-bit Arbiter PUFs whose single responses are XORed at the end of the structure) is considered secure in practice against all currently known machine learning techniques [42]. Nevertheless, this type of PUF would still allow the above attack in $72\,\mathrm{min}$.

Our attacks therefore enforce the use of PUFs with a challenge bitlength of 128 bits or more in Protocols 27 and 28. Since much research currently focuses on 64-bit implementations of electrical PUFs, publication and dissemination of the attack seems important to avoid their use in Protocols 27 and 28. Another aspect of our attack is that it motivates the search for OT- and BC-protocols that are immune, and which can safely be used with 64-bit implementations. The reason is that the usage of 128-bit PUFs doubles the area consumption of the PUF and negatively affects costs.

B.3.2 Optical PUFs

Let us now discuss the practical effect of our attack on the the optical PUF introduced by Pappu [32] and Pappu et al. [33]. The authors use a cuboid-shaped plastic token of size 1 cm $\times$ 1 cm $\times$ 2.5 mm, in which thousands of light scattering small spheres are distributed randomly. They analyze the number of applicable, decorrelated challenge-response pairs in their set-up, arriving at a figure of $2.37 \cdot 10^{10}$ [33]. Brzuska et al. assume that these challenges are encoded in a set of the form $\{0,1\}^\lambda$, in which case $\lambda = \lceil \log_2 2.37 \cdot 10^{10} \rceil = 35$. If this number of 2^{35} is reduced quadratically by virtue of Lemma 29, we obtain on the order of $2 \cdot 2^{18} = 5.2 \cdot 10^5$ CRPs that must be read out by an adversary in order to cheat. It is clear that even dedicated measurement set-ups for optical PUFs cannot realize the MHz rates of the electrical example in the last section. But even assuming mild read-out rates of 10 CRPs or 100 CRPs per second, we still arrive at small read-out times of $5.2 \cdot 10^4\,\mathrm{sec}$ or $5.2 \cdot 10^3\,\mathrm{sec}$, respectively. This is between 14.4 hours (for 10 CRPs per second) or 87 minutes (for 100 CRPs per second). If a malicious receiver holds the PUF for such a time frame before the protocol starts (which is impossible to control or prevent for the honest players), he can break the protocol's security.

Can the situation be cleared by simply scaling the optical PUF to larger sizes? Unfortunately, also an asymptotic analysis of the situation shows the same picture. All variable parameters of the optical PUF [33, 32] are the x-y-coordinate of the incident laser beam and the spatial angle Θ under which the laser hits the token. This leads to a merely cubic complexity in the three-dimensional diameter d of the cuboid scattering token. [18] Given our attack, this implies that the adversary must only read out $O(d^{\,1.5})$ challenges in order to cheat in Protocols 27 and 28. If only the independent challenges are considered, the picture is yet more drastic: As shown in [46], the PUF has at most a quadratic number of *independent* challenges in d. This reduces to a merely *linear* number of CRPs which the adversary must read out in our attack. Finally, we remark that scaling up the size of the PUF also quickly reaches its limits under practical aspects: The token considered by Pappu et al. [33, 32] has an area of 1 cm $\times$ 1 cm. In order to slow down the quadratic attack merely by a factor of 10, a token of area 10 cm $\times$ 10 cm would have to be used. Such a token is too large to even fit onto a smart card.

Overall, this leads to the conclusion that optical PUFs like the ones discussed in [32, 33] cannot be used safely with the Protocols 27 and 28 in the face of our attack. Due to their low-degree polynomial CRP complexity, and due to practical size constraints, simple scaling of the PUFs constitutes no efficient countermeasure. This distinguishes the optical approach from the electrical case of the last section. This observation has a particular relevance, since Brzuska et al. had explicitly suggested optical PUFs for the implementation of their protocols (see Section 8 of [8]).

B.4 Are There Counter Measures?

Let us quickly consider potential countermeasures against our attacks and their practical feasibility in this section. One first idea is: Can we bind the time in which the malicious player has got access to the PUF? The current Protocols 27 and 28 obviously are unsuited to this end; but could there be modifications of theirs which have this property? A simple approach would be to introduce one additional PUF transfer from the sender to the receiver in the initialization phase. This assumes that the sender initially holds the PUF, transfers it to the receiver, and

[18] Please note in this context that the claim of [8] that the number of CRPs of an optical PUF is super-polynomial must have been made erroneously or by mistake; our above brief analysis shows that it is at mostly cubic. The low-degree polynomial amount of challenges of the optical PUF is indeed confirmed by the entire literature on the topic, most prominently [33, 32, 46].

measures the time frame by which the receiver returns the PUF. The period in which the receiver had access to the PUF gives a bound on the number of CRPs he knows. This can be used in the protocols to guarantee security. Please note that a long and uncontrolled access time for the sender is no problem for the protocol's security, so we only need to bind the access time of the receiver by the above approach.

On closer inspection, however, there are some problems with this technique. The first and foremost problem is its mediocre practicality. In general, each PUF-transfer in a protocol is very costly. If executed via physical, postal delivery over long distances between arbitrary parties, it might cost days. Having two such transfers in one protocol is devastating for the protocol's practicality.

A second issue is that binding the adversarial access time in a tight manner by the suggested procedure is very difficult. How long will the physical transfer take? 1 day? What if the adversary or someone else can do it faster, and the adversary uses the gained time for executing measurements on the PUF? What if the adversary executes the physical transfer himself, and can measure the PUF while it is in transit? Obtaining a tight and short bound on the adversary's access time seems impossible here.

In summary, there are only very, very few circumstances where enforcing a time bound on the receiver's access time is possible in a realistic setting. The above idea is hence interesting for future PUF-protocol design, but cannot be considered a general countermeasure.

C Behavior of Known PUF-Protocols in the PAM

We will now illustrate how several known two-party PUF-protocols behave in the posterior access model (PAM), i.e., under the assumption that the adversary gains access to the PUF after a subsession of the protocols. For space reasons, we will carry out only one exemplary analysis, namely for the OT- and BC-protocol of Brzuska et al. The other cases (Brzuska's KE protocol and Rührmair's OT protocol) are somewhat similar. The well-known CRP-based PUF-identification protocol [32, 33] is not affected in the PAM. In order to achieve a self-contained treatment, the OT- and BC-protocol of Brzuska et al. have been described as Protocols 27 and 28 in Section B.1.1.

C.1 Brzuska et al.'s OT and BC Protocol in the PAM

Let us start by describing an attack on the OT-protocol of Brzuska et al. (which was described as Protocol 27 in Section B.1) in the PAM. In terms of notation, we relate to the terminology of Protocol 27 in our attack. The attack rests on the following *assumptions*:

1. The initialization phase of the OT-Protocol 27 is carried out between the sender and the receiver.
2. Later, different subsessions of the protocol are run. We assume that there is a subsession `ssid` with the following properties:
   - Eve was able to eavesdrop the binary communication between the sender and the receiver in the subsession `ssid`.
   - Eve can read-out CRPs from the PUF after the end of the subsession `ssid`, for example before a new subsession `ssid`$'$ is started.

Under these provisions, Eve can learn both bits s_0 and s_1 used by the sender in subsession `ssid`. This breaks the security of this subsession. The attack works as follows:

1. When the subsession `ssid` is run, Eve eavesdrops the messages in Steps 3, 4 and 6. She therefore learns the values $x_0, x_1, v\ (:= c \oplus x_b), S_0\ (:= s_0 \oplus r_0)$ and $S_1\ (:= s_1 \oplus r_1)$, hence, r_0 and r_1 are the responses to the challenges $c_0 (:= v \oplus x_0)$ and $c_1 (:= v \oplus x_1)$.
2. When Eve has got physical access to the PUF after the subsession `ssid`, she computes the challenges $c_0 := v \oplus x_0$ and $c_1 := v \oplus x_1$ herself. She applies these challenges to the PUF, and obtains the responses r_0 and r_1.
3. Eve derives s_0 and s_1 by computing the values $S_0 \oplus r_0 = s_0 \oplus r_0 \oplus r_0 = s_0$ and $S_1 \oplus r_1 = s_1 \oplus r_1 \oplus r_1 = s_1$. This breaks the security of the subsession `ssid`.

Please note that an attacker cannot learn the receiver's choice bit b by a similar attack, since the secrecy of the choice bit is unconditional and does not rely on the employed PUF.

Consequences for Bit Commitment. Since Protocol 28 is a direct reduction of BC to OT, the above attack directly affects Protocol 28 whenever it employs Protocol 27 as a subprotocol. A BC-sender who acts as adversary in Protocol 28 learns both s_0 and s_1 by applying our above attack, under the provision that he gets access to the PUF between the commit phase and the reveal phase, or simply after the commit phase in the case that the reveal phase is never executed. Knowledge of s_0 and s_1 enable him to open his commitment in both ways: to open the value "0", he sends $(0, s_0)$ in the reveal phase, and to open the value "1", he sends $(1, s_1)$. Note that the hiding property of the BC-Protocol 28 is not affected by our attack, since it is unconditional and independent of the PUF. Furthermore, we comment that the attack does not work if the adversary or malicious players get access to the PUF only after the end of the reveal phase. In this case, the BC-protocol remains secure.

Relevance for the OT-Protocol of Rührmair [38]. Please note that the above attack strategy directly carries over to the OT-protocol of Rührmair [38], which consequently is also not secure in the PAM. The details are straightforward and left to the reader for space reasons. For reasons of fairness, we would also like to remark that the protocol of Rührmair was merely suggested in a stand-alone setting from the start. Nevertheless, our attack indicates that the picture drastically changes in the PAM.

C.2 Conclusions

The security of many currently known PUF-protocols, including the protocols of Brzuska et al., is not maintained in PAM. We **stress** that these protocols were not designed for the PAM, and that the corresponding security proofs do not assume posterior access. On the other hand, the PAM constitutes a practically very viable attack scenario in most PUF applications, as argued in detail in Section 2. If the protocols of Brzuska et al. or of Rührmair were used in these applications, they would likely be faced with our attacks. This makes the behavior of the protocols in the PAM a relevant issue.

D Behavior of Known PUF-Protocols in the Bad PUF Model

We now briefly investigate the behavior of known PUF-protocols in the bad PUF model. Exemplarily, we will examine the security of the OT- and BC-protocol of Brzuska et al. [7] under the assumption that the players may generate and use simulatable PUFs. In order to achieve a self-contained treatment, the OT- and BC-protocol of Brzuska et al. have been described as Protocols 27 and 28 in Section B.1.1.

D.1 Brzuska et al.'s OT- and BC-Protocol and Simulatable PUFs

We start by the attack on the OT-protocol of Brzuska et al. The attack makes the following single assumption:

1. The receiver hands over a simulatable bad PUF instead of a proper PUF in the initialization phase, and furthermore possesses a simulation algorithm for this PUF.

The attack itself works as follows:

1. The receiver follows Protocol 27 as specified, and carries out a subsession `sid`.
2. When the subsession is completed, the receiver computes the two challenges $c_0 := v \oplus x_0$ and $c_1 := v \oplus x_1$. He can do so since he knows v, x_0 and x_1 from earlier protocol steps.
3. The receiver uses his simulation algorithm in order to compute the two responses r_0 and r_1 which correspond to the challenges c_0 and c_1.
4. The receiver derives both values s_0 and s_1 by computing $S_0 \oplus r_0 = s_0 \oplus r_0 \oplus r_0 = s_0$ and $S_1 \oplus r_1 = s_1 \oplus r_1 \oplus r_1 = s_1$. He can do so since he knows S_0, S_1 from step 6 of the OT-protocol. This breaks the security of the protocol.

Consequences for the Bit Commitment Protocol of [7]. The BC-protocol of Brzuska et al. is a direct reduction of BC to OT. The above attack on their OT-protocol hence transfers to their BC-protocol if the OT-protocol is used as a subprotocol therein. The BC-sender will be able to open his commitment in two ways. The hiding property of the BC-protocol is not affected; it is unconditional and independent of the used PUF.

Relevance for the OT-Protocol of Rührmair [38]. Please note that the above attack strategy directly carries over to the OT-protocol of Rührmair [38], which consequently is also not secure in the bad PUF model. The details are straightforward and left to the reader for space reasons.

D.2 Conclusions

The security of the OT- and BC-protocol of Brzuska et al. is not maintained under the use of simulatable PUFs. We **stress** that these protocols are not designed for such use, and that the corresponding security proofs do not assume simulatable PUFs. On the other hand, simulatable PUFs do constitute a practically viable attack scenario in many PUF applications: Physical authentication of PUFs is difficult and very laborious in practical settings, and bad PUFs introduced by the parties in two-party protocols are particularly difficult to detect, as we argued in all detail in Section 2. If the protocols of Brzuska et al. were used in practice, they would likely be faced with this class of attacks.

Please note that the previous discussion does not consider the effect of challenge-logging PUFs. The corresponding attacks are somewhat similar, but require the assumption that the PUFs are re-used in protocols with other players. The details are left to the readers for space reasons.

E Two Further Implementations of Simulatable Bad PUFs

In Section 2 we argued that a straightforward implementation of simulatable PUFs is the use of a pseudorandom number generator with a seed s that is known to the malicious party. We present two other viable constructions in this section to interested readers. They use existing PUF-designs and modify them in such a way that the PUF-manufacturer can machine learn and hence simulate them. The extra wires which allow this machine learning option are disabled before the PUF is released to the field, meaning that other parties will not be able to use them. The details are described in the next sections.

E.1 Simulatable PUFs by XOR Arbiter PUFs with Extra Wires

Currently, the most compact and secure electrical Strong PUF implementation probably are XOR Arbiter PUFs. In these constructions, several of the well-known Arbiter PUFs [17, 18, 24, 45] are used in parallel, and their outputs are XORed in order to obtain the output. Such XOR-constructions have explicitly been described and examined in [25, 45, 27].

Arbiter PUFs were tested for their security against machine learning-based modeling attacks in a number of publications, including [25, 28, 42]. In these attacks, an adversary collects a large set of CRPs of the PUF, and feeds them into a machine learning algorithm. The algorithm tries to derive a numeric simulation model of the PUF. If successful, the PUF-responses can afterwards be predicted numerically by this algorithm. The results of [25, 42] were as follows: (i) A single Arbiter PUF can be learned and predicted very efficiently (i.e., with very few CRPs and with very small computation times). (ii) The XOR of several single Arbiter PUF is increasingly hard to learn. Constructions which employ more than six single Arbiter PUFs and XOR their single outputs in order to create a single bit as the overall output cannot be machine learned efficiently with current methods.

PUF designers whose aim is the construction of secure PUFs thus will make sure that the adversary cannot access the outputs of the single Arbiter PUFs before they are XORed. Otherwise, an adversary could collect the CRPs from the single Arbiter PUFs, and so machine learn the behavior of each single Arbiter PUF. And once she can predict each single Arbiter PUF, she can also predict the XOR of all single Arbiter PUFs.

This situation on the forefront of electrical Strong PUF implementations leads to a very simple strategy for the fabrication of simulatable PUFs. The malicious party uses an XOR Arbiter PUF with, for example, eight single Arbiter PUFs, but with a little twist: There are extra wires which transfer the responses of the single Arbiter PUFs to the outside before they are fed into the XOR-operation. In an initial phase after fabrication, the malicious party

uses these extra wires in order to collect CRPs from each single Arbiter PUF, allowing her to machine learn the behavior of each single Arbiter PUF. This enables the malicious party to create a simulation algorithm for the responses of the entire XOR Arbiter PUF. Once this algorithm has been created, the extra wires are permanently disabled. The situation is depicted schematically in Figure 1.

One advantage of this XOR-based implementation of simulatable PUFs is the following: The resulting PUF has *exactly* the same output as a "normal" XOR Arbiter PUF, since the extra wires have no effect on this output. The simulatable XOR Arbiter PUF hence cannot be distinguished from a normal XOR Arbiter PUF by the honest party via mere CRP measurement, while the malicious party holds a simulation algorithm for it.

E.2 Optical Simulatable Bad PUFs

An interesting question is whether there are optical simulatable PUFs. For example: Is there a simulatable version of Pappu's optical PUF [33]? Currently, it seems impossible to answer this question with definiteness. Speculating about future developments, it may well be possible to arrange the scattering centers of Pappu's PUF in a way that simplifies the output. For example, using very few scattering centers may lead to a simplified or even trivial input-output behavior. Current protocols do not test or exclude this possibility: In Protocols 27, 28 and 9, the malicious party could even use a plastic token that does not contain any scattering centers at all (and which produces trivial outputs) without being detected, and could use this to cheat. Generally, the question whether optical PUFs a la Pappu [33] can be made simulatable seems undecided.

On the other hand, integrated optical PUFs are known to be attackable by machine learning methods in certain settings. This has been shown first in [37], which described attacks on a real optical system in the appendix of [37]. Integrated optical PUFs could hence in principle be used by malicious parties as simulatable PUFs.

Let us elaborate on this in more detail. Figure 2 shows a schematic example of an integrated optical PUF. The input and output of challenges and responses is in digital form. [37] proved that on the basis of a large number of plain scattering images (i.e., those images that have not yet been processed by some image transformation), the input-output behavior can be machine learned and simulated with very high accuracy. This can be exploited by a malicious party can to build an optical simulatable PUF in a straightforward fashion: She builds on option to read out the CCD images directly into the PUF. Once the PUF has been machine learned and the simulation algorithm has been constructed, this option is permanently disabled, similar to the electrical construction based on XOR Arbiter PUFs. The adversary then possesses a simulation model of the PUF and can use this to break protocols in which this PUF is used. For further details on the machine learning of integrated optical PUFs we refer the reader to [37].

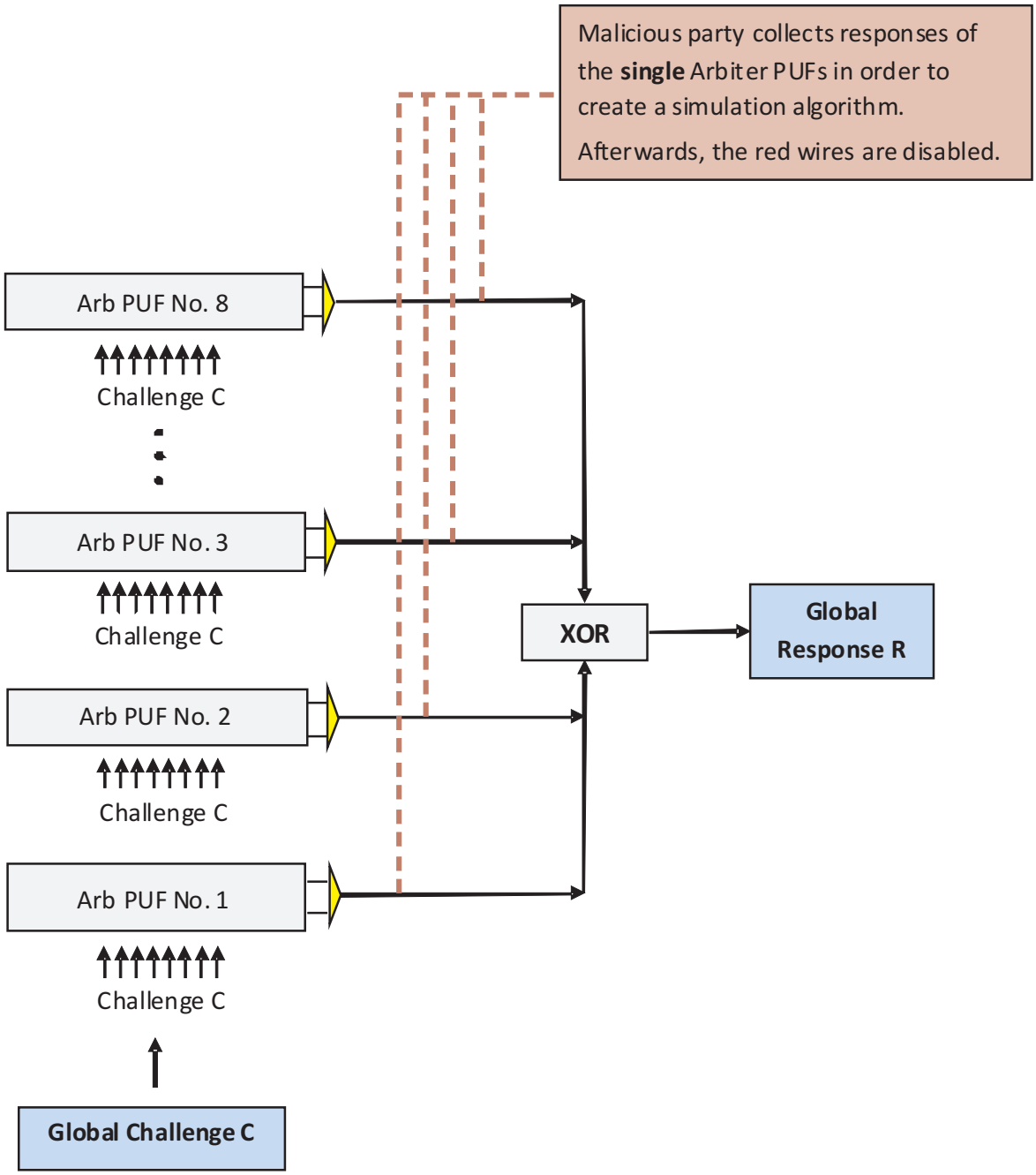

Figure 1: A schematic illustration of an XOR Arbiter PUFs used to create a simulatable PUF.

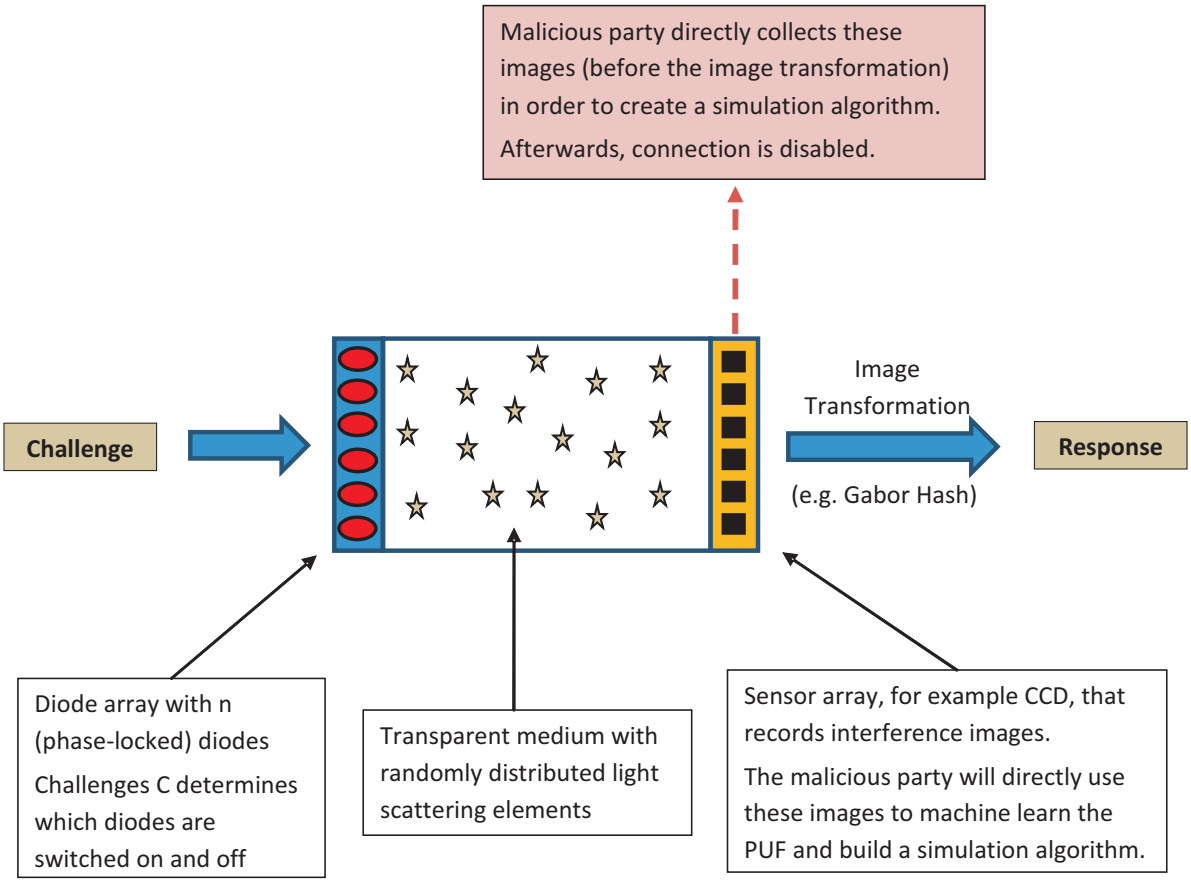

Figure 2: A schematic illustration of an integrated optical PUF that is used to create a simulatable PUF.