Universally Composable Secure Computation with (Malicious) Physically Uncloneable Functions

Rafail Ostrovsky[1,2], Alessandra Scafuro[1], Ivan Visconti[3], and Akshay Wadia[1]

[1] Department of Computer Science, UCLA, USA
[2] Department of Mathematics, UCLA, USA
{rafail,scafuro,awadia}@cs.ucla.edu
[3] Dipartimento di Informatica, University of Salerno, ITALY
visconti@dia.unisa.it

Abstract. Physically Uncloneable Functions (PUFs) [28] are noisy physical sources of randomness. As such, they are naturally appealing for cryptographic applications, and have caught the interest of both theoreticians and practitioners. A major step towards understanding and securely using PUFs was recently taken in [Crypto 2011] where Brzuska, Fischlin, Schröder and Katzenbeisser model PUFs in the Universal Composition (UC) framework of Canetti [FOCS 2001]. A salient feature of their model is that it considers *trusted* PUFs only; that is, PUFs which have been produced via the prescribed manufacturing process and are guaranteed to be free of any adversarial influence. However, this does not accurately reflect real-life scenarios, where an adversary could be able to create and use malicious PUFs.
The goal of this work is to extend the model proposed in [Crypto 2011] in order to capture such a real-world attack. The main contribution of this work is the study of the Malicious PUFs model. To this end, we first formalize the notion of "malicious" PUFs, and extend the UC formulation of Brzuska et al. to allow the adversary to create PUFs with *arbitrary* adversarial behaviour. Then, we provide positive results in this, more realistic, model. We show that, under computational assumptions, it is possible to UC-securely realize any functionality.

1 Introduction

The impossibility of secure computation in the universal composability framework was proved first by Canetti and Fischlin [9], and then strengthened by Canetti et al. in [10]. Impossibility of even weaker notions has been proved in [1, 5, 16].

As a consequence, several setup assumptions, and relaxations of the UC framework have been proposed to achieve UC security [4, 11, 19, 29].

In recent years, researchers have started exploring the use of secure hardware in protocol design. The idea is to achieve protocols with strong security guarantees (like UC) by allowing parties to use hardware boxes that have certain

security properties. An example of the kind of security required from such a hardware box is that of *tamper-proofness*; i.e., the receiver of the box can only observe the input/output behaviour of the functionality that the box implements. This property was formalized by Katz in [20], and it was shown that UC security is possible by relying on the existence of tamper-proof programmable hardware tokens, and computational assumptions. Smart cards are well understood examples of such tokens, since they have been used in practice in the last decades. Several improvements and variations of Katz's model have been then proposed in follow up papers (e.g., [17]).

Spurred by technological advances in manufacturing, recently a new hardware component has gained a lot of attention: Physically Uncloneable Functions (PUFs) [27,28]. A PUF is a hardware device generated through a special physical process that implements a "random" function[4] that depends upon the physical parameters of the process. These parameters can not be "controlled", and producing a clone of the device is considered infeasible[5] . Once a PUF has been constructed, there is a physical procedure to query it, and to measure its answers. The answer of a PUF depends on the physical behavior of the PUF itself, and is assumed to be unpredictable, or to have high min-entropy. Namely, even after obtaining many challenge-response pairs, it is infeasible to predict the response to a new challenge.

Since their introduction by Pappu in 2001, PUFs have gained a lot of attention for cryptographic applications like anti-counterfeiting mechanisms, secure storage, RFID applications, identification and authentication protocols [14, 15, 18, 18, 22, 33, 34]. More recently PUFs have been used for designing more advanced cryptographic primitives. In [31] Rührmair shows the first construction of Oblivious Transfer, the security proof of which is later provided in [32]. In [3], Armknecht et al. deploy PUFs for the construction of memory leakage-resilient encryption schemes. In [23] Maes et al. provide construction and implementation of PUFKY, a design for PUF-based cryptographic key generators. There exist several implementations of PUFs, often exhibiting different properties. The work of Armknecht et al. [2] formalizes the security features of physical functions in accordance to existing literature on PUFs and proposes a general security framework for physical functions. A survey on PUF implementations is given in [24]. Very recently in [21] Katzenbeisser et al. presented the first large scale evaluation of the security properties of some popular PUFs implementations (i.e., intrinsic electronic PUFs).

Modeling PUFs in the UC framework. Only very recently, Brzuska et al. [7] suggested a model for using PUFs in the UC setting that aims at abstracting real-world implementations. The unpredictability and uncloneability properties are modeled through an ideal functionality. Such functionality allows only the

[4] Technically, a PUF does not implement a function in the mathematical sense, as the same input might produce different responses.

[5] SRAM PUFs (memory-based PUFs) might be cloneable according to recent finding [6].

creation of trusted PUFs. In [7] PUFs are thought as non-PPT setup assumptions. As such, a PPT simulator cannot simulate a PUF, that is, PUFs are non-programmable. Although non-programmable, PUFs are not modeled as global setup [8]. [7] shows how to achieve unconditional UC secure Oblivious Transfer, Bit Commitment and Key Agreement with trusted PUFs.

1.1 Our Contribution

We observe that the UC formulation of PUFs proposed by Brzuska et al. makes the following crucial assumption: the model considers *trusted* PUFs only, that is, adversaries are assumed to be unable to produce fake/malicious PUFs. As we argue below, we feel that assuming that an adversary cannot misbehave by creating fake/malicious PUFs, might be unrealistic in the real world. Given that the study of PUFs is still in its infancy, it is risky to rely on assumptions on the impossibility of the adversaries in generating and accessing PUFs adversarially. The main contribution of this work is to study security models that capture such plausible real-world attacks, and provide protocols that are secure in the presence of such adversaries.

Modeling malicious PUFs. We augment the UC framework so to enable the adversary to create untrusted (malicious) PUFs. But what exactly are malicious PUFs? In real life, an adversary could tamper with a PUF in such a way that the PUF loses any of its security properties. Or the adversary may introduce new behaviours; for example, the PUF may start logging its queries. To keep the treatment of malicious behaviour as general as possible, we allow the adversary to send as PUF any hardware token that meets the syntactical requirements of a PUF. Thus, an adversary is assumed to be able to even produce fake PUFs that might be stateful and programmed with malicious code. We assume that a malicious PUF however cannot interact with its creator once is sent away to another party. If this was not the case, then we are back in the standard model (see the Introduction in the full version [26]).

UC secure computation with malicious PUFs. The natural question is whether UC security can be achieved in such a much more hostile setting. We give a positive answer to this question by constructing a *computational* UC commitment scheme in the malicious PUFs model. Our commitment scheme needs two PUFs that are transferred only once (PUFs do not go back-and-forth), at the beginning of the protocol and it requires computational assumptions. We avoid that PUFs go back-and-forth by employing a technique that requires OT. The results of Canetti, et al. [11] shows how to achieve general UC computation from computational UC commitments. Whether *unconditional* UC secure computation is possible in the malicious PUF model, is still an open problem.

Hardness assumptions with PUFs. Notice that as correctly observed in [7], since PUFs are not PPT machines, it is not clear if standard complexity-theoretic assumptions still hold in presence of PUFs. We agree with this observation.

However the critical point is that even though there can exist a PUF that helps to break in polynomial time a standard complexity-theoretic assumptions, it is still unlikely that a PPT adversary can find such a PUF. Indeed a PPT machine can only generate a polynomial number of PUFs, therefore obtaining the one that allows to break complexity assumptions is an event that happens with negligible probability and thus it does not effect the concrete security of the protocols.

In light of the above discussion, only one of the following two cases is possible. 1) Standard complexity-theoretic assumptions still hold in presence of PPT adversaries that generate PUFs; in this case our construction is secure. 2) There exists a PPT adversary that can generate a PUF that breaks standard assumptions; in this case our construction is not secure, but the whole foundations of complexity-theoretic cryptography would fall down (which is quite unlikely to happen) with respect to real-world adversaries. We elaborate on this issue in Section 3.1.

Additional results. We now mention additional results that can be found in the full version of this paper [26] but have been omitted from the present conference version due to lack of space. Firstly, we further investigate the feasibility of achieving *unconditional security* in the malicious PUF model. We leave the important question of unconditional UC open, but provide a construction of an unconditional commitment scheme in the malicious PUF model. Secondly, we propose and study another modification to the original model of Brzuska et al. In the new model which we call "oblivious-query model", all parties (and the adversary) use trusted PUFs, but in the security proofs, the simulator is *not* allowed to observe the adversary's queries to its PUF. The main motivation for studying this modification is that the ability of the simulator to observe adversary's queries stems from the assumption that there is only a single, prescribed procedure for evaluating a PUF. As we discuss in detail in the full version, this assumption is not well-justified in the real world. Our main contribution in the oblivious-query model is the construction of an unconditional UC protocol for OT. Lastly, we show that if both adversarial modes discussed above are combined, viz., adversaries can create malicious PUFs and may query honest PUFs via non-prescribed processes, then UC security is impossible.

Independent work. Very recently and independently of us, van Dijk and Rührmair [36] also study the use of PUFs in cryptographic protocols. Among other things, they consider the "bad" PUF model where PUFs can be augmented with malicious behaviour like keeping a log of queries, etc. They show that unconditional OT is impossible using bad PUFs, but their setting is very different from ours. For a detailed discussion about the work of van Dijk and Rührmair, see the Introduction of the full version [26].

2 Definitions

Notation. We let n be the security parameter and PPT be the class of probabilistic polynomial time Turing machines. We use $v \xleftarrow{\$} A()$ when the algorithm A is randomized. We denote by $\mathsf{dis}_{\mathsf{ham}}(a, b)$ the Hamming distance of a and b.

Physically uncloneable functions. We follow definitions given in [7]. A PUF is a noisy physical source of randomness. The randomness property comes from an uncontrollable manufacturing process. A PUF is evaluated with a physical stimulus, called the *challenge*, and its physical output, called the *response*, is measured. Because the processes involved are physical, the function implemented by a PUF can not (necessarily) be modeled as a mathematical function, neither can be considered computable in PPT. Moreover, the output of a PUF is noisy, namely, querying a PUF twice with the same challenge, could yield to different outputs. The mathematical formalization of a PUF due to [7] is the following.

A PUF-family $\mathcal{P}$ is a pair of (not necessarily efficient) algorithms Sample and Eval, and is parameterized by the bound on the noise of PUF's response d_{noise} and the range of the PUF's output rg. Algorithm Sample abstracts the PUF fabrication process and works as follows. On input the security parameter, it outputs a PUF-index id from the PUF-family satisfying the security property (that we define soon) according to the security parameter. Algorithm Eval abstracts the PUF-evaluation process. On input a challenge q, it evaluates the PUF on q and outputs the response a of length rg. The output is guaranteed to have bounded noise d_{noise}, meaning that, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$. Wlog, we assume that the challenge space of a PUF is a full set of strings of a certain length.

Definition 1 (Physically Uncloneable Functions). *Let rg denote the size of the range of the PUF responses of a PUF-family and d_{noise} denote a bound of the PUF's noise. $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ is a family of (rg, d_{noise})-PUF if it satisfies the following properties.*

Index Sampling. *Let $\mathcal{I}_n$ be an index set. On input the security parameter n, the sampling algorithm Sample outputs an index $\mathsf{id} \in \mathcal{I}_n$ following a not necessarily efficient procedure. Each $\mathsf{id} \in \mathcal{I}_n$ corresponds to a set of distributions $\mathcal{D}_{\mathsf{id}}$. For each challenge $q \in \{0,1\}^n$, $\mathcal{D}_{\mathsf{id}}$ contains a distribution $\mathcal{D}_{\mathsf{id}}(q)$ on $\{0,1\}^{rg(n)}$. $\mathcal{D}_{\mathsf{id}}$ is not necessarily an efficiently sampleable distribution.*

Evaluation. *On input the tuple $(1^n, \mathsf{id}, q)$, where $q \in \{0,1\}^n$, the evaluation algorithm Eval outputs a response $a \in \{0,1\}^{rg(n)}$ according to distribution $\mathcal{D}_{\mathsf{id}}(q)$. It is not required that Eval is a PPT algorithm.*

Bounded Noise. *For all indexes $\mathsf{id} \in \mathcal{I}_n$, for all challenges $q \in \{0,1\}^n$, when running $\mathsf{Eval}(1^n, \mathsf{id}, q)$ twice, the Hamming distance of any two responses a_1, a_2 is smaller than $d_{\mathsf{noise}}(n)$.*

In the paper we use $\mathsf{PUF}_{\mathsf{id}}(q)$ to denote $\mathcal{D}_{\mathsf{id}}(q)$. When not misleading, we omit id from $\mathsf{PUF}_{\mathsf{id}}$, using only the notation PUF.

Security of PUFs. We assume that PUFs enjoy the properties of *uncloneability* and *unpredictability*. Unpredictability is modeled via an entropy condition on the PUF distribution. Namely, given that a PUF has been measured on a polynomial number of challenges, the response of the PUF evaluated on a new challenge has still a significant amount of entropy. Formally,

Definition 2 (Unpredictability). *A* (rg, d_{noise})*-PUF family* $\mathcal{P} = (\mathsf{Sample}, \mathsf{Eval})$ *for security parameter* n *is* $(d_{\mathsf{min}}(n), m(n))$*-unpredictable if for any* $q \in \{0,1\}^n$ *and challenge list* $\mathcal{Q} = (q_1, \ldots, q_{\mathtt{poly}(n)})$*, one has that, if for all* $1 \leq k \leq \mathtt{poly}(n)$ *the Hamming distance satisfies* $\mathsf{dis}_{\mathsf{ham}}(q, q_k) \geq d_{\mathsf{min}}(n)$*, then the average min-entropy satisfies* $\tilde{H}_\infty(\mathsf{PUF}(q)|\mathsf{PUF}(\mathcal{Q})) \geq m(n)$*, where* $\mathsf{PUF}(\mathcal{Q})$ *denotes a sequence of random variables* $\mathsf{PUF}(q_1), \ldots, \mathsf{PUF}(q_{\mathtt{poly}(n)})$ *each corresponding to an evaluation of the PUF on challenge* q_k*. Such a PUF-family is called a* $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$*-PUF family.*

Fuzzy extractors. Fuzzy extractors of Dodis et al. [13] are applied to the outputs of the PUF, to convert such noisy, high-entropy measurements into *reproducible* randomness. Very informally, a fuzzy extractor is a pair of efficient randomized algorithms $(\mathsf{FuzGen}, \mathsf{FuzRep})$. FuzGen takes as input an ℓ-bit string, that is the PUF's response a, and outputs a pair (p, st), where st is a uniformly distributed string, and p is a public helper data string. FuzRep takes as input the PUF's noisy response a' and the helper data p and generates the very same string st obtained with the original measurement a. The security property of fuzzy extractors guarantees that, if the min-entropy of the PUF's responses are greater than a certain parameter m, knowledge of the public data p only, without the measurement a, does not give any information on the secret value st. The correctness property, guarantees that, all pairs of responses a, a' that are close enough, i.e., their hamming distance is less then a certain parameter t, will be recovered by FuzRep to the same value st generated by FuzGen. In order to apply fuzzy extractors to PUF's answers, it is sufficient to pick an extractor whose parameters match with the parameter of the PUF being used.

3 UC Security with Malicious PUFs

In Section 1 we have motivated the need of a different formulation of UC security with PUFs that allows the adversary to generate malicious PUFs. In this section we first show how to model malicious PUFs in the UC framework, and then show that as long as standard computational assumptions still hold when PPT adversaries can generate (even malicious) PUFs, there exist protocols for UC realizing any functionality with (malicious) PUFs.

3.1 Modeling Malicious PUFs

We allow our adversaries to send malicious PUFs to honest parties[6]. As discussed before, the motivation for malicious PUFs is that the adversary may have some

[6] Throughout this section, we assume the reader is familiar with the original UC PUF formulation of Brzuska et al. [7] (Section 4.2).

control over the manufacturing process and may be able to produce errors in the process that break the PUF's security properties. Thus, we would like parties to rely only on the PUFs that they themselves manufacture (or obtain from a source that they trust), and not on the ones they receive from other (possibly adversarial) parties.

Malicious PUFs families. In the real world, an adversary may create a malicious PUF in a number of ways. For example, it can tamper with the manufacturing process for an honestly-generated PUF to compromise its security properties (unpredictability, for instance). It may also introduce additional behaviour into the PUF token, like logging of queries. Taking inspiration from the literature on modeling tamper-proof hardware tokens, one might be tempted to model malicious PUFs analogously in the following way: to create a malicious PUF, the adversary simply specifies to the ideal functionality, the (malicious) code it wants to be executed instead of an honest PUF. Allowing the adversary to specify the malicious code enables the simulator to "rewind" the malicious PUF, which is used crucially in security proofs in the hardware token model. However, modeling malicious PUFs in this way would disallow the adversary from modifying honest PUFs (or more precisely, the honest PUF manufacturing process). To keep our treatment as general as possible, we do not place any restriction on a malicious PUF, except that it should have the same syntax as that of an honest PUF family, as specified in Definition 1. In particular, the adversary is *not* required to know the code of malicious PUFs it creates, and thus our simulator can not rely on rewinding in the security proofs. Formally, we allow the adversary to specify a "malicious PUF family", that the ideal functionality uses. Of course, in the protocol, we also want the honest parties to be able to obtain and send honestly generated PUFs. Thus our ideal functionality for PUFs, $\mathcal{F}_{\mathsf{PUF}}$ (Fig. 1) is parameterized by two PUF families: the normal (or honest) family $(\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ and the possibly malicious family $(\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$. When a party P_i wants to initialize a PUF, it sends a $\mathsf{init}_{\mathsf{PUF}}$ message to $\mathcal{F}_{\mathsf{PUF}}$ in which it specifies the $\mathtt{mode} \in \{\, \mathtt{normal}, \mathtt{mal} \,\}$, and the ideal functionality uses the corresponding family for initializing the PUF. For each initialized PUF, the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ also stores a tag representing the family (i.e., $\mathtt{mal}$ or $\mathtt{normal}$) from which it was initialized. Thus, when the PUF needs to be evaluated, $\mathcal{F}_{\mathsf{PUF}}$ runs the evaluation algorithm corresponding to the tag.

As in the original formulation of Brzuska et al., the ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ keeps a list $\mathcal{L}$ of tuples $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \hat{P}, \tau)$. Here, sid is the session identifier of the protocol and id is the PUF identifier output by the $\mathsf{Sample}_{\mathtt{mode}}$ algorithm. As discussed above $\mathtt{mode} \in \{\, \mathtt{normal}, \mathtt{mal} \,\}$ indicates the mode of the PUF, and $\hat{P}$ identifies the party that currently holds the PUF. The final argument τ specifies transition of PUFs: $\tau = \mathsf{notrans}$ indicates the PUF is not in transition, while $\tau = \mathsf{trans}(P_j)$ indicates that the PUF is in transition to party P_j. Only the adversary may query the PUF during the transition period. Thus, when a party P_i hands over a PUF to party P_j, the corresponding τ value for that PUF is changed from $\mathsf{notrans}$ to $\mathsf{trans}(P_j)$, and the adversary is allowed to send evaluation queries to this PUF. When the adversary is done with querying the PUF, it sends a $\mathsf{ready}_{\mathsf{PUF}}$

message to the ideal functionality, which hands over the PUF to P_j and changes the PUFs transit flag back to notrans. The party P_j may now query the PUF. The ideal functionality now waits for a $\mathsf{received}_{\mathsf{PUF}}$ message from the adversary, at which point it sends a $\mathsf{received}_{\mathsf{PUF}}$ message to P_i informing it that the hand over is complete. The ideal functionality is described formally in Fig. 1.

Allowing adversary to create PUFs. We deviate from the original formulation of $\mathcal{F}_{\mathsf{PUF}}$ of Brzuska et al. [7] in one crucial way: we allow the ideal-world adversary $\mathcal{S}$ to create new PUFs. That is, $\mathcal{S}$ can send a $\mathsf{init}_{\mathsf{PUF}}$ message to $\mathcal{F}_{\mathsf{PUF}}$. In the original formulation of Brzuska et al., $\mathcal{S}$ could not create its own PUFs, and this has serious implications for the composition theorem. We thank Margarita Vald [35] for pointing out this issue. We elaborate on this in Appendix H in the full version [26]. Also, it should be noted that the PUF set-up is non-programmable, but not *global* [8]. The environment must go via the adversary to query PUFs, and may only query PUFs in transit or held by the adversary at that time.

We remark that the OT protocol of [7] for honest PUFs, fails in the presence of malicious PUFs. Consider the OT protocol in Fig. 3 in [7]. The security crucially relies on the fact that the receiver P_j *can not* query the PUF after receiving sender's first message, i.e., the pair (x_0, x_1). If it could do so, then it would query the PUF on both $x_0 \oplus v$ and $x_1 \oplus v$ and learn both s_0 and s_1. In the malicious PUF model however, as there is no guarantee that the receiver can not learn query/answer pairs when a malicious PUF that he created is not in its hands, the protocol no longer remains secure.

PUFs and computational assumptions. The protocol we present in the next section will use computational hardness assumptions. These assumptions hold against probabilistic polynomial-time adversaries. However, PUFs use physical components and are not modeled as PPT machines, and thus, the computational assumptions must additionally be secure against PPT adversaries that have access to PUFs. We remark that this is a reasonable assumption to make, as if this is not the case, then PUFs can be used to invert one-way functions, to find collisions in CRHFs and so on, therefore not only our protocol, but any computational-complexity based protocol would be insecure. Note that PUFs are physical devices that actually exist in the real world, and thus all real-world adversaries could use them.

To formalize this, we define the notion of "admissible" PUF families. A PUF family (regardless of whether it is honest or malicious) is called *admissible* with respect to a hardness assumption if that assumption holds even when the adversary has access to PUFs from this family. We will prove that our protocol is secure when the $\mathcal{F}_{\mathsf{PUF}}$ ideal functionality is instantiated with admissible PUF families. In particular, all the cryptographic tools that we use to construct our protocol can be based on the DDH assumption. From this point on in this paper, a "PUF family" would be taken to mean a PUF family which is admissible with respect to DDH.

$\mathcal{F}_{\mathsf{PUF}}$ uses PUF families $\mathcal{P}_1 = (\mathsf{Sample}_{\mathtt{normal}}, \mathsf{Eval}_{\mathtt{normal}})$ with parameters $(rg, d_{\mathsf{noise}}, d_{\mathsf{min}}, m)$, and $\mathcal{P}_2 = (\mathsf{Sample}_{\mathtt{mal}}, \mathsf{Eval}_{\mathtt{mal}})$.
It runs on input the security parameter 1^n, with parties $\mathbb{P} = \{ P_1, \ldots, P_n \}$ and adversary $\mathcal{S}$.

- When a party $\hat{P} \in \mathbb{P} \cup \{ \mathcal{S} \}$ writes $(\mathsf{init}_{\mathsf{PUF}}, \mathsf{sid}, \mathtt{mode}, \hat{P})$ on the input tape of $\mathcal{F}_{\mathsf{PUF}}$, where $\mathtt{mode} \in \{ \mathtt{normal}, \mathtt{mal} \}$, then $\mathcal{F}_{\mathsf{PUF}}$ checks whether $\mathcal{L}$ already contains a tuple $(\mathsf{sid}, *, *, *, *)$:
  - If this is the case, then turn into the waiting state.
  - Else, draw $\mathsf{id} \leftarrow \mathsf{Sample}_{\mathtt{mode}}(1^n)$ from the PUF family. Put $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \hat{P}, \mathsf{notrans})$ in $\mathcal{L}$ and write $(\mathsf{initialized}_{\mathsf{PUF}}, \mathsf{sid})$ on the communication tape of $\hat{P}$.
- When party $P_i \in \mathbb{P}$ writes $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, P_i, q)$ on $\mathcal{F}_{\mathsf{PUF}}$'s input tape, check if there exists a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, q)$. Write $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ on P_i's communication input tape.
- When a party P_i sends $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if there exists a tuple $(\mathsf{sid}, *, *, P_i, \mathsf{notrans})$ in $\mathcal{L}$.
  - If not, then turn into waiting state.
  - Else, modify the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_i, \mathsf{notrans})$ to the updated tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$. Write $(\mathsf{invoke}_{\mathsf{PUF}}, \mathsf{sid}, P_i, P_j)$ on $\mathcal{S}$'s communication input tape.
- When the adversary sends $(\mathsf{eval}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S}, q)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains a tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(*))$ or $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \mathcal{S}, \mathsf{notrans})$.
  - If not, then turn into waiting state.
  - Else, run $a \leftarrow \mathsf{Eval}_{\mathtt{mode}}(1^n, \mathsf{id}, q)$ and return $(\mathsf{response}_{\mathsf{PUF}}, \mathsf{sid}, q, a)$ to $\mathcal{S}$.
- When $\mathcal{S}$ sends $(\mathsf{ready}_{\mathsf{PUF}}, \mathsf{sid}, \mathcal{S})$ to $\mathcal{F}_{\mathsf{PUF}}$, check if $\mathcal{L}$ contains the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$.
  - If not found, turn into the waiting state.
  - Else, change the tuple $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, \perp, \mathsf{trans}(P_j))$ to $(\mathsf{sid}, \mathsf{id}, \mathtt{mode}, P_j, \mathsf{notrans})$ and write $(\mathsf{handover}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ on P_j's communication input tape and store the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$.
- When the adversary sends $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ to $\mathcal{F}_{\mathsf{PUF}}$, check if the tuple $(\mathsf{received}_{\mathsf{PUF}}, \mathsf{sid}, P_i)$ has been stored. If not, return to the waiting state. Else, write this tuple to the input tape of P_i.

Fig. 1. The ideal functionality $\mathcal{F}_{\mathsf{PUF}}$ for malicious PUFs.

3.2 Constructions for UC Security in the Malicious PUFs Model

In this section we present a construction for UC-secure commitment scheme in the malicious PUFs model, which yields UC-security for any PPT functionality via the [11] compiler.

We first recall some of the peculiarities of the PUFs model. A major difficulty when using PUFs, in contrast to say tamper-proof tokens, is that PUFs are *not programmable*. That is, the simulator can not simulate the answer of a PUF, and must honestly forward the queries to the $\mathcal{F}_{\mathsf{PUF}}$ functionality. The only power of the simulator is to intercept the queries made by the adversary to honest PUFs. Thus, in designing the protocol, we shall force parties to query the PUFs with the critical private information related to the protocol, therefore allowing the simulator to extract such information in straight-line. In the malicious PUFs model the behaviour of a PUF created and sent by an adversary is entirely in the adversary's control. A malicious PUF can answer (or even abort) adaptively on the query according to some pre-shared strategy with the malicious creator. Finally, a side effect of the unpredictability of PUFs, is that the creator of a honest PUF is not able to check the authenticity of the answer generated by its own PUF, without having the PUF in its hands (or having queried the PUF previously on the very same value).

Techniques and proof intuition. Showing UC security for commitments requires obtaining straight-line extraction against a malicious sender and straight-line equivocality against a malicious receiver. Our starting point is the equivocal commitment scheme of [12] which builds upon Naor's scheme [25]. Naor's scheme consists of two messages, where the first message is a randomly chosen string r that the receiver sends to the sender. The second message is the commitment of the bit b, computed using r. More precisely, to commit to bit b, the second message is $G(s) \oplus (r \wedge b^{|r|})$, where $G()$ is a PRG, and s a randomly chosen seed. The scheme has the property that if the string r is crafted appropriately, then the commitment is equivocal. [12] shows how this can be achieved by adding a coin-tossing phase before the commitment. The coin tossing of [12] proceeds as follows: the receiver commits to a random string α (using a statistically hiding commitment scheme), the sender sends a string β, and then the receiver opens the commitment. Naor's parameter r is then set as $\alpha \oplus \beta$.

Observe that if the simulator can choose β after knowing α, then it can control the output of the coin-tossing phase, and therefore equivocate the commitment. Thus, to achieve equivocality against a malicious receiver, the simulator must be able to extract α from the commitment. Similarly, when playing against a malicious sender, the simulator should be able to extract the value committed in the second message of Naor's commitment.

Therefore, to construct a UC-secure commitment, we need to design an extractable commitment scheme for both directions. The extractable commitment of α that we construct for the receiver, must be statistically-hiding (this is necessary to prove binding). We denote such commitment as $\mathsf{Com}_{\mathsf{sh}} = (\mathsf{S}_{\mathsf{sh}}, \mathsf{R}_{\mathsf{sh}})$. On the other hand, the commitment sent by the sender, must be extractable and allow for equivocation. We denote such commitment as $\mathsf{Com}_{\mathsf{eq}} = (\mathsf{S}_{\mathsf{eq}}, \mathsf{R}_{\mathsf{eq}})$. As we shall see soon, the two schemes require different techniques as they aim to different properties. However, they both share the following structure to achieve extractability.

The receiver creates a PUF and queries it with two randomly chosen challenges (q_0, q_1), obtaining the respective answers (a_0, a_1). The PUF is then sent to the sender. To commit to a bit b, the sender first needs to obtain the value q_b. This is done by running an OT protocol with the receiver. Then the sender queries the PUF with q_b and commits to the response a_b. Note that the sender does not commit to the bit directly, but to the *answer* of the PUF. This ensures extractability. To decommit to b, the sender simply opens the commitment of the PUF-answer sent before. Note that the receiver can check the authenticity of the PUF-answer without having its own PUF back. The simulator can extract the bit by intercepting the queries sent to the PUF and taking the one that is close enough, in Hamming distance, to either q_0 or q_1. Due to the security of OT, the sender can not get both queries (thus confusing the simulator), neither can the receiver detect which query has been transferred. Due to the binding property of the commitment scheme used to commit q_b, a malicious sender cannot postpone querying the PUF to the decommitment phase (thus preventing the simulator to extract already in the commitment phase). Due to the unpredictability of PUFs, the sender cannot avoid to query the PUF to obtain the correct response.

This protocol achieves extractability. To additionally achieve statistically hiding and equivocality, protocol $\mathsf{Com}_{\mathsf{sh}}$ and $\mathsf{Com}_{\mathsf{eq}}$ develop on this basic structure in different ways accordingly to the different properties that they achieve. The main difference is in the commitment of the answer a_b.

In Protocol $\mathsf{Com}_{\mathsf{sh}}$, S_{sh} commits to the PUF-response a_b using a statistically hiding commitment scheme. Additionally, S_{sh} provides a statistical zero-knowledge argument of knowledge of the message committed. This turns out to be necessary to argue about binding (that is only computational). Finally, the OT protocol executed to exchange q_0, q_1 must be statistically secure for the OT receiver. The formal description of protocol $\mathsf{Com}_{\mathsf{sh}}$ is provided in Fig. 2.

In Protocol $\mathsf{Com}_{\mathsf{eq}}$ the answer a_b is committed following Naor's commitment scheme. The input of S_{eq} is the Naor's parameter decided in the coin-flipping phase, and is the vector $\bar{r}$ of strings $r_1, \ldots, r_l$ (a_b is a l-bit string, where l is the range of the PUF). Earlier we said that the simulator can properly craft $\bar{r}$, so that it will be able equivocate the commitment of a_b. However, due to the structure of the extractable commitment shown above, being able to equivocate the commitment of a_b is not enough anymore. Indeed, in the protocol above, due to the OT protocol, the simulator will be able to obtain only one of the PUF-queries among (q_0, q_1), and it must choose the query q_b already in the commitment phase (when the secret bit b is not known to the simulator). Thus, even though the simulator has the power to equivocate the commitment to any string, it might not know the correct PUF-answer to open to. We solve this problem by asking the receiver to reveal both values (q_0, q_1) played in the OT protocol (along with the randomness used in the OT protocol), obviously only after S_{eq} has committed to the PUF-answer. Now, the simulator can: play the OT protocol with a random bit, commit to a random string (without querying the PUF), and then obtain both queries (q_0, q_1). In the decommitment phase, the simulator gets the actual bit b. Hence, it can query the PUF with input

q_b, obtain the PUF-answer, and equivocate the commitment so to open to such PUF-answer. There is a subtle issue here and is the possibility of *selective abort* of a malicious PUF. If the PUF aborts when queried with a particular string, then we have that the sender would abort already in the commitment phase, while the simulator aborts only in the decommitment phase. We avoid such problem by requiring that the sender continues the commitment phase by committing to a random string in case the PUF aborts. The above protocol is statistically binding (we are using Naor's commitment), straight-line extractable, and assuming that Naor's parameter was previously ad-hoc crafted, it is also straight-line equivocal. To commit to a bit we are committing to the l-bit PUF-answer, thus the size of Naor's parameter $\bar{r}$, is $N = (3n)l$. Protocol $\mathsf{Com}_{\mathsf{eq}}$ is formally described in Fig. 3.

The final UC-secure commitment scheme $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{S}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ consists of the coin-flipping phase, and the (equivocal) commitment phase. In the coin flipping, the receiver commits to α using the statistically hiding straight-line extractable commitment scheme $\mathsf{Com}_{\mathsf{sh}}$. The output of the coin-flipping is the Naor's parameter $\bar{r}{=}\alpha \oplus \beta$ used as common input for the extractable/equivocal commitment scheme $\mathsf{Com}_{\mathsf{eq}}$. Protocol $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{S}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ is formally described in Fig. 4.

Both protocol $\mathsf{Com}_{\mathsf{sh}}, \mathsf{Com}_{\mathsf{eq}}$ require one PUF sent from the receiver to the sender. We remark that PUFs are transferred only *once at the beginning of the protocol*. We finally stress that we do not assume authenticated PUF delivery. Namely, the privacy of the honest party is preserved even if the adversary interferes with the delivery process of the honest PUFs (e.g., by replacing the honest PUF).

Theorem 1. *If* $\mathsf{Com}_{\mathsf{sh}} = (\mathsf{S}_{\mathsf{sh}}, \mathsf{R}_{\mathsf{sh}})$ *is a statistically hiding straight-line extractable commitment scheme in the malicious PUFs model, and* $\mathsf{Com}_{\mathsf{eq}} = (\mathsf{S}_{\mathsf{eq}}, \mathsf{R}_{\mathsf{eq}})$ *is a statistically binding straight-line extractable and equivocal commitment scheme in the malicious PUFs model, then* $\mathsf{Com}_{\mathsf{uc}} = (\mathsf{S}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$ *in Fig. 4, is a UC-secure commitment scheme in the malicious PUFs model.*

The above protocol can be used to implement the multiple commitment functionality $\mathcal{F}_{\mathsf{mcom}}$ by using independent PUFs for each commitment. Note that in our construction we can not reuse the *same* PUF when multiple commitments are executed *concurrently*[7]. The reason is that, in both sub-protocols $\mathsf{Com}_{\mathsf{sh}}, \mathsf{Com}_{\mathsf{eq}}$, in the opening phase the sender forwards the answer obtained by querying the receiver's PUF. The answer of a malicious PUF can then convey information about the value committed in concurrent sessions that have not been opened yet.

When implementing $\mathcal{F}_{\mathsf{mcom}}$ one should also deal with malleability issues. In particular, one should handle the case in which the man-in-the-middle adversary forwards honest PUFs to another party. However such attack can be ruled out by exploiting the unpredictability of honest PUFs as follows. Let P_i be the creator of PUF_i, running an execution of the protocol with P_j. Before delivering its own

[7] Note that however our protocol enjoys parallel composition and reuse of the same PUF, i.e., one can commit to a string reusing the same PUF.

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

Commitment Phase

R_{sh} : (Initialization and PUF exchange)

1. Create $\mathsf{PUF}_{\mathsf{R}}$; obtain $a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$, $a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$, for $(q_0, q_1) \xleftarrow{\$} \{0,1\}^n$.
2. $(st_0, p_0) \leftarrow \mathsf{FuzGen}(a_0)$, $(st_1, p_1) \leftarrow \mathsf{FuzGen}(a_1)$.
3. Handover $\mathsf{PUF}_{\mathsf{R}}$ to S_{sh}.

$\mathsf{R}_{\mathsf{sh}} \Leftrightarrow \mathsf{S}_{\mathsf{sh}}$: (Statistical OT phase)
R_{sh} runs as the OT Sender with input (q_0, q_1), and S_{sh} runs as the OT Receiver with input b. Let q'_b be the local output of S_{sh}.

$\mathsf{S}_{\mathsf{sh}} \Leftrightarrow \mathsf{R}_{\mathsf{sh}}$: (Statistically Hiding Commitment)
S_{sh} queries $\mathsf{PUF}_{\mathsf{R}}$ on input q'_b and obtains a'_b. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, set $a'_b \xleftarrow{\$} \{0,1\}^l$. S_{sh} commits to a'_b using a statistically-hiding commitment scheme. Let c be the transcript of the commitment phase.

$\mathsf{S}_{\mathsf{sh}} \Leftrightarrow \mathsf{R}_{\mathsf{sh}}$: (Proof of knowledge of the Decommitment)
S_{sh} proves that he knows the decommitment of c running a statistical ZK Argument of Knowledge protocol. If the proof is not accepting, R_{sh} aborts.

Decommitment Phase

S_{sh} : if $\mathsf{PUF}_{\mathsf{R}}$ did not abort, send opening to a'_b to R_{sh}.

R_{sh} : if the opening for a'_b is accepting and $\mathsf{FuzRep}(a'_b, p_b) = st_b$ then accept b. Otherwise reject.

Fig. 2. Statistically Hiding Straight-Line Extractable Bit Commitment $\mathsf{Com}_{\mathsf{sh}}$.

PUF, P_i queries it with the identity of P_j concatenated with a random *nonce*. Then, at some point during the protocol execution with P_j it will ask P_j to evaluate PUF_i on such *nonce* (and the identity). Due to the unpredictability of PUFs, and the fact that *nonce* is a randomly chosen value, P_j is able to answer to such a query only if it *possesses* the PUF. The final step to obtain UC security for any functionality consists in using the compiler of [11], which only needs a UC secure implementation of the $\mathcal{F}_{\mathsf{mcom}}$ functionality.

4 Conclusion

We introduce the Malicious PUF model which models the very realistic attack of an adversary replacing a proper PUF with a "PUF-looking" device that implements an arbitrary malicious functionality. We show that in this model is possible to achieve UC-security relying on complexity-theoretic assumptions, by providing an implementation of UC-secure commitment scheme.

Acknowledgments

Research supported in part by NSF grants CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; US-Israel BSF grant 2008411, OKAWA

Sender's Input: Bit $b \in \{0,1\}$.
Common Input: $\bar{r} = (r_1, \ldots, r_l)$.

Commitment Phase

R_{eq} : (Initialization and PUF exchange)

1. Create $\mathsf{PUF}_{\mathsf{R}}$. Obtain $a_0 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_0)$, $a_1 \leftarrow \mathsf{PUF}_{\mathsf{R}}(q_1)$, for $(q_0, q_1) \xleftarrow{\$} \{0,1\}^n$.
2. $(st_0, p_0) \leftarrow \mathsf{FuzGen}(a_0)$, $(st_1, p_1) \leftarrow \mathsf{FuzGen}(a_1)$.
3. Handover $\mathsf{PUF}_{\mathsf{R}}$ to S_{eq};
4. Choose random tape $\mathsf{ran}_{\mathsf{OT}} \xleftarrow{\$} \{0,1\}^*$.

$\mathsf{R}_{\mathsf{eq}} \Leftrightarrow \mathsf{S}_{\mathsf{eq}}$: (Statistical OT phase)
R_{eq} runs as the OT Sender with input (q_0, q_1) using randomness $\mathsf{ran}_{\mathsf{OT}}$, and S_{eq} runs as the OT Receiver with input b. Let q'_b be the local output of S_{eq}. Let τ_{OT} be the transcript of the execution of the OT protocol.

S_{eq}: (Statistically Binding Commitment)

1. $a'_b \leftarrow \mathsf{PUF}_{\mathsf{R}}(q'_b)$. If $\mathsf{PUF}_{\mathsf{R}}$ aborts, $a'_b \xleftarrow{\$} \{0,1\}^l$.
2. for $1 \le i \le l$, pick $s_i \xleftarrow{\$} \{0,1\}^n$, $c_i = G(s_i) \oplus (r_i \wedge a'_b[i])$. [a]
3. send $c_1, \ldots, c_l$ to R_{eq}.

R_{eq}: upon receiving $c_1, \ldots, c_l$, send $\mathsf{ran}_{\mathsf{OT}}, q_0, q_1$ to S_{eq}.
S_{eq}: check if transcript τ_{OT} is consistent with $(\mathsf{ran}_{\mathsf{OT}}, q_0, q_1, b)$. If the check fails abort.

Decommitment Phase

S_{eq} : if $\mathsf{PUF}_{\mathsf{R}}$ did not abort, send $((s_1, \ldots, s_l), a'_b), b$ to R_{sh}.
R_{eq} : if for all i, it holds that $(c_i = G(s_i) \oplus (r_i \wedge a'_b[i])) \wedge (\mathsf{FuzRep}(a'_b, p_b) = st_b)$ then accept. Else reject.

[a] where $(r_i \wedge a'_b[i])[j] = r_i[j] \wedge a'_b[i]$.

Fig. 3. Statistically Binding Straight-line Extractable/Equivocal Commitment $\mathsf{Com}_{\mathsf{eq}}$.

Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Foundation Award, Teradata Research Award, Lockheed-Martin Corporation Research Award and the European Commission through the FP7 programme under contract 216676 ECRYPT II. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

The authors thank Margarita Vald for pointing out the problem with the original formulation of the [7] PUF ideal functionality where the adversary is not allowed to create PUFs. The authors also thank Marten van Dijk and Ulrich Rührmair for extensive discussions on their and our paper.

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

Commitment Phase

$\mathsf{R}_{\mathsf{uc}} \Leftrightarrow \mathsf{S}_{\mathsf{uc}}$: (Coin Flipping)

1. R_{uc} picks $\alpha \xleftarrow{\$} \{0,1\}^N$; commit to α by running as sender in protocol $\mathsf{Com}_{\mathsf{sh}}$ while S_{uc} runs as receiver.
2. S_{uc} sends $\beta \xleftarrow{\$} \{0,1\}^N$ to R_{uc}.
3. R_{uc} sends decommitment for α to S_{uc}.
4. S_{uc}: if the decommitment is not accepting, abort. Else R_{uc} and S_{uc} set $\bar{r} = \alpha \oplus \beta$.

$\mathsf{S}_{\mathsf{uc}} \Leftrightarrow \mathsf{R}_{\mathsf{uc}}$: (Equivocal Commitment)

S_{uc} commit to b by running as sender in $\mathsf{Com}_{\mathsf{eq}}$ where R_{uc} runs as receiver, and the common input is $\bar{r}$.

Decommitment Phase

S_{uc} runs the decommitment phase of $\mathsf{Com}_{\mathsf{eq}}$ to open b.
R_{uc} accepts if the decommitment of b is accepting.

Fig. 4. Computational UC Commitment Scheme $(\mathsf{S}_{\mathsf{uc}}, \mathsf{R}_{\mathsf{uc}})$.

References

1. Shweta Agrawal, Vipul Goyal, Abhishek Jain, Manoj Prabhakaran, and Amit Sahai. New impossibility results for concurrent composition and a non-interactive completeness theorem for secure computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 443–460. Springer, 2012.
2. Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Francois-Xavier Standaert, and Christian Wachsmann. A formalization of the security features of physical functions. In *IEEE Symposium on Security and Privacy*, pages 397–412. IEEE Computer Society, 2011.
3. Frederik Armknecht, Roel Maes, Ahmad-Reza Sadeghi, Berk Sunar, and Pim Tuyls. Memory leakage-resilient encryption based on physically unclonable functions. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 685–702. Springer, 2009.
4. Boaz Barak, Ron Canetti, Jesper B. Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *Foundations of Computer Science (FOCS'04)*, pages 394–403, 2004.
5. Boaz Barak, Manoj Prabhakaran, and Amit Sahai. Concurrent non-malleable zero knowledge. In *FOCS*, pages 345–354, 2006.
6. Christian Boit, Clemens Helfmeier, Dimitry Nedospasaov, and Jean-Pierre Seifert. Private Communication, 2013.
7. Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70. Springer, 2011.

8. Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 61–85. Springer, 2007.
9. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 2001.
10. Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 68–86, Warsaw, Poland, May 4–8, 2003. Springer, Berlin, Germany.
11. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, Lecture Notes in Computer Science, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.
12. Giovanni Di Crescenzo and Rafail Ostrovsky. On concurrent zero-knowledge with pre-processing. In *CRYPTO*, pages 485–502, 1999.
13. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.
14. Ilze Eichhorn, Patrick Koeberl, and Vincent van der Leest. Logically reconfigurable pufs: memory-based secure key storage. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, STC '11, pages 59–64, New York, NY, USA, 2011. ACM.
15. Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, and Ivan Visconti. Secure set intersection with untrusted hardware tokens. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011.
16. Sanjam Garg, Abishek Kumarasubramanian, Rafail Ostrovsky, and Ivan Visconti. Impossibility results for static input secure computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 424–442. Springer, 2012.
17. Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 308–326, Zurich, Switzerland, February 9–11, 2010. Springer, Berlin, Germany.
18. Jorge Guajardo, Sandeep S. Kumar, Geert Jan Schrijen, and Pim Tuyls. Fpga intrinsic pufs and their use for ip protection. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2007.
19. Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In *37th Annual ACM Symposium on Theory of Computing*, pages 644–653, 2005.
20. Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007.

21. Stefan Katzenbeisser, Ünal Koccabas, Vladimir Rozic, Ahmad-Reza Sadeghi, Ingrid Verbauwhede, and Christian Wachsmann. Pufs: Myth, fact or busted? a security evaluation of physically unclonable functions (pufs) cast in silicon. In Prouff and Schaumont [30], pages 283–301.
22. Ünal Koccabas, Ahmad-Reza Sadeghi, Christian Wachsmann, and Steffen Schulz. Poster: practical embedded remote attestation using physically unclonable functions. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 797–800. ACM, 2011.
23. Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. Pufky: A fully functional puf-based cryptographic key generator. In Prouff and Schaumont [30], pages 302–319.
24. Roel Maes and Ingrid Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 3–37. Springer Berlin Heidelberg, 2010.
25. Moni Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, pages 128–136, 1989.
26. Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. Cryptology ePrint Archive, Report 2012/143, 2012. `http://eprint.iacr.org/2012/143/`.
27. Ravikanth S. Pappu, Ben Recht, Jason Taylor, and Niel Gershenfeld. Physical one-way functions. *Science*, 297:2026–2030, 2002.
28. Ravikanth Srinivasa Pappu. *Physical One-Way Functions*. PhD thesis, MIT, 2001.
29. Manoj Prabhakaran and Amit Sahai. New notions of security: achieving universal composability without trusted setup. In *36th Annual ACM Symposium on Theory of Computing*, pages 242–251, 2004.
30. Emmanuel Prouff and Patrick Schaumont, editors. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*. Springer, 2012.
31. Ulrich Rührmair. Oblivious transfer based on physical unclonable functions. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *TRUST*, volume 6101 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 2010.
32. Ulrich Rührmair, Stefan Katzenbeisser, and H. Busch. Strong pufs: Models, constructions and security proofs. In A. Sadeghi and P. Tuyls, editors, *Towards Hardware Intrinsic Security: Foundations and Practice*, pages 79–96. Springer, 2010.
33. Ahmad-Reza Sadeghi, Ivan Visconti, and Christian Wachsmann. Enhancing rfid security and privacy by physically unclonable functions. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 281–305. Springer Berlin Heidelberg, 2010.
34. Pim Tuyls and Lejla Batina. Rfid-tags for anti-counterfeiting. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 115–131. Springer, 2006.
35. Margarita Vald. Private Communication, 2012.
36. Marten van Dijk and Ulrich Rührmair. Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results. *IACR Cryptology ePrint Archive*, 2012:228, 2012.