PUFs in Security Protocols: Attack Models and Security Evaluations

Ulrich Rührmair
Computer Science Department
Technische Universität München
80333 München, Germany
ruehrmair@in.tum.de

Marten van Dijk
CSAIL
MIT
Cambridge, Massachusetts
marten@mit.edu

***Abstract*—In recent years, PUF-based schemes have not only been suggested for the basic security tasks of tamper sensitive key storage or system identification, but also for more complex cryptographic protocols like oblivious transfer (OT), bit commitment (BC), or key exchange (KE). In these works, so-called "Strong PUFs" are regarded as a new, fundamental cryptographic primitive of their own, comparable to the bounded storage model, quantum cryptography, or noise-based cryptography. This paper continues this line of research, investigating the correct adversarial attack model and the actual security of such protocols.**

In its first part, we define and compare different attack models. They reach from a clean, first setting termed the *"stand-alone, good PUF model"* to stronger scenarios like the *"bad PUF model"* and the *"PUF re-use model"*. We argue why these attack models are realistic, and that existing protocols would be faced with them if used in practice. In the second part, we execute exemplary security analyses of existing schemes in the new attack models. The evaluated protocols include recent schemes from Brzuska et al. published at Crypto 2011 [1] and from Ostrovsky et al. [18]. While a number of protocols are certainly secure in their own, original attack models, the security of *none* of the considered protocols for OT, BC, or KE is maintained in all of the new, realistic scenarios.

One consequence of our work is that the design of advanced cryptographic PUF protocols needs to be strongly reconsidered. Furthermore, it suggests that Strong PUFs require additional hardware properties in order to be broadly usable in such protocols: Firstly, they should ideally be *"erasable"*, meaning that single PUF-responses can be erased without affecting other responses. If the area efficient implementation of this feature turns out to be difficult, new forms of Controlled PUFs [8] (such as Logically Erasable and Logically Reconfigurable PUFs [13]) may suffice in certain applications. Secondly, PUFs should be *"certifiable"*, meaning that one can verify that the PUF has been produced faithfully and has not been manipulated in any way afterwards. The combined implementation of these features represents a pressing and challenging problem, which we pose to the PUF hardware community in this work.

***Keywords*-(Strong) Physical Unclonable Functions; (Strong) PUFs; Attack Models; Oblivious Transfer; Bit Commitment; Key Exchange; Erasable PUFs; Certifiable PUFs**

I. Introduction

Today's electronic devices are mobile, cross-linked and pervasive, which makes them a well-accessible target for adversaries. The well-known protective cryptographic techniques all rest on the concept of a secret binary key: They presuppose that devices store a piece of digital information that is, and remains, unknown to an adversary. It turns out that this requirement is difficult to realize in practice. Physical attacks such as invasive, semi-invasive or side-channel attacks carried out by adversaries with one-time access to the devices, as well as software attacks like application programming interface (API) attacks, viruses or Trojan horses, can lead to key exposure and security breaks. As Ron Rivest emphasized in his keynote talk at CRYPTO 2011 [21], 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 main motivation for the development of *Physical Unclonable Functions (PUFs)* was their promise to better protect secret keys. A PUF is an (at least partly) disordered physical system P that can be challenged with so-called external stimuli or challenges c, upon which it reacts with corresponding responses r. Contrary to standard digital systems, these responses depend on the micro- or nanoscale structural disorder of 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. Any PUF P thus implements a unique and individual function f_P that maps challenges c to responses $r = f_P(c)$. Thereby 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 the system in an easily accessible digital form. Finally, certain types of PUFs are naturally tamper sensitive: Their exact behavior depends on minuscule manufacturing irregularities, often in different layers of the IC. Removing or penetrating these layers will automatically change the PUF's read-out values. These facts have been exploited in the past for different PUF-based security protocols. Prominent examples include identification [20], [9], key exchange [20], and various forms of (tamper sensitive) key storage and applications thereof, such as intellectual property protection or read-proof

1081-6011/13 $26.00 © 2013 IEEE
DOI 10.1109/SP.2013.27

memory [11], [15], [31].

In recent years, however, also the use of PUFs in more advanced cryptographic protocols together with formal security proofs has been investigated. In these protocols, PUFs with a very large challenge set and a freely accessible challenge-response interface are employed. This type of PUF sometimes has been referred to as *Physical Random Function* [9] or *Strong PUF* [11], [29], [28], [23] in the literature (see also Appendix A). [1] The (Strong) PUF is used similar to a "physical random oracle" in these protocols, which is passed on between the parties, and which can be read-out exactly by the very party who currently holds physical possession of it. Its input-output behavior is assumed to be so complex that its response to a randomly chosen challenge cannot be predicted numerically and without direct physical measurement, not even by a person who had physical access to the Strong PUF at earlier points in time.

In 2010, Rührmair [22] showed that oblivious transfer can be realized between two parties by physically transferring a Strong PUF in this setting. He observed that via the classical reductions of Kilian [14], this implies PUF-based bit commitment and PUF-based secure multi-party computations. In the same year, the first formal security proof for a Strong PUF protocol was provided by Rührmair, Busch and Katzenbeisser [23]. They present definitions and a reductionist security proof for Strong PUF based identification. In 2011, Rührmair, Jaeger and Algasinger [26] discuss an attack on a PUF-based session key exchange scheme of Tuyls and Skoric [32], in which the scheme is broken under the provision that it is executed several times and that the adversary gains access to the PUF more than once. Their attack motivated our PUF re-use model. At CRYPTO 2011 Brzuska, Fischlin, Schröder and Katzenbeisser [1] adapted Canetti's universal composition (UC) framework [3] to include PUFs, giving PUF-protocols for oblivious transfer (OT), bit commitment (BC), and key exchange (KE). At CHES 2012, Rührmair and van Dijk [25] presented a quadratic attack on Brzuska et al.'s OT- and BC-protocols, showing that their security is not maintained if optical PUFs or electrical PUFs with challenge length of 64 bits are used in their implementation. Two very recent eprint papers continue this general line of work: Ostrovsky, Scafuro, Visconti and Wadia [18] [2] investigate the use of so-called "malicious PUFs", and furthermore extend Brzuska et al.'s communication model in the UC framework.In independent and simultaneous work, van Dijk and Rührmair proposed a model equivalent to "malicious PUFs" under the name "bad PUF model", and a new attack model termed "PUF re-use model". The authors devise the first impossibility results for PUF-protocols in these two models [6].

While the body of work on (Strong) PUFs in cryptographic protocols is obviously growing, most papers use different implicit attack models, making it difficult to compare their results. There are situations where practically relevant attacks exist on protocols that are provably secure in other, perhaps mainly theoretical models. This motivates a comparative, systematic study of attack models.

Scope of this Work: This paper continues the above line of research. It investigates the UC-models of Brzuska et al. [1] and Ostrovsky et al. [18], and introduces several other, practically relevant attack scenarios. These include the *"stand-alone, good PUF model"*, the *"bad PUF model"*, and the *"PUF re-use model"*:

1) In the *stand-alone, good PUF model*, we assume that there is only one single, isolated protocol execution, and that all parties faithfully generate and never manipulate PUF hardware.
2) In the *PUF re-use model*, we extend this setting, and allow adversaries multiple access to PUFs. Its mildest form is the so-called one-time posterior access model (PAM), which allows one-time access to the PUF after a given protocol, and delimits the adversary to mere CRP-measurement on the PUF.
3) In the *bad PUF model*, we allow fraudulent parties and adversaries to manipulate PUF hardware and to use so-called "bad PUFs". These are PUFs which look like a normal PUF from the outside, having a standard CRP-interface etc., but which have extra properties that allow cheating. A scenario equivalent to the bad PUF model has been introduced under the name "malicious PUFs" by Ostrovsky, Scafuro, Visconti and Wadia in an eprint paper [18], their work being independent and simultaneous to the first publication of the bad PUF model by van Dijk and Rührmair in another eprint [6].

In order to illustrate the effect of the new models, we carry out exemplary security analyses of several protocols of Brzuska et al. [1] and Ostrovsky et al. [18] in the bad PUF and PUF re-use model.

Our Results: Our analyses of existing protocols show the following outcome.

1) A recent BC-protocol of Ostrovsky et al. [18] (which we give in Appendix D) can be successfully attacked in its own attack model of "malicious" or "bad" PUFs. The attack is presented in Section III-E.
2) A recent OT protocol of Brzuska et al. [1] is insecure in the PUF re-use model and in the bad PUF model. The attacks are presented in Sections III-A and III-B.
3) A recent KE-protocol of Brzuska et al. [1] is insecure in the PUF re-use model and in the combined PUF re-use, bad PUF model. The respective attacks are presented in Sections III-C and III-D.

The above, exemplary security evaluations are carried out in full detail. In addition to that, we observe that several other

[1] We stress that the Weak/Strong PUF terminology, which was originally introduced by Guajardo, Kumar, Schrijen and Tuyls [11], is certainly not meant or to be misunderstood in a judgemental or pejorative manner.

[2] This paper has been accepted at Eurocrypt 2013 very recently, but we had only access to the eprint version [18] at the time of writing.

known PUF-protocols are insecure in the bad PUF and the PUF re-use model. Since the attacks are very similar to the abovementioned, we merely sketch them for space reasons in Section III-F. They include the following:

4) An early OT-protocol of Rührmair [22] and an early KE-protocol by van Dijk [5] are insecure in the bad PUF and the PUF re-use model (see Section III-F).
5) An OT-protocol of Ostrovsky et al. [18] is insecure in the bad PUF and the PUF re-use model (see Section III-F).
6) Two special BC-protocols of Ostrovsky et al. [18], and consequently their construction for UC-secure computation built on these two protocols, are insecure in the bad PUF model, too (see Section III-F). The attacks require the use of more complex bad PUF constructions such as Communicating PUFs and Marionette PUFs, though (see Section II-E for an explanation of the latter two).

Two important aspects should not go unnoticed. First, apart from the attack on Ostrovsky et al.'s BC protocol mentioned in item 1 above, **all** of the presented attacks are outside the original attack models of the respective papers. However, we argue in great detail in Section II why the new attack scenarios must be considered realistic, and why the protocols would be faced with them in any practically relevant settings.

Secondly, our attacks in the bad PUF model require only very mild forms of bad PUFs. The attack in item 1 utilizes a bad PUF that implements a simple linear function (see Section III-E). Furthermore, the attacks of items 2 to 5 merely require so-called Challenge-Logging PUFs and Simulatable PUFs. The only exception is the attack mentioned in item 6: It requires a more sophisticated type of PUFs, namely Communicating PUFs (or special variants of it, such as Marionette PUFs); see Section II-E.

Besides the above new findings, two already published results should be added to complete the picture:

7) A PUF-based session key exchange protocol by Tuyls and Skoric [32] has already been attacked by Rührmair, Algasinger and Jaeger [26] under conditions similar to the PUF re-use model (without explicitly using this term). Their attack partly motivated the formal introduction of the PUF re-use model in this paper.
8) There are quadratic attacks on the security of the OT- and BC-protocol of Brzuska et al. [1] which have been presented at CHES 2012 by Rührmair and van Dijk [25]. They show that the security of these protocols is not maintained if optical PUFs or electrical PUFs with challenge length of 64 bits are used in their implementation.

As indicated by the above items (1) to (8), our analysis focuses on the impact of our attack models for "advanced" PUF protocols like OT, BC and KE. The elementary PUF use as internal key storage element and in simple identification protocols [19], [20] appears less affected (see Section V).

Consequences: The findings of our analysis are somewhat alarming. They suggest that attack models and protocol design for "advanced" Strong PUF protocols should be strongly reconsidered. As PUFs are hardware systems that can have hidden extra features, new strategies become necessary here.

One possible countermeasure is to (i) allow additional computational assumptions in the protocols; (ii) assume that the PUFs can be shielded during the course of the protocol in order to prevent communication between the bad PUF and malicious parties; and (iii) to use each PUF only once, destroying it at the end of the protocol in order to prevent access by adversaries after the protocol. This path is taken by Ostrovsky et al. in their work [18]. However, there are some downsides associated with this approach: The introduction of additional computational assumption takes away some of the appeal of Strong PUFs as a new, independent cryptographic primitive. The effective shielding of PUFs until their destruction is hard to achieve in concurrent, complex environments. And, perhaps most importantly, the one-time use and destruction of the used PUFs after each protocol execution is extremely costly in practice. It constitutes a theoretically viable, but practically and commercially essentially infeasible measure.

A second option to encounter our attacks is to add two new hardware features to Strong PUFs. Firstly, one can require that Strong PUF's responses should be *"erasable"*, meaning that single responses can be "erased" (made unreadable for good). Ideally this erasure should not affect other responses; if this requirement is hard to realize in practice, then also concept similar to the logical reconfigurability of PUFs [13] may be applicable in certain settings (see Section IV). This step immunizes Strong PUF protocols against PUF re-use attacks. Secondly, Strong PUFs should be *"certifiable"*, meaning that parties holding a Strong PUF can verify that the PUF has been produced faithfully and has not been manipulated in any way afterwards. This guarantees security in the bad PUF model. The combination of both features can fully restore the applicability of Strong PUFs in concurrent, complex application environments without further restrictions (such as the above one-time use of PUFs). The implementation of these features, however, constitutes a challenging open problem that we pose to the community in this work.

Organization of this paper: In Section II we discuss and introduce various attack models for Strong PUF protocols. In Section III, we evaluate the security of many existing protocols in the new attack models. Section IV discusses the consequences of our work, in particular the need for Erasable PUFs and Certifiable PUFs. Section V summarizes the paper.

The appendix provides extra information: In Appendix A we give background on Strong PUFs to the readers who are not familiar with this concept. In Appendices B, C and D we provide some of the analyzed PUF-protocols from Brzuska et al. and Ostrovsky et al. for the convenience of the readers.

II. Attack Models for Strong PUF Protocols

Building on the general description of Strong PUF protocols in the introduction and also in Appendix A, we will now describe a number of attack scenarios for Strong PUF protocols.

A. The Stand-Alone, Good PUF Model

In the stand-alone, good PUF model, we make the following assumptions:

1) The protocol is executed only once in a stand-alone setting, meaning that the protocol is never re-run, also not any (sub-)sessions of it. The employed PUF(s) cannot be accessed or communicated with after the end of the protocol.
2) The employed PUFs are all "good PUFs", meaning that are drawn faithfully from a previously specified distribution of PUFs and are not modified in any way afterwards, neither by malicious players nor by external adversaries. They only have the properties and functionalities expected by the honest protocol participants.

It seems that several early Strong PUF protocols were more or less implicitly designed for a stand-alone, good PUF setting, for example van Dijk's key exchange scheme [5] and Rührmair's OT protocol [22]. The stand-alone model will neither be realistic nor efficiently realizable in most practical PUF-applications, but makes a clean first scenario for studying the security of PUF-protocols. For practical appliances it needs to be extended, as described below.

B. The UC-Model of Brzuska et al.

In order to model the execution of multiple PUF protocols, Brzuska, Fischlin, Schröder and Katzenbeisser [1], [2] proposed one possible method how Canetti's UC-framework [3] can be adapted to PUFs. For a detailed treatment we refer the readers to the original papers [1], [2], but summarize the features of their model that are most relevant for us below.

1) It is assumed that all used PUFs are drawn faithfully from a previously specified distribution of PUFs, a so-called "PUF-family", and are not modified in any way afterwards, neither by malicious players nor by external adversaries. They only have the properties and functionalities that honest protocol participants expect from them. This feature is in common with the above stand-alone, good PUF model.
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 [2]: 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 [2]. 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 [2]. 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 [1], [2], but are central to the security of their protocols and the validity of their security proofs.

C. The UC-Model of Ostrovsky et al.

Ostrovsky, Scafuro, Visconti and Wadia modify the UC-model of Brzuska et al. in a number of aspects in a recent eprint [18]. Among other things, they suggest an attack scenario termed "malicious PUFs". It is equivalent to the "bad PUF model" proposed independently by van Dijk and Rührmair [6], which is detailed in Section II-E of this paper; both models seem to have been developed independently and simultaneously.

The two author groups use their equivalent models for different purposes, though: Ostrovsky et al. give several protocols that are purportedly still secure under use of malicious/bad PUFs. Most of their constructions employ three extra assumptions: (i) they use *additional, classical computational assumptions* alongside with PUFs; (ii) they assume that the bad PUFs do not communicate with the malicious parties (compare Section II-E); and (iii) they assume that the PUFs are used only once, and can be kept away for good from the adversary or destroyed afterwards. On the other hand, van Dijk and Rührmair show that if one wants to design PUF-protocols that *solely* rest on the security of the employed PUFs, i.e., *without* additional computational assumptions, then the existence of malicious/bad PUFs leads to hard impossibility results.

We remark that in practice, the above assumption (iii) would have to be realized by destroying the PUF after each protocol, or by locking it away for good. In commercial applications, such a measure would probably be too costly and economically infeasible. The PUF re-use model in the next Section II-D investigates the consequences if it cannot be realized in practice.

D. The PUF Re-Use Model

Let us now step by step extend the model of Brzuska et al. [1], [2], and partly also of Ostrovsky et al. [18]. One implicit assumption of Brzuska et al. is that the adversary cannot access the PUF between different (sub-)sessions, and that the PUF is never re-used in another protocol session (see Section II-B). However, this assumption seems difficult to guarantee in many natural PUF appliances.

To see this, consider the well-established application scenario of a PUF on a bank card, which has been issued by a central authority CA and is subsequently used in different terminals [20], [19]. To be more concrete, let us assume that the PUF is repeatedly employed for a session key exchange between the CA and the smart-card/terminals. 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 it is employed in different contexts (for example when the user is paying with it), a realistic assumption is that an adversary will have *repeated* temporary physical access to the PUF between the different key exchange (sub-)sessions. However, such access is not foreseen in the models and protocols of Brzuska et al.

The example illustrates that in practice, adversaries and malicious players may gain access to the PUF at least occasionally between different (sub-)sessions. This constitutes a new, relevant attack point and motivates an extension of the model of Brzuska et al. [1]. Ostrovsky et al. [18] deal with this observation in their own manner: As described in Section II-C, they implicitly assume a one-time use of the PUF. Such one-time use, and subsequent destruction or locking away of the PUF, results in substantial practical costs, however. It constitutes a theoretically acceptable, but at the same time commercially somewhat infeasible measure. These considerations motivate the following attack model:

The PUF Re-Use Model: We assume that at least a subset of the PUFs employed in the original protocol is used on more than one occasion, i.e., not all PUFs are used only once and destroyed immediately afterwards. The adversary or malicious parties have access to the PUF more than once, for example before, after or between different protocols or protocol (sub-)sessions (if there are any).

The description leaves some detail open, the simple reason being that many differing variants of the PUF re-use model are possible. For example, one can distinguish between the type of adversarial access: (i) full physical access, where the adversary can attempt arbitrary actions on the PUF, including arbitrary measurements or active physical modification of the PUF, or (ii) CRP access, where the adversary's actions are limited to the mere measurement of CRPs. One can also differentiate the number of occasions on which access is possible; or the relative time of the access, such as before or after the attacked protocol; or the number of CRPs the adversaries can read out during his access time. One can further distinguish between different types of re-use: Is the PUF re-used by the same parties in another instance of the same protocol, or by entirely new parties in a different protocol? Instead of declining through all possible scenarios formally here, we suggest that such differentiation should be made in the respective security analyses directly.

There is only one specific instantion we would like to define explicitly here, since it has special relevance for us.

The One-Time Posterior Access Model (PAM): In the PAM, we assume that the adversary has got access to at least a subset of all PUFs employed in the original protocol on exactly one occasion after the end of the protocol (or protocol (sub-)session, if there are any), and is furthermore limited to the measurement of standard CRPs.

Please note that the PAM is arguably the mildest possible form of the PUF re-use model. Still, it suffices to successfully attack many existing schemes (see Section III).

E. The Bad PUF Model

One other central assumption in the UC-model of Brzuska et al. is that the players are not allowed to use "bad", fraudulent PUF-hardware with properties beyond the expected PUF functionality. This assumption can again be difficult to uphold in practice, as has been observed independently by Ostrovsky et al. [18] (see Section II-C).

To motivate bad PUFs, consider once more the earlier smart-card example. 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 function (PRF) or a pseudo-random number generator (PRNG) with a seed s known to the CA. If the PRF will have the same, digital input-output interface as the normal PUF, such a step will remain unnoticed. Still, it 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 assumption that the CA replaces the PUF by a PRF with a seed known to the CA, the well-known OT protocols of Rührmair [22] and Brzuska et al. [1] are no longer secure. If the CA acts as OT-receiver, for example, it can learn both bits of the OT-sender (see Section III-B for details).

Abstracting from this specific example, the general problem is that in a typical two-party protocol, one of the parties can fabricate the PUF, while the other party may only use the PUF "from the outside" via a (digital) challenge-response interface. It is hard to verify that there is no unexpected, malicious functionality on the other side of the interface. From a practical perspective, this observation is most severe for electrical Strong PUFs, which are the most widely distributed Strong PUFs today. But it also holds for *integrated* optical PUFs as given by Tuyls and Skoric [32].

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, exhibiting a input-output behavior indistinguishable from a proper PUF, but which possesses secret, additional properties that allow cheating. Its assumed similar input-output behavior shall make it infeasible to distinguish a bad PUF from a proper PUF by digital challenge-response measurements. In order to detect bad PUFs, honest parties would need to physically open the PUF-hardware and to inspect it thoroughly, as a regular and dedicated step of the protocol. While detection of bad PUFs would not even be guaranteed by such a step (adversaries would presumably develop obfuscation techniques), it would surely destroy the opened PUF, even if it was non-manipulated. In addition, the inspection step would be beyond the capabilities of an average user.

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 exchanged in classical cryptographic protocols, these objects may bring about unwanted properties. They can act as real, physical "Trojans" and other malicious hardware.

Even though there is a practically infinite number of possibilities how Strong PUFs can act, two types of bad PUFs that we focus on in this paper are (i) PUFs that are numerically simulatable by their manufacturer (but by no one else), and (ii) bad PUFs that "log" or record all challenges that have been applied to them. Both are particularly easy to implement, but suffice for attacks on existing protocols.

Simulatable Bad PUFs (SIM-PUFs): A simulatable PUF (or SIM-PUF, for short) 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. [3]

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 an underlying, "normal" 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. Please note that many alternative ways for activating the output mode of the challenge-logger are conceivable, such as radiowave triggering etc., and even entirely other forms of logging and read-out "modes" of the logger are possible (see below).

CL-PUFs can be implemented particularly easily in any *integrated* optical or electrical PUFs. But even for Pappu's optical, non-integrated PUF [20] challenge logging appears feasible. Imagine a special, transparent, additional layer on top of Pappu's light scattering token, which is altered by the incoming laser light. The alteration of the layer would

[3] The replacement of the internals of a PUF by a pseudo-random function is particularly hard to detect for any integrated PUFs (be they optical or electrical), since they communicate with external parties only via their integrated, digital CRP-interface; the PUF is never measured directly by the external parties. Such integrated PUFs constitute the clear majority of currently investigated PUFs. But even for Pappu's optical PUF, simulatability can be an issue: It is by no means ruled out that the adversary builds a light scattering token that has a particular, well-ordered structure, which leads to simple and simulatable outputs. Current protocols would not even detect if the adversary used an "empty" plastic token, which did not contain any scatterers at all, and which was trivially simulatable.

not necessarily be visible by the sheer eye, but could reveal itself only under UV-light or other special illumination. Such a sensitive layer would indicate the point of incidence (and perhaps even the angle) of the challenge, i.e., it would show some form challenge logging.

Finally, we observe that there are two fundamentally different types of CL-PUFs: PUFs that have been malicously constructed with a challenge-logger from the start; and CL-PUFs where a logger-module has been added externally by malicious parties after their construction. The former seem yet more easy to implement, but also the second type is a viable attack strategy. In any way, CL-PUFs act as real, physical Trojans: They record and store security-critical information and pass it on to the adversary when he holds possession of the PUF again.

Discussion of Potential Countermeasures: A straightforward countermeasure against bad PUFs seems to "authenticate" or "certify" the PUF in one way or the other in order 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 very problematic, and pretty much falls apart.

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 or 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. Physical inspection of the inner configuration of the PUF as a regular protocol step seems no viable possibility, as discussed in the previous paragraphs. Furthermore, if efficient methods for certifying the integrity of (PUF-)hardware existed, then the same methods could be applied to protect security modules built on classical keys, making PUFs obsolete. Once more, this makes bad PUFs a realistic and efficient method to cheat.

Brzuska et al. [1] 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, and not only external adversaries, generate and use manipulated PUFs. We comment that in a typical two-party protocol, a PUF originating from a malicious party must be considered as nothing else than an untrusted piece of hardware that stems from the adversary.

Advanced Bad PUFs: How "bad" can a PUF be? Having focused on simple features in the last section (which still suffice to attack many existing protocols), we will play with a number of more sophisticated properties now. The purpose of our discussion is to complement the picture; we will not fully work out every construction in detail.

To start with, it is of course possible to imagine bad PUFs that communicate information (e.g., wirelessly) to the malicious party. Such a *"Communicating PUF"* could transmit the challenge, the response, or both, to fraudulent parties. The transmission could be carried out in real time, or may be delayed to later, when the PUF is released from the control of the honest parties. It is relatively straightforward that such a feature destroys the security of all existing protocols. Necessary, but also very costly countermeasures were shielding the PUF during the protocol and destroying them immediately afterwards.

Another advanced bad PUF example is a PUF which transmits all challenges to the malicious party in real-time; waits for the malicious party to individually select and return a response R_{bad}; and then outputs R_{bad} (as if it was the natural response of the PUF itself). The latter type of PUF could be called the *Marionette PUF* for obvious reasons. It seems clear that there is no security benefit of using PUFs in cryptographic protocols if the adversary can use Marionette PUFs. Their employment makes PUFs useless, in the sense that for any protocol that uses PUFs and which securely implements a task T even if Marionette PUFs are employed, there will be a protocol that securely implements T and does not use the (Marionette) PUFs at all. Therefore the existence and use of Marionette PUFs must be ruled out in most advanced Strong PUF protocols such as OT, BC and KE by whatever means. One potential, but again costly countermeasure to prevent Marionette PUFs would be the shielding of the PUF during the entire course of the protocol.

A third example are bad PUFs that adapt or alter their response behavior over time. This adaption could be a function of the challenge that is applied to them, or a function of all previous challenges. Other variants of adaptive bad PUF behavior include the following: (i) The PUF could automatically alter its response behavior after a certain time period t_0. This means that the malicious party can influence the protocol by delaying the protocol; note that this is explicitly allowed in the UC-model. (ii) The PUF could change its CRPs upon a wireless triggering signal it receives. (iii) The PUF could even change upon a certain, triggering *challenge* that is applied to it. This allowed the malicious party to influence the bad PUF even while it is not in her possession, simply by causing the honest party to apply a certain challenge to the PUF.

A final example are bad PUFs that implement arbitrary digital functions f with special, fraudulent properties. Simulatable PUFs (where f is simulatable and the simulation code is known to the malicious party) are one special case of this approach. But the function f could have other handy properties for the adversary. For example, it might be a function for which the computation of inverses is simple. This case is actually relevant for our attack in Section III-E.

Many other examples of advanced bad PUFs are conceivable. Actually, any such bad PUF types have to be taken into consideration when the security of a PUF protocol is analyzed. But since the earlier, simpler types of SIM-PUFs and CL-PUFs already suffice for attacking many protocols, we will not deal too much with advanced bad PUFs further in this paper.

A Final Thought on Bad PUFs: Let us conclude this section by a general thought. Why are bad PUFs so powerful? Consider the following line of thought: Suppose that a PUF-protocol utilizes some property $\mathcal{P}$ of the employed PUF to achieve its security. Then there will (almost with certainty) be a bad PUF which is hard to recognize from the outside, but which *does not* possess the property $\mathcal{P}$. The security of the protocol and the validity of the proof will no longer be guaranteed if the adversary uses this bad PUF not possessing $\mathcal{P}$. This makes bad PUF a broadly applicable method of cheating. The cost of implementing the imagined bad PUF type determines how practically relevant the resulting attack is; we focused on relatively easily implementable variants of bad PUFs in this paper.

III. Security Evaluations in the PUF Re-Use and Bad PUF Model

We will now conduct three detailed, exemplary security analyses in the new attack models. We selected the PUF-based OT- and KE-protocol of Brzuska et al. from Crypto 2011 [1] and the recent BC-protocol by Ostrovsky et al. [18] to this end. The protocols are given in a simplified form in Appendices B, C and D for the convenience of the readers. The notation employed in our attacks actually refers to these appendices. We would like to stress that the first two protocol by Brzuska et al. are secure in their own, original attack model (apart from a recent attack on Brzuska's OT-Protocol by Rührmair and van Dijk [25]). But, as argued earlier, the protocols would likely be faced with the PUF re-use model and the bad PUF model once they were used in practice. In opposition to this, the BC-protocol of Ostrovsky et al. is actually attacked in their own, original "malicious" PUF model.

A. OT-Protocol of Brzuska et al. in the PUF Re-Use Model

We start by analyzing the OT-Protocol of Bruzska et al. [1] (see Protocol 1 in Appendix B) in the PUF re-use model, or, to be more precise, in the mildest form of the PUF re-use model, the PAM. Our attack rests on the following *assumptions*:

1) After the initialization phase of the OT-Protocol 1, 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. (Note that this assumption is derived from the PAM.)

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 $S_1\ (:= s_1 \oplus r_1)$. Thereby 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 the role of Eve can also be played by a malicous receiver. Interestingly, 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 rest on the employed PUF.

B. OT-Protocol of Brzuska et al. in the Bad PUF Model

Let us now describe an attack on the OT-Protocol of Brzuska et al. [1] (see Protocol 1 in Appendix B) in the bad PUF model, which works under the following single assumption:

1) The receiver can hand over a simulatable bad PUF instead of a normal 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 1 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.

The Sender hence learns both strings s_0 and s_1, breaking the security of the protocol. We comment that the attack only requires the use of simulatable PUFs by the receiver, which are particularly easy to implement.

C. KE-Protocol of Brzuska et al. in the PUF Re-Use Model

We describe below how the KE-Protocol of Brzuska et al. [1] (see Protocol 2 in Appendix C) can be attacked in the PUF re-use model, or more, precisely, in its mildest form, the PAM. The attack is quite straightforward and rests on the following assumptions:

1) After the initialization phase of Protocol 2, 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 Alice and the Bob 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 the exchanged key K. The attack is relatively obvious and works as follows:

1) When the subsession `ssid` is run, Eve eavesdrops step 2 and learns the values c and d.
2) When Eve has got physical access to the PUF after subsession `ssid`, she applies the challenge c to the PUF, measures the (noisy) response r', and derives the secret st from r' by the help of d.

As $st = K$ in subsession `ssid`, this breaks the security of this subsession.

D. KE-Protocol of Brzuska et al. in the Combined PUF Re-Use and Bad PUF Model

Let us continue examining the security of the KE-Protocol of Brzuska et al. [1] (Protocol 2 in Appendix C). Since in a simple stand-alone scenario neither Alice nor Bob have an incentive to use bad PUFs, this is a welcome opportunity to illustrate the impact of a combined attack model: namely a combination of the PUF re-use and the bad PUF model.

We make the following assumptions:

1) The KE protocol is executed between Alice and Bob (including an initialization phase and an arbitrary number of subsessions), and later between Bob and Claire (again including an initialization phase *with the same PUF* and later subsessions).
2) Alice plays maliciously, and uses a challenge-logging PUF in her initialization phase.

Under this assumption, Alice can learn the key exchanged by Bob and Claire as follows:

1) When the PUF is in transition from Bob to Claire in step 3 of the initialization phase of their protocol, Alice gains physical access to the PUF.
2) Alice reads out the last previously applied challenge c, applies it to the PUF, and obtains response r.
3) In the next subsession phase, Alice intercepts the helper data d that is sent from Bob to Claire in step 2.
4) Alice utilizes her knowledge of r and d to infer $st = K$.

Alice hence learns the key K exchanged by Bob and Claire, breaking the protocol. Let us mention a few simple modifications of the attack: Alice could alternatively use a simulatable PUF (instead of a CL-PUF), leading to a similar attack. If the PUF is obtained by Alice from a third party manufacturer, then the manufacturer can mount the same attack by using CL- or simulatable PUFs. Finally, if an external adversary Eve is able to add a challenge logger while the PUF is in transit from Alice to Bob, then she can derive both Alice's and Bob's key as well as Bob's and Claire's key by reading out the challenge logger when the PUF is in transit from Bob to Claire. The details of these variants are similar to the attack above and are left to the reader.

E. An Unconditional BC-Protocol of Ostrovsky et al. in the Bad PUF Model

Ostrovsky et al. [18] describe an unconditional BC-protocol (i.e., one that does not use any additional computational assumptions) in Fig. 6 of their paper (see Protocol 3 in Appendix D). The protocol is purportedly secure under the use of malicious/bad PUFs. However, we describe below a very simple bad PUF type that allows cheating. The bad PUF is not even required to communicate with the malicious party.

Our attack makes the following assumption:

1) The committer $\mathsf{C}_{\mathsf{uncon}}$ uses/initializes a bad PUF in step 1 of the protocol (instead of a good PUF). The bad PUF has the following feature: Instead of outputting essentially random and unpredictable responses, it implements a linear function in its challenge-response behavior: For a fixed value X, it maps every challenge C to the output $R := C \oplus X$.

The attack then proceeds as follows:

1) The committer $\mathsf{C}_{\mathsf{uncon}}$ initializes the above bad PUF in step 1 of the protocol.
2) In order to open to bit $b = 0$ in the decommitment phase, the committer sends the challenge $st \oplus X$. In

order to open to bit b=1, the committer sends the challenge $r \oplus st \oplus X$.
In the first case, the (bad) PUF outputs $st \oplus X \oplus X = st$, meaning that the receiver accepts the opening phase for $b = 0$.
In the second case, the (bad) PUF outputs $r \oplus st \oplus X \oplus X = r \oplus st$, implying that the receiver accepts the opening phase for $b = 1$.

This breaks the security of the protocol under the assumption that bad/malicious PUFs can be used. The attack does not require elaborate bad PUF behavior (such as communication between the PUF and the malicious party), as already remarked above, but can be implemented relatively simply. Instead of using the specific linear function mentioned above, the attack can also be built on any other bad PUF that implements an arbitrary, efficiently invertible function f in its challenge-response behavior.

F. Security of Other Protocols in the PUF Re-Use Model and Bad PUF Model

For reasons of brevity, we focused on the three above protocols in our detailed security analysis. Other Strong PUF protocols for OT, BC or KE can be attacked in similar manners. Since the attacks are analog to the work in the above sections, we merely sketch and summarize them in the following list:

1) The OT-protocol of Rührmair [22] is no longer secure in the bad PUF and PUF re-use model, and similar considerations hold for the key exchange protocol of van Dijk [5]. This can be seen relatively easily, since the attacks are essentially equivalent to the attacks in the last subsetions on Brzuska et al.'s OT and KE protocol.
2) It is not too difficult to see that the unconditional OT-protocol of Ostrovsky et al. for honest PUFs (see Fig. 7 of [18]) is not secure in the PUF re-use model. If the receiver of the protocol gets access to the used PUFs after the end of the protocol, he can learn both strings s^0 and s^1.
Furthermore, if the sender uses bad, challenge-logging PUFs instead of honest PUFs as $\mathsf{sid}^{\mathsf{S}}_1, \dots, \mathsf{sid}^{\mathsf{S}}_{2k}$, then he can obviously learn the value of the b_i, which allows him to derive the the choice bit b from the values b'_{i_j} which he receives in step 4 of the protocol. Actually, even something weaker suffices: If only one of the PUFs $\mathsf{sid}^{\mathsf{S}}_j$ for $j \in S$ is challenge logging, then b is revealed. Since $S \subset [2k]$ is a randomly chosen subset of size k, the latter condition can be enforced by merely making $k + 1$ of the $2k$ PUFs challenge logging. In other words, the attack also works if only a strict subset of all PUFs are bad.
3) The statistically hiding, straight-line extractable bit commitment scheme of Fig. 10 of Ostrovsky et al. [18], and the statistically binding, straight-line extractable equivocal commitment scheme of Fig. 11 of the same paper, can be attacked by communicating bad PUFs, which maliciously transfer the challenges applied to the PUF in the commit phase to the receiver. This allows the receiver to learn the committed bit before the reveal phase.
We stress once more that Ostrovsky et al. seem to implicitly assume that there is no communication between the malicious party and the PUF, i.e., we are again extending the original attack model of Ostrovsky et al. here. However, as discussed earlier, Communicating PUFs seem hard to prevent in certain settings. If they are considered realistic, then also the construction for UC-secure computation of Ostrovsky et al., which is built on the commitment schemes in Figs. 10 and 11 of [18], breaks down.

G. Summary of Our Security Discussion

To summarize, all Strong PUF protocols for OT, BC and KE examined in this paper can be attacked in variants of the PUF re-use model, the bad PUF model, or the combined PUF re-use, bad PUF model. Only one of these attacks (see item 3 of Section III-F above) requires Communicating PUFs, which are somewhat complex. The majority of attacks, however, can be carried out in simple variants of the bad PUF model, using simulatable or challenge-logging PUFs, or straight away in the ordinary PUF re-use model.

We stress again that most of the attacks work outside the attack scenarios and communication models of the original papers, but we argued in Section II why we consider the new models realistic. One notable exception is the attack on Ostrovsky's unconditional bit commitment scheme in the malicous PUF model (see Section III-E), which actually works in the original attack model of Ostrovsky et al.

The authors of this paper are not aware of any PUF protocols for OT, BC or KE which can withstand all said attack models, and in which (i) plain Strong PUFs with no additional hardware properties are used, (ii) no additional assumptions (set-up assumptions, classical computational assumptions, etc.) apart from the security (i.e., unpredictability) of the Strong PUF are made. This illustrates the acuteness of re-thinking current PUF protocol design.

IV. CONSEQUENCES, OR: THE NEED FOR ERASABLE AND CERTIFIABLE PUFS

What are the consequences of the observations of the last sections? The first and foremost implication is that attack models for PUF protocols should be reconsidered. PUFs are different from other cryptographic primitives in that they are real pieces of hardware that can have all kinds of malicious properties. Future protocol design and security analyses must take this into account.

One potential route to evade some of our attacks has been considered by Ostrovsky et al. in [18]. They combine three steps to construct secure PUF-protocols in the presence of malicious/bad PUFs: (i) They allow additional, standard computional assumptions in the protocols. (ii) They assume that the PUF cannot communicate with the malicious party, in particular, that the PUF is no Marionette PUF and no Communicating PUF. (iii) They assume a strict one-time use of the PUF; potentially malicious parties must be kept away from the PUF after it has been used. Measures (ii) and (iii) essentially must be realized by effectively shielding the PUF continuously until it is destroyed at the end of its one-time use. These are certainly very costly and non-trivial measures. They lead us to the question whether other approaches for fighting the PUF re-use model and bad PUFs exist in practice.

Erasable and Certifiable PUFs: Two other, direct countermeasures against the PUF re-use model and bad PUFs are so-called Erasable and Certifiable PUFs. Erasable PUFs are Strong PUFs with the additional feature that single responses can be erased from the PUF (i.e., made impossible to read out forever) without affecting any of the other responses. Erasable PUFs have been considered for the first time by Rührmair, Algasinger and Jaeger in [26], who also suggest an implementation based on so-called crossbar structures. This implementation is very area consuming, however. Area efficient implementations have not been suggested up to this date.

In order to better understand the challenges and the novelty behind Erasable PUF design, consider two of the currently most established Strong PUF designs: Arbiter PUFs [30] and optical PUFs [20]. In both designs, many subparts of the PUF interact in order to generate a response. If one response shall be altered or erased, at least one of the subparts must be changed. In the example of optical PUFs, certain subparts of the scattering platelet would need to be modified; in the case of the Arbiter PUF, at least one internal delay value would need to be altered. But this will necessarily also affect and modify other responses, contradicting the requirements of an Erasable PUF. Reconfigurable PUFs [16] are unsuited as Erasable PUFs for the same reason: Their reconfiguration operation by definition alters all responses of the PUF in one step. This makes any previously collected CRPs of the PUF invalid.

If the area efficient, direct implementation of Erasable PUFs remains difficult in the future, then an alternative strategy could be equipping Strong PUFs with a surrounding control logic. This logic is supposed to guard and regulate the access to the Strong PUF's challenge-response interface; such constructions are also known as Controlled PUFs [8]. Along these lines, one could construct "Logically Erasable" PUFs by letting the control logic maintain some record of the previously applied and of the erased challenges (e.g., in the form of an authenticated hash tree). Also Logically Reconfigurable PUFs (LR-PUFs) as introduced by Katzenbeisser et al. [13] can be an option in this context. They allow the manufacturer of the PUF to collect a CRP-list that remains valid even after many reconfiguration operations. This may suffice to ensure the security of certain protocols in the PUF re-use model. We remark, however, that such versions of Controlled PUFs introduce additional assumptions, for example that it is impossible to circumvent, modify or tamper the control logic around the underlying Strong PUF.

Certifiable PUFs, on the other hand, are PUFs that allow an offline certification of the fact that they have only those properties that the honest parties expect from them. It is possible to verify that they have been drawn faithfully from the expected PUF distribution, and that they have not been modified by anyone in any way afterwards. We argued already in Section II-E why it is important that such a certification can be carried out offline: Communication with a trusted authority upon every protocol execution (in order to certify the PUF) makes the use of PUFs obsolete. One could then implement the desired functionalities easier by using the trusted authority itself. Currently, however, no measures whatsoever have been considered in the literature how such authentication can be achieved.

The combination of certifiability and erasability (or variants such as logical erasability/reconfigurability) in a single piece of hardware therefore poses a highly relevant, but very challenging open problem to the PUF hardware community. It should be resolved in order to restore the full applicability of Strong PUFs as a general, broadly, and efficiently usable cryptographic tool. It would allow PUF protocols in complex environments without additional computational assumptions, and without an economically unrealistic one-time use of PUFs.

V. Summary and Future Work

We introduced a number of new attack models for Strong PUF protocols in this paper, including the *"PUF re-use model"* and the *"bad PUF model"*. These models, so we argued, constitute practically relevant and hard-to-detect attack strategies, and are strongly relevant for practical PUF usage scenarios.

We then illustrated the power of the new models by analyzing the security of several known protocols. The results were already summarized in detail in Section I. In short, all analyzed oblivious transfer (OT), bit commitment (BC) and key excahnge (KE) protocols for Strong PUFs can be attacked successfully in the bad PUF model and/or the PUF re-use model. This includes schemes by Rührmair [22], van Dijk [5], Brzuska et al. presented at Crypto 2011 [1], and Ostrovsky et al. [18]. With one exception, where so-called Communicating PUFs are required, all attacks in the bad PUF model only utilize very simple types of bad PUFs, such as simulatable PUFs and challenge-logging PUFs. The attacks in the ordinary PUF re-use model are even simpler to execute, and do not require physical modification of PUFs at all.

We remark **once more** that our attacks leave the original attack models of the protocols (with the exception of our attack on Ostrovsky et al.'s BC protocol of Section III-E, which works in their own, original model). Still, our attack models seem realistic, and indeed closely follow practical usage scenarios of PUFs. Depending on the exact application, the protocols would likely be faced with them once they were used in practice. This implies that current attack models and design strategies for advanced PUF protocols such as OT, BC or KE must strongly be re-thought.

Two potential classes of countermeasures against our attacks were analyzed in Section IV: The first is the employment of classical computational assumptions in combination with a strict one-time use of PUFs and shielding of the PUFs against communication with the malicious party until its destruction [18]. This step maintains the usability of standard Strong PUFs in advanced settings and in the presence of bad PUFs, but is practically very costly and seems difficult to realize. Furthermore, it takes away some of the appeal of PUFs as a new, independent, and post-quantum cryptographic primitive that enables advanced protocols without classical computational assumptions.

A second possibility, that would restore the usability of PUFs in complex application settings without any restrictions, is the use of Certifiable and Erasable PUFs. These are PUFs which can be certified offline for their genuineness, and for the fact that they have no other features than those expected by the honest parties ("certifiability"); and PUFs that allow the selective erasure of single PUF responses without affecting other responses ("erasability"). Without presenting a formal proof of this claim in this paper, they seem to allow efficient and secure PUF protocols whose security is built on the unpredictability of the PUF alone, without requiring additional computational assumptions. These novel PUF types could maintain the status of Strong PUFs as a general, new cryptographic primitive. If Erasable PUFs maintain hard to realize in practice, then also variants such as logical erasability/reconfigurability [13] could be interesting in our context. In order to fight both the bad PUF and the PUF re-use model, however, erasability (or variants of it) and certifiability have to be combined in a single piece of hardware. No strategies for this exist in the current literature.

Relation to PUF-Based Key Storage and Strong PUF-based Identification: Apart from their use in basic cryptographic protocols, a second established application of PUFs is their usage as (tamper-sensitive) key storage element. This application has at times been termed a "physically obfuscated key" or POK [7], sometimes also a "Weak PUF" [11]. We stress that this application scenario is not the topic of our paper. Independent of whether such an assumption is considered realistic or not, POKs explicitly suppose that the PUF's responses remain internal forever, and can only be accessed by the system itself to derive an internal secret key. This makes this PUF-type unusable for the type of protocols considered in this paper; and at the same time, it makes the attacks in the PUF re-use model meaningless. Also the bad PUF model seems obsolete: PUF-based key storage assumes that the manufacturer in a secure set-up phase can read out the key derived from the PUF, and uses it later on in communication with the PUF. This presupposes some basic trust in the manufacturer in the first place, since the secret key is shared with him from the beginning. The exact relation between POKs and the bad PUF model will be the topic of future analysis.

Something similar holds for the common Strong PUF based identification protocol by Pappu et al. [20], [19]. The use of bad PUFs here appears less relevant, and the PUF re-use model does not seem to pose a significant threat. On the other hand, a manufacturer who uses a simulatable PUF can later impersonate the PUF-carrying hardware. The exact implications of our attack models on PUF-based identification were not the topic of this paper, and are left for future investigations.

Future Work: We expect two strands of substantially new research to emerge from our findings. The first will be concerned with the theory behind Strong PUF protocols: New attack models and security definitions must be developed, for example in the context of the UC-framework. They must include the formal definition of Erasable PUFs (and variants such as Logically Erasable/Reconfigurable PUFs) and Certifiable PUFs, and the investigation of "PUF attestation" as standard protocol step. New security proofs will need to be led in these environments. Finally, the exact implications of our attack models for other PUF applications than OT, BC and KE must be determined.

The second strand of research regards PUF hardware, and concerns the development of efficient Erasable and Certifiable PUFs. As briefly addressed in Section IV, combining these two features in a single piece of hardware efficiently is highly non-trivial. The same holds for combinations of logical erasability/reconfigurability and certifiability. We would like to pose these problems as central future challenges to the PUF hardware community in this work.

Acknowledgements

The authors would like to thank Jürg Wullschleger for enjoyable discussions and his contributions on the bad PUF model and challenge-logging PUFs.

References

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

[2] C. Brzuska, M. Fischlin, H. Schröder, S. Katzenbeisser: *Physical Unclonable Functions in the Universal Composition Framework.* Full version of the paper. Cryptology ePrint Archive, Report 2011/681, 2011. Downloaded March 2012.

[3] R. Canetti: *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. FOCS 2001: 136-145.

[4] Q. Chen, G. Csaba, P. Lugli, U. Schlichtmann, U. Rührmair: *The Bistable Ring PUF: A new architecture for strong Physical Unclonable Functions*. HOST 2011: 134-141

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

[6] M. van Dijk, U. Rührmair: *Physical Unclonable Functions in Cryptographic Protocols: Security Proofs and Impossibility Results.* Cryptology ePrint Archive, Report 2012/228, 2012. Downloaded April 2012.

[7] B. Gassend, *Physical Random Functions*, MSc Thesis, MIT, 2003.

[8] B. Gassend, M. van Dijk, D.E. Clarke, E. Torlak, S. Devadas, P. Tuyls: *Controlled physical random functions and applications.* ACM TISSEC 10(4), 2008.

[9] B. Gassend, D. E. Clarke, M. van Dijk, S. Devadas: *Silicon physical random functions*. ACM Conference on Computer and Communications Security 2002: 148-160

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

[11] J. Guajardo, S. S. Kumar, G. J. Schrijen, P. Tuyls: *FPGA Intrinsic PUFs and Their Use for IP Protection*. CHES 2007: 63-80

[12] D. E. Holcomb, W. P. Burleson, K. Fu: *Initial SRAM state as a fingerprint and source of true random numbers for RFID tags.* In: In Proceedings of the Conference on RFID Security, 2007.

[13] S. Katzenbeisser, Ü. Koçabas, V. van der Leest, A.-R. Sadeghi, G. J. Schrijen, C. Wachsmann: *Recyclable PUFs: Logically Reconfigurable PUFs*. Journal of Cryptographic Engineering 1(3): 177-186 (2011)

[14] J. Kilian: *Founding cryptography on oblivious transfer*. STOC, 1988

[15] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen, P. Tuyls: *The Butterfly PUF: Protecting IP on every FPGA*. HOST 2008: 67-70

[16] K. Kursawe, A. R. Sadeghi, D. Schellekens, P. Tuyls, B. Skoric: *Reconfigurable physical unclonable functions – Enabling technology for tamper-resistant storage*. HOST 2009: 22-29.

[17] 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, 2004.

[18] R. Ostrovsky, A. Scafuro, I. Visconti, A. Wadia: *Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions*. Cryptology ePrint Archive, Report 2012/143, 2012. First version downloaded in April 2012. Throughout our paper, we refer to the numbering of figures and protocols of the latest version of Ostrovsky et al. that was available at the time of preparing our camera ready paper. This latest version stems from Nov. 14, 2012.

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

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

[21] R. Rivest: *Illegitimi non carborundum*. Invited keynote talk, CRYPTO 2011.

[22] U. Rührmair: *Oblivious Transfer based on Physical Unclonable Functions*. TRUST 2010, pp. 430 - 440, Springer 2010.

[23] 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.

[24] U. Rührmair, S. Devadas, F. Koushanfar: *Security based on Physical Unclonability and Disorder*. In: M. Tehranipoor and C. Wang (Editors): Introduction to Hardware Security and Trust. Springer, 2011

[25] U. Rührmair, M. van Dijk: *Practical Security Analysis of PUF-based Two-Player Protocols*. Cryptographic Hardware and Embedded Systems (CHES 2012), Springer, 2012.

[26] U. Rührmair, C. Jaeger, M. Algasinger: *An Attack on PUF-based Session Key Exchange and a Hardware-based Countermeasure: Erasable PUFs*. Financial Cryptography (FC 2011), Springer, 2011.

[27] U. Rührmair, C. Jaeger, M. Bator, M. Stutzmann, P. Lugli, and G. Csaba: *Cryptographic Applications of High-Capacity Crossbar Memories*. IEEE Transactions on Nanotechnology, 2011.

[28] 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 (CCS'10), 2010.

[29] U. Rührmair, J. Sölter, F. Sehnke: *On the Foundations of Physical Unclonable Functions.* Cryptology ePrint Archive, Report 2009/277, 2009.

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

[31] P. Tuyls, G. J. Schrijen, B. Skoric, J. van Geloven, N. Verhaegh, R. Wolters *Read-Proof Hardware from Protective Coatings.* CHES 2006, pp. 369-383, 2006.

[32] 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.

Appendix

A. Strong PUFs

Different subtypes of PUFs exist (see [28], [29], [24]), each with their own security properties and applications. Strong PUFs are an important and central of these subtypes. They have also been called Physical Random Functions due to their similarity with the more classical Pseudo-Random Functions [10]. A Strong PUF is a PUF with the following features (for formal definitions see [29], [23], [1]):

1) *Public CRP interface:* Its challenge-response mechanism is publicly accessible. Everyone who holds a Strong PUF can apply challenges to it and read out the corresponding responses.
2) *Large CRP set:* It has a very large number of challenges, ideally exponentially many in some system parameter, such as the system's physical size or the challenge length. Together with the finite read-out speed of the Strong PUF, the large number of challenges makes it impossible to read out all CRPs in a limited time, such as days or even weeks.
3) *Unpredictability:* The CRP-behavior of Strong PUFs is so complex that it cannot be modeled or machine learned or otherwise predicted. An adversary who knows a large subset of all CRPs nevertheless cannot build a model that allows him to correctly predict the response to a randomly chosen, previously unknown challenge with high probability.

The above features imply that only the very party who currently holds possession of a Strong PUF can determine the correct response to a randomly chosen challenge with high probability, even if the PUF has been in the possession of other parties before. This observation can be exploited cryptographically in various ways, as we will see later in this paper. Typical examples of Strong PUFs include Pappu's optical PUF [19], [20], the Arbiter PUF [9], [30], the Crossbar PUF [27], and the Bistable Ring PUF [4]. Modeling Attacks on Strong PUFs have been reported, among other places, in [28]. The authors show how to attack Arbiter PUFs and variants up to a substantial level of size and complexity, but at the same time indicate that a sufficiently large number (≥ 8) of XORed Arbiter PUFs of sufficient bitlength (≥ 64) is resilient against current modeling strategies.

One advantage of Strong PUFs over other types of PUFs (such as Weak PUFs/POKs, see again [28]) is that their responses do not need to remain secret, and do not require protection inside the embedding hardware.

B. OT-Protocol of Brzuska et al.

The OT protocol of Brzuska et al. [1] 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 $\mathsf{PUF}: \{0,1\}^\lambda \rightarrow \{0,1\}^{rg(\lambda)}$. We often use this model throughout this paper, too. In the upcoming protocol, they furthermore assume that $rg(\lambda) = \lambda$, i.e., that the PUF implements a function $\mathsf{PUF}: \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda$ (compare [1], [2]).

Protocol 1: PUF-based OT by Brzuska et al. ([1], 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.

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 protocols: Indeed, all protocols discussed in this paper do require PUFs with a large number of challenges, a publicly accessible challenge-response interfaces, and an unpredictable CRP-behavior (or, in other words, *Strong PUFs*).

Furthermore, please note that no physical transfer of the PUF and no adversarial access is envisaged during the subsessions of the protocol, as already indicated in Section II-B.

C. KE-Protocol of Brzuska et al.

Together with CRP-based identification, key exchange (KE) was among the first security applications suggested for PUFs. Pappu et al. were the first to mention *"key establishment"* as a potential PUF application [20], and van Dijk gives the first concrete protocol in a patent writing [5]. The KE protocol of Brzuska et al. [1] picks up these known approaches. We again describe it in a simplified form, partly without the UC-notation and merely with a high-level description of error correction. The key exchange is carried out between two parties, let us call them Alice and Bob.

Protocol 2: PUF-BASED KEY EXCHANGE ([1], 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) Alice holds a PUF which has been drawn from the family $\mathcal{P}$.
2) Repeat N times:
   - Choose a challenge c at random, measure the response r of the PUF, create helper data d, and extract a secret st from r. Add the tuple (c, r, st, d) to the list $\mathcal{L}$.
3) Alice sends the PUF to Bob.

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

1) Alice picks a tuple (c, r, st, d) from the list $\mathcal{L}$ at random.
2) Alice sends (c, d) to Bob over the authenticated binary channel.
3) Bob measures a (possibly noisy) response r' to the challenge c. He uses the helper data d to recover the same secret st as the Server.
4) Both Alice and Bob set their key $K = st$. Alice erases the tuple (c, r, st, d) from the list $\mathcal{L}$.

Comments: For the same reasons as discussed in Section B, the above KE protocols assumes (and indeed requires) a Strong PUF. If the PUF has only got a small CRP-set, then the adversary can fully read out all CRPs when the PUF is in transition from Alice to Bob. Furthermore, no adversarial access is foreseen or allowed between the different subsessions of the protocol.

D. An Unconditional BC-Protocol of Ostrovsky et al. in the Malicious/Bad PUF Model

Ostrovsky et al. [18] give an unconditional BC-protocol (i.e., one that does *not* use computational assumptions) in Fig. 6 of their paper. The protocol purportedly is secure in the malicious PUF model (however, we show an attack on the protocol in this very model in Section III-E). We provide the protocol below.

Protocol 3: PUF-BASED BC IN THE MALICIOUS PUF MODEL BY OSTROVSKY ET AL. [18]

Committer's Input: Bit $b \in \{0, 1\}$.

Commitment Phase

1) $\mathsf{C}_{\mathsf{uncon}} \Rightarrow \mathsf{R}_{\mathsf{uncon}}$: Committer sends $(\mathsf{init}_{\mathsf{PUF}}, \mathtt{normal}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}})$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtains response $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$. Committer uniformly selects a query $q \in \{0, 1\}^n$ and sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}}, q)$ and receives response $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$. Committer obtains $(st, p) \leftarrow \mathsf{FuzGen}(a)$, and sends p to $\mathsf{R}_{\mathsf{uncon}}$. Committer sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}}, \mathsf{R}_{\mathsf{uncon}})$ to $\mathcal{F}_{\mathsf{PUF}}$.
2) $\mathsf{C}_{\mathsf{uncon}} \Leftarrow \mathsf{R}_{\mathsf{uncon}}$: Receiver receives p' from the committer and $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{C}_{\mathsf{uncon}})$ from $\mathcal{F}_{\mathsf{PUF}}$. It uniformly chooses $r \in \{0, 1\}^l$ and sends it to the committer.
3) $\mathsf{C}_{\mathsf{uncon}} \Rightarrow \mathsf{R}_{\mathsf{uncon}}$: If $b = 0$, committer sends $y = st$ to the receiver. Else it sends $y = r \oplus st$.

Decommitment Phase

1) $\mathsf{C}_{\mathsf{uncon}} \Rightarrow \mathsf{R}_{\mathsf{uncon}}$: Committer sends (b, q) to receiver.
2) $\mathsf{C}_{\mathsf{uncon}} \Leftarrow \mathsf{R}_{\mathsf{uncon}}$: Receiver receives (b', q') from the committer and sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathsf{R}_{\mathsf{uncon}}, q')$ to $\mathcal{F}_{\mathsf{PUF}}$ and obtains $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q', a')$. It then computes $st' \leftarrow \mathsf{FuzRep}(a', p')$. If $b = 0$, it checks if $st' = y$. Else, it checks if $st' = y \oplus r$. If the check passes, it accepts, else it rejects.

Comments: We again stress that the above protocol requires a Strong PUF: If the PUF has only a small set of CRPs, the Committer can read out all CRPs and subsequently may open his commitment in both ways without being caught.