Erasable PUFs: Formal Treatment and Generic Design

Chenglu Jin
New York University
Brooklyn, New York, USA
chenglu.jin@nyu.edu

Wayne Burleson
University of Massachusetts Amherst
Amherst, Massachusetts, USA
burleson@umass.edu

Marten van Dijk
CWI Amsterdam & University of Connecticut
Amsterdam, Netherlands
marten.van.dijk@cwi.nl

Ulrich Rührmair
LMU München & University of Connecticut
Munich, Germany
ruehrmair@ilo.de

ABSTRACT

Physical Unclonable Functions (PUFs) have not only been suggested as new key storage mechanism, but — in the form of so-called *"Strong PUFs"* — also as cryptographic primitives in advanced schemes, including key exchange, oblivious transfer, or secure multi-party computation. This notably extends their application spectrum, and has led to a sequence of publications at leading venues such as IEEE S&P, CRYPTO, and EUROCRYPT in the past [3, 6, 10, 11, 29, 41]. However, one important unresolved problem is that adversaries can break the security of *all* these advanced protocols if they gain physical access to the employed Strong PUFs *after* protocol completion [41]. It has been formally proven [49] that this issue cannot be overcome by techniques on the protocol side alone, but requires resolution on the hardware level — the only fully effective known countermeasure being so-called *Erasable PUFs* [36]. Building on this work, this paper is the first to describe a generic method how any given silicon Strong PUF with digital CRP-interface can be turned into an Erasable PUF. We describe how the Strong PUF can be surrounded with a trusted control logic that allows the blocking (or "erasure") of single CRPs. We implement our approach, which we call *"GeniePUF"*, on FPGA, reporting detailed performance data and practicality figures. Furthermore, we develop the first comprehensive definitional framework for Erasable PUFs. Our work so re-establishes the effective usability of Strong PUFs in advanced cryptographic applications, and in the realistic case adversaries get access to the Strong PUF after protocol completion.

CCS CONCEPTS

• **Hardware** → **Integrated circuits**; • **Security and privacy** → **Embedded systems security**.

KEYWORDS

Physical Unclonable Functions (PUFs), PUF Re-Use Model, Erasable PUFs, Reconfigurable PUFs, GeniePUFs

ASHES'20, November 13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8090-4/20/11...$15.00
https://doi.org/10.1145/3411504.3421215

ACM Reference Format:
Chenglu Jin, Wayne Burleson, Marten van Dijk, and Ulrich Rührmair. 2020. Erasable PUFs: Formal Treatment and Generic Design. In *4th Workshop on Attacks and Solutions in Hardware Security (ASHES'20), November 13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3411504.3421215

1 INTRODUCTION AND OVERVIEW

1.1 PUFs and Their Differing Applications

The last years have witnessed an increasing interest in directly leveraging the physical properties of computer hardware for cryptographic and security purposes. One prime example are so-called Physical Unclonable Functions (PUFs), which exploit the natural manufacturing variations arising in most modern hardware systems [14, 24, 30].

In their early days, PUFs were predominantly seen as a novel tool for physical key storage and system identification. To start with, so-called Weak PUFs [35] (for example SRAM PUFs [17], transistor-based PUFs [24], diode-based PUFs [19], or DRAM PUFs [52]) were suggested as source of system-specific keys in hardware that had no non-volatile memory (NVM) on board. This successively established Weak PUFs as viable, arguably more secure alternative to classical NVMs in key storage applications [18, 22, 25, 26, 43].

In addition, so-called Strong PUFs [35] (such as Arbiter PUFs [45] and optical PUFs [30]), were proposed for new types of remote identification protocols [30]. In these schemes, a randomly chosen subset of PUF Challenge Response Pairs (CRPs) is directly sent in the clear and unencrypted in each protocol run, proving the identity of the PUF-holder to a remote party. Fresh CRPs have to be employed in each execution, necessitating a large CRP-space of the underlying Strong PUF. Interestingly, the use of Strong PUFs here leads to new hardware *and* new cryptographic protocols. In both cases, Weak and Strong PUFs empower electronic systems to identify themselves *without* having classical, permanent digital keys on board. This notable fact has created sustainable research interest in the security community ever since.

In recent years, an important second research strand on PUFs has evolved, however, which reaches strictly beyond the above key storage or identification scenarios. In this second avenue, *solely* Strong PUFs [35] [1] are employed as *"cryptographic primitive"* in

[1] Weak PUFs [35] are not suited for the application as cryptographic primitive in advanced protocols in the above sense: This scenario inevitably requires a large, inexhaustible CRP space with many possible challenges, numerically unpredictable responses, and a publicly accessible CRP-interface of the PUF, where every protocol

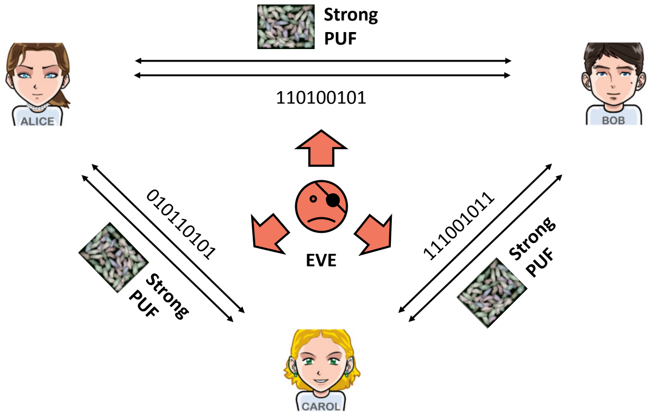

Figure 1: The most general setting for the application of Strong PUFs in cryptographic protocols: All participants are connected pairwise via a binary channel, and via a separate physical channel over which physical objects like Strong PUFs can be sent. Eve can access both the physical and binary channels alike.

advanced schemes like key exchange (KE), bit commitment (BC), oblivious transfer (OT), or secure multi-party computation (SMC) [6, 29, 32]. It turned out that all these tasks can solely be built on Strong PUFs and their features, without employing any additional security assumptions. This makes Strong PUFs a novel, independent fundament for cryptography: Their security is based on their assumed physical unclonability and numeric unpredictability of their responses, not on the purported intractability of number theoretic problems, such as the factoring or discrete logarithm problems. The resulting Strong PUF protocols are hence resilient against a potential advent of quantum computing, and so in line with recent efforts in post-quantum cryptography [8]. As mentioned earlier, this research thread on advanced Strong PUF protocols has led to publications at leading venues of the theoretical cryptography community, including IEEE S&P, CRYPTO, EUROCRYPT, or TCC [3, 6, 10, 11, 29, 41].

The communication scenario of said protocols is similar to classical, token-based cryptography [31], as depicted in Figure 1: Several parties are connected via *(i)* a digital channel, over which binary messages can be sent, and *(ii)* a physical channel, over which real, physical objects (like Strong PUFs) can be transferred. The adversary Eve has potential access to both channels. Since a Strong PUF's challenge-response interface is by definition publicly accessible [35], Eve and all other protocol participants can query the Strong PUF for any CRPs of their choice during their respective access periods to the PUF — while not being able to read out the Strong PUF's CRP-space completely, as it is too large [35]. We stress that the physical transfer of Strong PUFs can often be accomplished naturally and with surprisingly little effort: Whenever a customer carries a bank card plus Strong PUF from terminal to terminal in his wallet; whenever an electronic hardware is shipped from a manufacturer to an end consumer; or when a mobile device such as a laptop or smartphone is carried around and connects to different base stations; an automatic and implicit physical transfer of objects (and potentially of PUFs) takes place. Taken together, the future of Strong PUFs as cryptographic primitive seemed bright both in practice and theory, promising a broad spectrum of applications.

1.2 PUF Re-Use and Relevance of Erasable PUFs

These bright hopes were partly dashed when a class of simple, yet efficient attacks on advanced Strong PUF protocols was discovered: The so-called *"PUF Re-Use Model"* or *"Post-Protocol Access Model"* [41]. It realistically assumes that in almost all practical use cases, the same Strong PUF will be re-employed in more than one protocol run. Adversaries may thus gain physical access to the PUF not just during a run, but also *after* the run has been completed. As Strong PUFs by definition possess a publicly accessible, unprotected CRP-interface [35], such "post-protocol access" allows the read-out of any CRPs of the adversary's choice. This potentially includes CRPs that had been employed earlier by other parties, provided that the respective challenges are, or have become, known to the adversary. Post-protocol access can hence retrospectively allow attacks on previous protocol executions. Unfortunately, this simple attack method has proven effective on all currently known Strong PUF based KE, OT, or SMC schemes [41, 49].

We would like to stress that the PUF Re-Use Model *differs fundamentally* from the trivial case of a Weak PUF whose responses have been exposed to the adversary. Please recall the strict differences between Weak PUFs and Strong PUFs [35] in this context: Weak PUFs inherently are based on the hypothesis that their responses remain internal and unknown to adversaries forever. Exposing these responses hence trivially breaks security. With Strong PUFs, more or less the converse holds: Their CRP-interface by definition is designed to be public from the start. They should hence intuitively withstand any adversarial access to their CRP-interface, including the post-protocol access assumed in the PUF Re-Use Model — but actually cannot, as it turns out [36, 41, 49].

This leads to the question whether new protocols for KE, OT, or SMC could be designed that evade the above issues. Unfortunately, it was formally proven [49] that any standard Strong PUFs, whose responses are unaltered and can still be read out during post-protocol access, are not useful in building secure KE, OT, or SMC schemes. This implies that the problems arising from post-protocol access must be solved on a hardware level — not on a protocol level. They must be overcome by devising novel physical types of PUFs, which can alter some of their CRPs for good [49].

The above discussion could be interpreted as suggesting so-called Reconfigurable PUFs [12, 20, 23, 53] in order to resolve the issues of post-protocol access. Please recall that by definition [20], all of their responses can be randomly altered in one single step by a simple reconfiguration operation. However, complete, general, and non-selective reconfigurability is not what we need in our context: For illustration, consider the setting depicted in Figure 1. Let us assume that different cryptographic protocols are being run between multiple parties in this setting, using the same PUF. Some of these parties will have measured CRP-lists of this PUF earlier. Fully reconfiguring the entire PUF implies that all of these lists become obsolete, however. They would have to be measured anew.

participant and also adversaries can apply challenges and read-out responses freely [6, 29, 32] — or, in one term, a Strong PUF [35].

This implies that the reconfigured PUF needs to physically return to all respective parties, which is highly inefficient and impractical.

In sum, this renders Erasable PUFs (and their finegrained, granular erasability on a single CRP-level) the *only known possibility* [49] to fully restore the usability of Strong PUFs in advanced cryptographic protocols (such as KE, OT, or SMC) in the most general usage scenarios [36, 41].

1.3 Related Work and Fundamental Challenges

Despite their abovementioned relevance, no fully viable Erasable PUF candidates have been devised to this date. The deeper reasons are some non-trivial, quite fundamental challenges in Erasable PUF design: Recall that Erasable PUFs are a subclass of Strong PUFs, i.e., they must possess a very large number of possible inputs, and a complex, numerically hard-to-predict input-output behavior [35]. However, essentially all known Strong PUF architectures, such as Arbiter PUFs [45] and optical PUFs [30], realize this feature by a *complex interplay* of many system-internal components in the response generation. If a single response shall be altered irrecoverably in such a construction, at least one of these internal components must be modified. This will alter the targeted response; but it will also inevitably affect many other responses as well! For example, modifying a single scattering element in an optical PUF, or changing a single runtime delay in an Arbiter PUF, will necessarily influence many CRPs — not just one. This complicates or even disables finegrained erasure operations on a *single* CRP level.

The only known Erasable PUF candidate prior to our work was based on a large, monolithic crossbar architecture [36]. It carries diodes with random current-voltage characteristics at each of its nanoscale crosspoints [36]. This design, known as SHIC PUF [37], leads to a very large number of completely and information-theoretically independent CRPs. By intentionally overloading a selected diode, a breakthrough could be induced at any given crosspoint, effectively *"erasing"* the PUF-response deduced from this crosspoint *without* affecting other responses [36].

However, one substantial problem with this existing construction, which the authors of [36] did not mentioned, is that after breakthrough, the rectification rates of the broken diodes are not high enough to guarantee a fully functional read-out procedure in the large monolithic crossbar in the future. Concretely, it is reported [36] that the rectification rates drop from 10^7 to as low as 10^2 after breakthrough. This means that additional parasitic paths will arise in the large monolithic structures with every new breakdown, quickly disabling exact future read-out operations after an increasing number of erased responses.

Other publications that are directly related to our work include the following: Firstly, the already mentioned Reconfigurable PUFs [12, 20, 23, 53], which pursue a reconfiguration of the *entire* PUF with *all* its CRPs. However, this global reconfigurability leads to the protocol and efficiency issues that we described already in Section 1.2. This renders Reconfigurable PUFs not well applicable in our situation. Secondly, Controlled PUFs [13, 15] have some aspects in common with our architectures, as they also employ a trusted logic around a standard PUF in order to realize novel security features. However, their design goals and envisaged applications vastly differ from ours [13, 15]: For example, standard Controlled PUFs assume the secret storage and error correction of several earlier response values inside the trusted computing base (TCB) of the Controlled PUF. This is contrary to our approach, which does not induce any secrets inside the TCB and surrounding control logic of our Erasable PUF constructions.

1.4 Our Contributions

Within the above research landscape, we make the following novel contributions:

- We develop a formal, but easily comprehensible new framework for PUF definitions, and give the first formal definitions of Erasable PUFs. Our treatment could potentially help coining a novel, easily accessible, yet precise style in any future PUF-related definitions. Furthermore, we lead some first proofs on the exact relation between Strong PUFs and Erasable PUFs in our framework.
- Our erasable PUF design is generic in the sense that it can turn any given integrated Strong PUF with a digital challenge-response interface into an Erasable PUF. We therefore call our design "GeniePUF" for **Gen**er**i**c **E**rasable PUF. We also proved "GeniePUF" is a secure Erasable PUF, given that its underlying PUF is a secure Strong PUF.
- On the technical side, the GeniePUF approach uses red-black tree and authenticated search tree techniques in untrusted memory, while storing a public, i.e., non-secret, but authenticated root hash inside its Trusted Computing Base (TCB). The root hash's length is *independent* of the number of already erased CRPs.
- We implement our approach on Zynq FPGA, reporting detailed performance and practicality figures. We show that a CRP erasure operation takes no more than 18 μs and 10 μs for the hardware TCB and software interface, respectively, even if 100,000 CRPs have been erased previously as an example case. This time only rises mildly even for larger erased CRP sets.

1.5 Organization of This Paper

Section 2 presents a novel definitional framework for Erasable PUFs and the first formal Erasable PUF definitions. Our **Gen**er**i**c **E**rasable PUFs (GeniePUFs), which are based on programmable logic and authenticated tree structures, are discussed in Section 3. Section 4 shows the security and practicality of the GeniePUF construction, with a concrete cryptographic application (i.e., their use on bank cards/smart cards in communication terminals). Finally, the paper concludes in Section 5.

2 A FORMAL FRAMEWORK FOR ERASABLE PUFS

2.1 Basic Aspects of (Strong) PUFs

While their challenge-response behavior can be, and actually often is, modeled *mathematically*, PUFs in the end are *physical* objects. It thus makes sense to start our definitional framework by a few basic, mostly physical aspects.

Definition 1 (PUFs). A PUF P is a physical system that can be stimulated with so-called challenges c_i from a challenge set $C_P \subseteq$

$\{0,1\}^k$, upon which it reacts by producing corresponding responses r_i from a response set $R_P \subseteq \{0,1\}^m$. Each response r_i shall depend on the applied challenge, but also on manufacturing variations in P that are practically unclonable with currently existing technology. The tuples (c_i, r_i) are usually called the challenge-response pairs (CRPs) of P. If required, we explicitly write $r^P_{c_i}$ or r^P_i for denoting the response of P to challenge c_i.

We comment that it seems necessary to stipulate that PUF-responses must depend on unclonable manufacturing variations in the PUF: Only this feature distinguishes PUFs from a piece of standard, digital hardware implementing a pseudo-random function, say. Furthermore, the definition implicitly assumes that any potentially noisy PUF-responses can be stabilized via suitable error-correcting means. This allows regarding the PUF's behavior as a function F_P mapping challenges to responses, and to consider fixed CRPs (c_i, r_i), with $r_i = F_P(c_i)$. Making this assumption is in accordance with the PUF-literature, and also strongly simplifies our later treatment: It allows us to talk about a single, fixed response (after error correction) to a given challenge.

Let us next define secure Strong PUFs, one of the main PUF-subtypes. For simplicity, hereinafter, we will use "strong PUF" to represent "secure strong PUF" defined in Definition 2.

Definition 2 (Secure Strong PUFs)**.** Let P be a PUF and $\mathcal{A}$ be an adversary. P is called a $(k, t_{\text{att}}, \epsilon)$*-secure Strong PUF* with respect to $\mathcal{A}$ if $\mathcal{A}$ has a probability of at most ϵ to *"win"* the following security game:

SecGameStrong $(P, \mathcal{A}, k, t_{\text{att}})$:

1. The PUF P is handed over to $\mathcal{A}$, starting the game. [2]
2. $\mathcal{A}$ is allowed to conduct physical actions and to carry out numeric computations, potentially exploiting his physical access to P. These actions and computations are limited by the laws of physics and by $\mathcal{A}$'s individual capabilities and equipment.
3. At an adaptive point in time of $\mathcal{A}$'s choice, $\mathcal{A}$ hands back P (or whatever physically remains of it). [2, 3]
4. Then for $j = 1, \ldots, k$, the following **loop** is repeated:
   (a) A challenge c^j from P's challenge space C_P is chosen uniformly at random.
   (b) P and c^j are handed over to $\mathcal{A}$. [2]
   (c) $\mathcal{A}$ is allowed further physical actions and numeric computations, with the exception of asking P for the response of c^i for $1 \le i \le j$. These actions and computations are again limited by the laws of physics and by $\mathcal{A}$'s capabilities and equipment.
5. $\mathcal{A}$ chooses to guess a response for one of the challenges c^j, i.e., he outputs a tuple $(j^*, r^{j^*}_{\text{guess}})$, with $1 \le j^* \le k$ and $r^{j^*}_{\text{guess}} \in R_P$, and the game ends. [2]

$\mathcal{A}$ *"wins"* the game if:

- $\mathcal{A}$ has made an output $(j^*, r^{j^*}_{\text{guess}})$ and $r^{j^*}_{\text{guess}}$ is equal to the correct response of P on challenge c^{j^*}, i.e.,

$$r^{j^*}_{\text{guess}} = r^P_{c^{j^*}}.$$

- The cumulative time that has elapsed in Step 2 and in the k repetitions of Step 4c within the **loop** does not exceed t_{att}.

In all of this, the probability ϵ is taken over the random choice of all the c^{j^*}, and over all random procedures that $\mathcal{A}$ employs in the security game. ■

Notice that adversary $\mathcal{A}$ may have been lucky in that $\mathcal{A}$ queried PUF P for some challenge c^j before it was given to $\mathcal{A}$ in Step 4b (after which $\mathcal{A}$ is not allowed to query c^j any more). Let α be the fraction of all challenge-response pairs that can be retrieved from P within time t_{att} in Step 2 (we choose $\mathcal{A}$ not to do anything in Step 4c for each iteration). Then, the probability that one of c^j corresponds to one of the retrieved challenge-response pairs in Step 2 is equal to $1-(1-\alpha)^k$. If this happens, then $\mathcal{A}$ wins the game, since he knows the response and therefore can predict the response. This shows that $\epsilon \ge 1-(1-\alpha)^k$. For large challenge-response spaces, α is small such that $1-(1-\alpha)^k \approx \alpha \cdot k$ and $\epsilon \ge \alpha \cdot k$.

Proposition 1 (Strong PUFs for Different Sizes of k)**.** Let P be a PUF and $k \le k'$. For every $\mathcal{A}$ that wins **SecGameStrong** $(P, \mathcal{A}, k, t_{\text{att}})$, there is another adversary $\mathcal{A}'$ who performs the same actions as $\mathcal{A}$ (plus some dummy waiting operations) and that wins **SecGameStrong** $(P, \mathcal{A}', k', t'_{\text{att}})$, where t'_{att} is equal to t_{att} plus the time cost for the dummy waiting operations.

Proof. Without adding extra adversarial capabilities or time complexity, any adversary $\mathcal{A}$ can be modified to an adversary $\mathcal{A}'$ who is like $\mathcal{A}$ but does not to do anything in Step 4c for iterations $j = k+1, \ldots, k'$ and chooses to select j^* in the range $1 \le j^* \le k$ in Step 5. This reduces **SecGameStrong** $(P, \mathcal{A}, k, t_{\text{att}})$ to **SecGameStrong** $(P, \mathcal{A}', k', t'_{\text{att}})$. □

Our definition is a simplified, perhaps more easily accessible version of the existing, game-based Strong PUF definitions [2, 34, 40]. These usually employ multiple parameters to define Strong PUF security [2, 34]; we reduce this to merely three characteristic figures, namely the attack time t_{att}, the size k of the subset of challenges for which an adversary needs to predict one of its responses, and the adversarial guessing probability ϵ. We do not employ physical Turing machines as the formal model of computation for the adversary, since this can lead to intricate definitions [33]. Furthermore, we do not consider infinite families of PUFs as in [2, 6], avoiding an asymptotic treatment with its associated pitfalls and issues [34, 40]. Instead, we assume that one specific adversary $\mathcal{A}$ with certain (assumed) abilities is under consideration. Finally, our definition does not suppose an information-theoretic security of the Strong PUF, as some earlier works did [6]: The reason is that most Strong PUFs (such as the Arbiter PUF and variants thereof) do not possess information-theoretic, but only computational security, as all the existing modeling attacks show in passing [38, 39]. (Recall that in these modeling atacks, a small set of CRPs is collected in order to extrapolate the PUF's behavior on other CRPs; this would be provably impossible if all CRPs were information-theoretically independent.)

[2] We assume that the physical handover procedures in Step 1 and Step 3, as well as the choice and presentation of c^j in Step 4, are carried out in negligible time compared to the rest of the security game, i.e., we model them to take time of 0 sec, not causing any additional delays.

[3] Note that $\mathcal{A}$ may have potentially physically altered or even destroyed P.

Overall, we hope that our definition strikes a good balance between formal rigour and accessibility.

2.2 Erasable PUFs

Loosely speaking, Erasable PUFs are Strong PUFs (see Definition 2 and [16, 35]) with one extra property: Users can select an arbitrary challenge $c_{\text{erase}} \in C_P$, and apply a *"secure erasure operation"* **ER** for this challenge to the PUF. This operation shall irrecoverably *"erase"* the single CRP $(c_{\text{erase}}, r_{\text{erase}})$ from the PUF, *without* affecting any other CRPs. More precisely speaking, the erasure operation should affect or alter the original response r_{erase} in such a way that adversaries later cannot recover r_{erase} with a probability better than random guessing, while all other responses shall remain unchanged. The erasability operation shall be applicable k times, ideally for values of k that reach up to the size of the entire challenge space C_P of the Erasable PUF.

This leads to the following two definitions.

Definition 3 (Erasure Operations for PUFs)**.** An erasure operation **ER** for a PUF P is a specific physical or logical process that takes as input the PUF P and a challenge $c_{\text{erase}} \in C_P$, and produces as output a related PUF P' with the following properties:

- P' has the same challenge set as P.
- For all challenges $c \neq c_{\text{erase}}$, P' has a functional challenge-response behavior, and the responses of P and P' to c are equal, i.e., $r_c^{P'} = r_c^P$.
- The response of P' to challenge c_{erase} is altered or affected in a certain fashion that is specific to **ER**; for example, **ER** may routinely overwrite the original response r_{erase} by a "0", a "1", or by a fault symbol "$\perp$".

Any PUF P to which its erasure operation **ER** has been applied k times may be denoted as $P^{(k)}$, with $P^{(0)}$ being the original PUF P. For any PUF $P^{(k)}$, the set of challenges for which the erasure operation has been applied to P since its fabrication, is denoted by $\mathcal{E}(P^{(k)})$. ■

Note that the above definition does not say anything about dedicated security aspects, such as the irrecoverability of erased responses; this is taken care of next.

Definition 4 (Secure Erasable PUFs)**.** Let P be a PUF with erasure operation **ER**, k be a positive integer, and $\mathcal{A}$ be an adversary. P is called a $(k, t_{\text{att}}, \epsilon)$*-secure Erasable PUF* with respect to $\mathcal{A}$ if $\mathcal{A}$ has a probability of at most ϵ to *"win"* the following security game:

SecGameErasable $(P, \mathcal{A}, k, t_{\text{att}})$:

(1) The PUF P is handed over to $\mathcal{A}$, starting the game. [2]

(2) $\mathcal{A}$ is allowed to conduct physical actions and to carry out numeric computations, potentially exploiting his physical access to P. These actions and computations are limited by the laws of physics and by $\mathcal{A}$'s individual capabilities and equipment.

(3) At an adaptively selected point in time of $\mathcal{A}$'s choice, he hands back P $(= P^{(0)}$, see Definition 3). [2]

(4) Then for $j = 1, \ldots, k$, the following **loop** is repeated:

  (a) A challenge c_{erase}^j is chosen uniformly at random from C_P, and the CRP $(c_{\text{erase}}^j, r_{\text{erase}}^j)$ is erased from the PUF $P^{(j-1)}$. This creates a PUF $P^{(j)}$. [2]

  (b) $P^{(j)}$ and c_{erase}^j are handed over to $\mathcal{A}$. [2]

  (c) $\mathcal{A}$ is allowed to conduct physical actions and to carry out numeric computations, possibly exploiting his physical access to $P^{(j)}$. These actions and computations are again limited by the laws of physics and by $\mathcal{A}$'s individual capabilities and equipment.

(5) $\mathcal{A}$ chooses to guess one of the previously erased responses, i.e., he outputs a tuple $(j^*, r_{\text{guess}}^{j^*})$, with $1 \leq j^* \leq k$ and $r_{\text{guess}}^{j^*} \in R_P$, and the game ends. [2]

$\mathcal{A}$ *"wins"* the game if:

- $\mathcal{A}$ has made an output $(j^*, r_{\text{guess}}^{j^*})$ and $r_{\text{guess}}^{j^*}$ is equal to the original response of P on challenge $c_{\text{erase}}^{j^*}$, i.e., if
$$r_{\text{guess}}^{j^*} = r^P_{c_{\text{erase}}^{j^*}} .$$
- The cumulative time that has elapsed in Steps 2 and in the k repetitions of Step 4c within the **loop** does not exceed t_{att}.

In all of this, the probability ϵ is taken over the random choice of all the $c_{\text{erase}}^{j^*}$, and over all random procedures that $\mathcal{A}$ employs in the security game. ■

The security game of Definition 4 specifies a rather strong attack scenario: Overall k randomly chosen CRPs are successively erased from the PUF, while the adversary has access for time periods of his adaptive choice before any erasures have taken place, and also once after every single erasure operation was conducted.

Similar to Definition 2, Definition 4 deliberately is non-asymptotic, and considers only a single PUF with respect to a given adversary $\mathcal{A}$ and its capabilities (please compare the discussion following Definition 2). Similar to Proposition 1, we can prove:

Proposition 2 (Erasable PUFs for Different Sizes of k)**.** Let P be a PUF with erasure operation and let $k \leq k'$. For every $\mathcal{A}$ that wins **SecGameErasable** $(P, \mathcal{A}, k, t_{\text{att}})$, there is another adversary $\mathcal{A}'$ who performs the same actions as $\mathcal{A}$ (plus some dummy waiting operations) and that wins **SecGameErasable** $(P, \mathcal{A}', k', t'_{\text{att}})$, where t'_{att} is equal to t_{att} plus the time cost for the dummy waiting operations.

This implies that in order to be a $(k, t_{\text{att}}, \epsilon)$-secure Erasable PUF, the erasure operation must work securely for *any i*-element subset $\mathcal{E} \subseteq C_P$ of size smaller than or equal to k. k so becomes one of several quality measure of Erasable PUFs — the larger k, the better. As before, the definition was designed in order to strike a good balance between accessibility and formal rigor.

As already mentioned, Erasable PUFs are a sub-class of Strong PUFs, i.e., every Erasable PUF is also a Strong PUF automatically. This intuitive fact is in agreement with our above framework, as proven formally in the next theorem. It tells us that every Erasable PUF (according to Definition 4) is also a Strong PUF (according to Definition 2) with respect to the same adversary.

Theorem 1 (Erasable PUFs are Strong PUFs)**.** Let P be a $(k, t_{\text{att}}, \epsilon)$-secure Erasable PUF with respect to some adversary $\mathcal{A}$. Then P is a $(k, t_{\text{att}}, \epsilon)$-secure Strong PUF with respect to the same adversary $\mathcal{A}$.

The proof of Theorem 1 is given in Appendix C.1. This concludes our formal definitional framework and treatment of Erasable PUFs.

The next section will deal with the first practically viable and also *generic* silicon implementation strategy for Erasable PUFs.

3 GENERIC ERASABLE PUF DESIGN

3.1 Basic Idea and Overview

The most straightforward approach for implementing Erasable PUFs would presumably consist of the following steps: *(i)* Take an arbitrary silicon Strong PUF with digital challenge-response interface. *(ii)* Surround it by a trusted control logic that guards the application of challenges and the collection of responses. *(iii)* Keep a public, but authenticated LIST of all CRPs that have been declared *"erased"* earlier. *(iv)* Whenever a new challenge c_i is applied, the control logic compares c to LIST, and blocks it from application if c_i has been declared *"erased"* earlier.

There is one obvious issue with the above implementation strategy, though: The entire LIST needs to be an authenticated part of the TCB, in the sense that it must be unalterable for external adversaries. At the same time, however, LIST will grow larger, as more CRPs are declared erased. This implies that the size of the TCB grows with the number of declared CRPs, with the latter potentially growing rather large over the lifetime of the Erasable PUF. This is obviously undesirable.

In this section, we therefore devise and implement a construction that keeps the size of the TCB constant over the lifetime of the Erasable PUF, regardless of how many CRPs have already been erased. We show that by combining authenticated search trees [7] and red-black trees [9], LIST can be authenticated by a public, merely constant-length string RootHash, which does not grow as more CRPs are erased. RootHash is stored inside the TCB of the Erasable PUF, where it does not need to be kept secret, but only needs to be protected against adversarial manipulation of its value (i.e., it merely needs to be authentic, not secret). At the same time, LIST may be stored in public, untrusted memory. We also describe how LIST and RootHash can be updated efficiently whenever new challenges are declared *"erased"*. Our basic approach of read-out and erasure are illustrated in Figure 2 and 3, respectively.

The described approach has several upsides: Firstly, it transforms any given Strong PUF into an Erasable PUF, being **"generic"** in this sense. Secondly, its assumption of a public, but authenticated piece of data is long established in the PUF area, being similar to the authenticated, public helper data required in the error correction of Weak PUFs [35]. Thirdly, also the presumption of a surrounding, trusted control logic (or TCB) is long accepted in the field: Please recall that it is a standard ingredient of Controlled PUFs [13, 15]. Finally, as already indicated, our construction does not require any digital secret keys in the TCB. This makes it an advantageous, generic method for Erasable PUF design. We will unfold its full details over the next subsections, calling it **Gen**eri**c** **E**rasable PUF (GeniePUF).

3.2 Read-Out Mechanism of the GeniePUF

Given the basic approach of Figure 2, we need to describe two operations in order to fully specify our GeniePUF construction: Firstly, how CRPs can be read out from the GeniePUF; secondly, how CRPs can be erased from it. Both will be accomplished in this and the next subsection. We emphasize that our read-out and erasure procedures heavily rely on authenticated red-black trees [9]; readers who are not familiar with this technique can turn to Appendix A and B to obtain all relevant knowledge in a compact form.

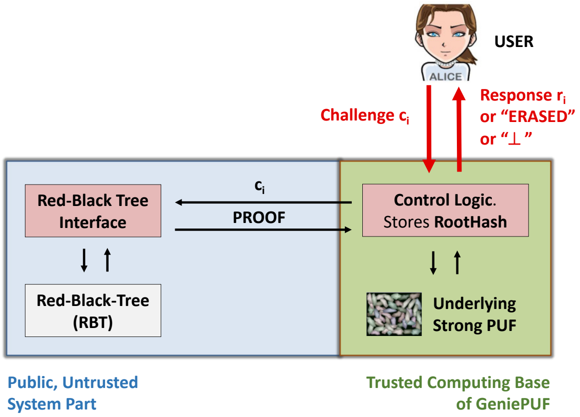
USER
ALICE
Challenge c_i
Response r_i or "ERASED" or "⊥"
Red-Black Tree Interface
c_i
PROOF
Control Logic. Stores RootHash
Red-Black-Tree (RBT)
Underlying Strong PUF
Public, Untrusted System Part
Trusted Computing Base of GeniePUF

Figure 2: Schematic illustration of the read-out mechanism of GeniePUFs (compare Scheme 1), differentiating between the public, untrusted system part (blue) and the trusted computing base of the GeniePUF (green).

Let us then turn to the read-out mechanism of the GeniePUF. As mentioned above, the LIST that contains all CRPs that have been declared *"erased"* earlier is implemented by a RBT, and is stored in public, untrusted memory (see Figure 2). The much shorter RootHash of the RBT, on the other hand, is stored within the Trusted Computing Based (TCB) of the GeniePUF in order to authenticate the entire RBT. If some USER wants to obtain CRPs from the GeniePUF, the following steps are executed:

Scheme 1: READING CRPS FROM GENIEPUFS (FIGURE 2)

(1) The USER sends a challenge c_i to the Control Logic (CL) of the GeniePUF, which is part of the GeniePUF's TCB.
(2) The CL passes on c_i to the RBT interface, which belongs to the public, untrusted system part.
(3) The RBT interface checks if c_i is in the RBT, and generates a PROOF whether c_i is in the RBT (*"proof of existence"*) or whether it is not in the RBT (*"proof of non-existence"*). Subsequently, PROOF is sent over from the RBT to the CL. The detailed procedure of proof generation and the format of the PROOF is given in Scheme 4 in the Appendix.
(4) The CL verifies the PROOF of existence or non-existence. The procedure of proof verification is presented in Scheme 5 in the Appendix.
    - If the PROOF is a valid proof of non-existence, the CL applies c_i to the Strong PUF, which is part of its TCB. It passes the obtained response r_i on to the USER.
    - If the PROOF is a valid proof of existence, then CL denies access to the PUF and outputs "ERASED" to the USER.
    - If the PROOF (either non-existence proof or existence proof) is invalid, then CL outputs "⊥" to the USER.

3.3 Erasure Mechanism of the GeniePUF

The mechanism for Erasing CRPs in the GeniePUF is again built on the basic functionality of our underlying authenticated data structure, namely red-black trees (RBTs). In a nutshell, anyone can erase CRPs from the GeniePUF, by sending an *"Erase c_i"* command to the Control Logic (CL) of the GeniePUF. Subsequently, the challenge c_i is added to the LIST, or in our case, to the RBT, thus declaring it *"ERASED"*. More precisely, the following procedure takes place:

Scheme 2: ERASING CRPS FROM GENIEPUFS (FIGURE 3)

(1) The USER sends an *"Erase c_i"* command to the CL.
(2) The CL passes on this command to the RBT interface.
(3) The RBT interface performs the same operations as step 3 in Scheme 1. Besides, if c_i is not in the RBT, the interface will also attach the information about how the RBT might rotate its structure (denoted as RotInfo). RotInfo will be sent over together with PROOF to the CL. Notice that this computation does not include computing updated hashes which RBT will receive from CL in Step 5. [4]
(4) Similar to step 4 in Scheme 1, the CL first verifies the PROOF.
  - If the PROOF is a valid proof of non-existence, the CL starts performing the erasure operation for c_i. All necessary tree structure updates, if happen in the RBT, can be replicated by CL with RotInfo. Knowing the updated tree structure and the hash values of all the nodes that require an updated hash, which are all contained in PROOF, the CL is able to compute the new hash values of all the nodes in PROOF, including a new RootHash. The CL updates the RootHash in the TCB, sends the USER an "OK" message, replies all the NewHash to the RBT interface.
  - If the PROOF is a valid proof of existence, the CL replies "OK" to the USER.
  - If the PROOF is invalid, then CL outputs "⊥" to the USER.
(5) Upon receiving the NewHash, the RBT interface updates the hashes in the RBT and completes an erasure request to GeniePUF.

4 SECURITY AND PRACTICALITY OF OUR DESIGN

4.1 Security of Our Construction

Informally speaking, and in a nutshell, the security of Scheme 1 depends on the following assumptions:

- Adversaries cannot circumvent the Control Logic (CL), applying their own challenges directly to the underlying Strong PUF, reading out the corresponding responses r_i.
- Adversaries cannot modify the CL, for example such that it cannot correctly verify the validity of PROOF. In particular, the implementation of the hash function F_{Hash} must remain correct and unchanged.
- Adversaries may read the stored RootHash, but not modify it. It is public, but authentic.

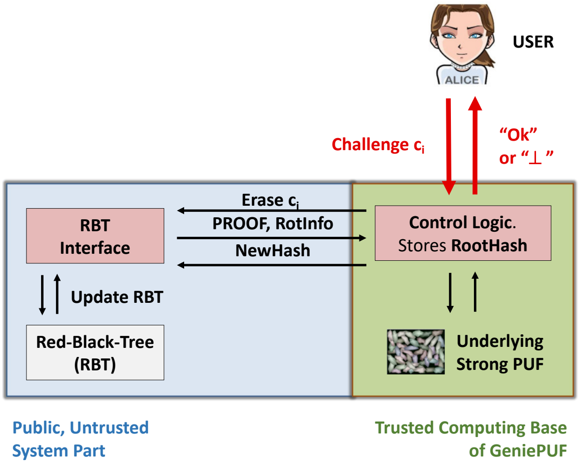

Figure 3: Schematic illustration of the erasure mechanism of GeniePUFs (compare Scheme 2), differentiating between the public, untrusted sytem part (blue) and the trusted computing base of the GeniePUF (green).

- The employed hash function F_{Hash} must be collision resistant.

Under these prerequisites, adversaries cannot read out CRPs that have successfully been declared "ERASED" earlier, and which are part of the LIST (i.e., of our RBT).

Furthermore, the required security feature of the *"Erase"* command is that a malicious RBT interface, or an adversary who intercepts the communication between the RBT interface and the CL, cannot send a Proof that is accepted by the CL, while it does not relate to an updated RBT that contains all previously erased challenges. As before, this is achieved by the collision resistance of the employed hash function F_{Hash}.

Let us start our more formal analysis of the GeniePUF construction with stipulating some terminology.

Definition 5 (GeniePUFs Based on a Given PUF P)**.** Let P be a PUF with a digital challenge-response interface. Then we use the term GeniePUF(P) to denote the Erasable PUF that is obtained by utilizing P within the construction detailed in Section 3. That is, briefly speaking, GeniePUF(P) denotes the PUF obtained from P by:

- Surrounding P with some trusted logic that guards access to P's CRP-interface, and that implements the operations of the TCB, including the verification of PROOF and the access control of P.
- Storing RootHash inside this trusted logic.
- Storing LIST outside the trusted logic.

It is inherently assumed in the GeniePUF(P) construction that:

- The adversary can access LIST and can actively overwrite and modify it.
- The adversary can read RootHash, but cannot modify it.
- The adversary cannot tamper with the functionality of the PUF-surrounding trusted logic. E.g., he cannot access the

[4] As a self-balancing binary search tree, a RBT will adjust (rotate) its tree structure to maintain the balance of itself, when it is unbalanced. Detailed description of the rotations can be found in [9], and examples can be found in Appendix B.

CRP-interface of the PUF P directly, circumventing the control logic. Or he cannot modify the erasure or read-out functionalities implemented by the trusted logic.
- The employed hash function is collision resistant. ■

The next theorem states and proves the security of this construction under the above assumptions.

Theorem 2 (Security of GeniePUF(P)). Let P be a PUF with challenge set C_P. Let $\mathcal{A}$ be an adversary for GeniePUF(P) who is modeled by Definition 5. Then GeniePUF(P) is $(k, t_{\text{att}}, \epsilon + \rho)$-secure Erasable PUF with respect to $\mathcal{A}$ if the following two conditions hold:

(1) $\mathcal{A}$ cannot compute in time t_{att} with probability $\geq \rho$ a collision (x_1, x_2) for the employed hash function F_{Hash}, i.e., $F_{\text{Hash}}(x_1) = F_{\text{Hash}}(x_2)$ and $x_1 \neq x_2$ (ρ is called the collision probability).

(2) PUF P is a $(k, t_{\text{att}}, \epsilon)$-secure Strong PUF with respect to adversary $\mathcal{A}$' that is constructed from $\mathcal{A}$ in the following fashion: $\mathcal{A}$' acts exactly like $\mathcal{A}$, but *(i)* all requested erasure operations are discarded, *(ii)* whenever a PROOF is computed, then such a PROOF is ignored (and not communicated to P), and *(iii)* any attempt by $\mathcal{A}$ to read state in RBT or control logic CL is replaced by dummy observations.

The proof sketch of Theorem 2 is presented in Appendix C.2.

Security against Physical Attacks. The following analysis builds on the taxonomy of secrets in security hardware introduced in [42]. Firstly, using the nomenclature of [42], there obviously are no *"permanent"* or *"non-volatile"* digital secrets inside the TCB or the RBT interface. This implies that the GeniePUF design is immune to any attacks stealing such digital secrets, including side channels [44] or probing attacks [50]. Depending on the employed underlying Strong PUF, other secrets may be contained in the GeniePUF: If an iPUF is used, as in our example, the physical runtime delays in the iPUF will constitute *"permanent physical secrets"*, again using the language of [42]. Furthermore, the digital values in the latches at the end of the single Arbiter PUFs within the iPUFs will constitute so-called *" volatile digital secrets"* [42]. Finally, the digital signals entering the XOR gate in the lower layer of the iPUF will constitute *"transient digital secrets"* according to [42]: Knowing the value of these signals, the attackers can successively learn the individual, single Arbiter PUFs in the structure, and subsequently break the entire iPUF.

Still, these permanent physical and volatile and transient digital secrets will arguably be harder to extract from hardware than standard, permanently stored digital keys. To start with, if the used iPUF is made large enough, it will be practically infeasible to extract these runtimes via machine learning techniques. To this end, please compare [51], where the limit of successful machine learning is found to be around 10 parallel, XORed Arbiter PUF of length 64 in the lower iPUF layer. Furthermore, it will be very difficult to physically extract the runtime delays physically in practice, albeit not impossible [46]. Also reading out the content of the latches or the transient signals entering the XOR gate during the operation of the iPUF appears complicated and more intricate than reading out a standard, permanently stored digital key from NVM.

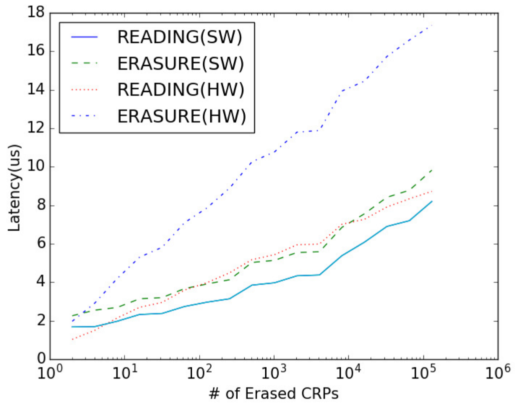

Figure 4: Latency of the software interface (SW) and hardware TCB (HW) of GeniePUF for a reading or erasure operation.

Considering active attacks, launching fault injection attacks on control logic or RootHash [4] could possibly attack our GeniePUF construction. Still, it would violate one of our security assumptions, which assumes the RootHash and the control logic are tamper resistant.

Denial of Service Attacks. By manipulating the RBT (LIST) or a PROOF, an attacker can launch denial of service attacks to our GeniePUF. However, we would like to argue in this case that as a physical object defined in Definition 1, a PUF just like any other piece of hardware can never be secure against physical denial of service attacks: The adversary can always physically alter or damage the PUF or hardware under consideration when he holds physical possession of it. One more subtle attack can be to act as a normal user and erase a large number of challenges from the GeniePUF, in the hope that a large RB tree can make a legitimate user's evaluation overwhelmingly slow. However, thanks to our red black tree structure, this attack can never be efficient and effective, because the attacker has to erase N challenges to slow down the evaluation process by $log(N)$ times.

4.2 Practicality and Performance Figures of Our Construction

Our above GeniePUF architecture has been implemented on Xilinx Zynq FPGA with a so-called *Interpose PUF* or *iPUF* [28] as its underlying PUF. Interpose PUFs are constructed from Arbiter PUFs of variable length, and consist of several parallel layers of these Arbiter PUFs, similar to the well-known XOR Arbiter PUF architectures [45]. The 64-bit Interpose PUF chosen for our specific implementation contains a 64-bit Arbiter PUF in its top layer and a 65-bit 9-XOR Arbiter PUF in its lower layer. [5] Due to the fixed length of the inputs to the hash function of the resulting GeniePUF,

[5] At the time of our implementational work, this size of the iPUF was considered secure; we remark that this no longer holds due to some recent advances in iPUF modeling attacks [47, 51]. But since our GeniePUF technique is generic, it could also be implemented with larger iPUF sizes that are secure, or with alternative future secure implementations of Strong PUFs, of course.

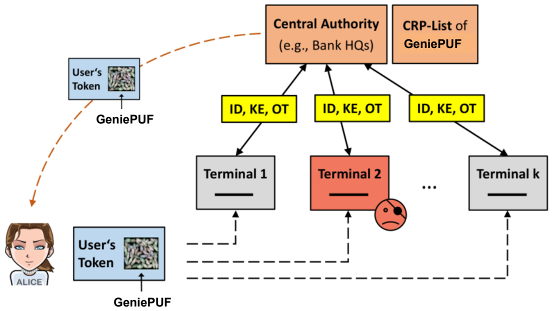

Figure 5: One concrete application example of GeniePUFs: A user Alice **carries around a token with a GeniePUF, employing it in k potentially untrusted terminals. This allows identification (ID), key exchange (KE) and oblivious transfer (OT) protocols between the terminals or PUF/token on the one hand, and the central authority on the other hand. After any KE and OT protocols, the erasure functionality of the GeniePUF must be used in order to prevent attacks [41].**

we decided to build a hash function from AES-128 in the Davies-Meyer construction [1, 27] with 64-bit collision resistance for the GeniePUF.

We measured the performance of our proof-of-concept implementation for reading and erasing CRPs. Figure 4 shows how the latency of hardware TCB and software interface grow with respect to the number of previously erased CRPs (the size of LIST). The frequency of the ARM processor and the FPGA fabric in the Zynq system are 666 MHz and 100 MHz, respectively. In Figure 4, it clearly shows that the latency grows logarithmically with respect to the number of previously erased CRPs. This is consistent with the complexity analysis of the search operation in a red-black tree. In concrete numbers, the latency is on the order of a few ten microseconds for both software interface and hardware verifier, even if the size of LIST has grown above 100,000, which is far beyond its practical need. In case multiple CRPs are needed in one authentication, one can divide the entire challenge space into disjoint subsets, and implement a challenge expansion function to derive all challenges in one subset from one seed challenge. After verifying that the seed challenge is not erased, the TCB can allow the whole challenge subset to be queried to the PUF. This method avoids repeated RB tree insertions for single authentication, and this also enables the erasure of multiple challenges by erasing one seed challenge.

4.3 Applications of Erasable PUFs

As depicted in Figure 5, we choose a bank card like scenario, in which a token carrying a GeniePUF is used in k terminals in sequence. We imagine that between the terminals or the PUF/token on the one hand, and some central authority (CA) or bank headquarters (HQ) on the other hand, cryptographic protocols are run, which are all based on this PUF. Structurally similar communication settings would occur in many other applications: If payments are being made at various shop terminals by a consumer's smart phones, if access cards are used in order to gain entry to different facilities, or if smart phones enroll in different cells of a network, just to name a few examples. In all of these scenarios, Erasable PUFs might be used beneficially. We stress that some of the terminals may be controlled by the adversary (while it is not known, of course, which of them are infiltrated and which are not). This means that adversaries can gain access to the PUF between protocol runs.

The run PUF-protocols shall include simple, plain CRP-based identification, as first proposed in [14, 30], KE as first suggested in [6, 33, 48], and OT as first put forward in [6, 32]. Their completion enables identification and secure communication between the terminals and PUF-carrying token on the one hand, and the CA or bank HQ on the other hand. It also allows any secure-two party computations or zero-knowledge proofs between these parties (which can be built on OT, as long known [21]). The exact identification, KE and OT schemes based on Erasable PUFs usable in this context are exactly like the existing Strong PUF based ID [30], KE [6, 33, 48] or OT [6, 32] schemes from the literature — with one twist: At the end of the KE and OT scheme, all CRPs employed in the protocol must be erased from the PUF in a final step. This ensures the long-term security of KE and OT [41, 49]. At the end of the identification protocol, no further steps need to be added, since no long-term confidentiality is required there.

5 SUMMARY AND FUTURE WORK

PUFs have enjoyed the intense attention of the security community for around one and a half decades by now. While their main applications initially consisted of key storage and system identification, a no less interesting second research strand has evolved in recent years: So-called Strong PUFs have been suggested as cryptographic primitive in advanced protocols such as key exchange (KE), bit commitment (BC), oblivious transfer (OT), or secure multiparty computation (SMC).

One fundamental and unresolved problem in the area is the re-use of employed PUFs in multiple protocol runs, however. While such re-use appears imperative from an economic and efficiency perspective, it creates severe security issues [41]: All abovementioned KE, OT, and SMC protocols can be broken in such a scenario. It can be formally proven [49] that this issue cannot be overcome by additional protocol or software steps alone. Instead, it requires resolution on the hardware level and a novel PUF-type, so-called Erasable PUFs. By definition, they allow that single CRPs can be *"altered"* or *"erased"* for good, without affecting any other CRPs.

This paper now for the first time proposes a fully viable construction for Erasable PUFs, which in addition is generic, i.e., which can turn any Strong PUF with a digital challenge-response interface into an Erasable PUF. In greater detail, our approach named *"GeniePUF"* is based on a trusted control logic that surrounds the given Strong PUF. This comes at the price of extending the trusted computing base of the system, now including the PUF's control logic. This might seem unusual at first sight. On the other hand, the same approach has long been accepted in the PUF-area in other contexts, for example in the construction of so-called Controlled PUFs [13]. Furthermore, our GeniePUFs require a public, but authenticated piece of information accompanying the PUF. Again, this assumption might appear exotic at first glance, but has long been introduced and accepted in the standard key derivation from Weak PUFs. Overall, our construction hence rests on previously known and somewhat principles within the PUF-area. Also the use of a hash function in connection with PUFs, even inside the TCB (which makes things more tedious) has been used before us, namely in the context of Controlled PUFs [13].

We also to our knowledge presented the first formal definitional framework for Erasable PUFs. Using a parametric, non-asymptotic style of definitions, not considering infinite PUF-families, but single PUFs and their properties, we tried to clearly define our objects of study. Compared to other approaches, the compact and semi-formal style of our framework makes it easily accessible, also for non-theorists. Our hope is that this might allow the definitions (and similar, future ones that might adopt their style) to act as link between PUF-theorists and PUF-practitioners. We also proved the relationship between strong PUFs and erasable PUFs (GeniePUFs). In that, we tried to demonstrate how one can reason somewhat formally and rigidly about security while using our semi-formal definitional framework.

Future Work. We believe that various future research opportunities arise from our work. Starting with the practical and implementational side, further optimization of our logical Erasable PUFs together with prototyping in FPGAs and ASICs seems a worthwhile endeavour. Other Strong PUFs than the iPUF [28] can be used in connection with the generic GeniePUF technique. On the theory side, our novel definitional framework will first of all hopefully spark a new style of easily acccessible, intuitive PUF-definitions in follow-up works. Secondly, follow-up theory works could utilize Erasable PUFs in advanced protocols, in which PUFs can indeed be securely re-used, going beyond the original set-up and communication model of [6]. Formalizing and proving the security of such new schemes, for example in the universal composition framework (compare [6]), appears interesting for future theory papers.

ACKNOWLEDGMENTS

Chenglu Jin was supported by NSF award CNS 1617774, NYU CCS, and NYU CUSP. Wayne Burleson was supported by NSF/SRC grant CNS-1619558. Marten van Dijk was supported by NSF award CNS 1617774. Ulrich Rührmair acknowledges support by BMBF-project QUBE and by BMBF-project PICOLA. Finally, we would like to thank the reviewers for their very valuable comments!

REFERENCES

[1] AES, NIST. 2001. Advanced encryption standard. *Federal Information Processing Standard, FIPS-197* 12 (2001).

[2] Frederik Armknecht, Daisuke Moriyama, Ahmad-Reza Sadeghi, and Moti Yung. 2016. Towards a unified security model for physically unclonable functions. In *Cryptographers' Track at the RSA Conference.* Springer, 271–287.

[3] Saikrishna Badrinarayanan, Dakshita Khurana, Rafail Ostrovsky, and Ivan Visconti. 2017. Unconditional UC-secure computation with (stronger-malicious) PUFs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 382–411.

[4] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. 2012. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE* 100, 11 (2012), 3056–3076.

[5] Rudolf Bayer. 1972. Symmetric binary B-trees: Data structure and maintenance algorithms. *Acta informatica* 1, 4 (1972), 290–306.

[6] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. 2011. Physically uncloneable functions in the universal composition framework. In *Advances in Cryptology CRYPTO 2011.* Springer, 51–70.

[7] Ahto Buldas, Peeter Laud, and Helger Lipmaa. 2000. Accountable certificate management using undeniable attestations. In *Proceedings of the 7th ACM conference on Computer and communications security.* ACM, 9–17.

[8] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. 2016. *Report on post-quantum cryptography.* US Department of Commerce, National Institute of Standards and Technology.

[9] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, et al. 2001. *Introduction to algorithms.* Vol. 2. MIT press Cambridge.

[10] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. 2014. Feasibility and infeasibility of secure computation with malicious PUFs. In *Advances in Cryptology CRYPTO 2014.* Springer, 405–420.

[11] Ivan Damgård and Alessandra Scafuro. 2013. Unconditionally secure and universally composable commitments from physical assumptions. In *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 100–119.

[12] Ilze Eichhorn, Patrick Koeberl, and Vincent van der Leest. 2011. Logically reconfigurable PUFs: Memory-based secure key storage. In *Proceedings of the sixth ACM workshop on Scalable trusted computing.* ACM, 59–64.

[13] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. 2002. Controlled physical random functions. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual.* IEEE, 149–160.

[14] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. 2002. Silicon physical random functions. In *Proceedings of the 9th ACM conference on Computer and communications security.* ACM, 148–160.

[15] Blaise Gassend, Marten van Dijk, Dwaine Clarke, Emina Torlak, Srinivas Devadas, and Pim Tuyls. 2008. Controlled physical random functions and applications. *ACM Transactions on Information and System Security (TISSEC)* 10, 4 (2008), 3.

[16] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. 2014. Physical unclonable functions and applications: A tutorial. *Proc. IEEE* 102, 8 (2014), 1126–1141.

[17] Daniel E Holcomb, Wayne P Burleson, and Kevin Fu. 2007. Initial SRAM state as a fingerprint and source of true random numbers for RFID tags. In *Proceedings of the Conference on RFID Security*, Vol. 7.

[18] Daniel E Holcomb, Wayne P Burleson, and Kevin Fu. 2009. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans. Comput.* 58, 9 (2009), 1198–1210.

[19] C Jaeger, M Algasinger, U Rührmair, G Csaba, and M Stutzmann. 2010. Random pn-junctions for physical cryptography. *Applied Physics Letters* 96, 17 (2010), 172103.

[20] Stefan Katzenbeisser, Ünal Kocabaş, Vincent van Der Leest, Ahmad-Reza Sadeghi, Geert-Jan Schrijen, and Christian Wachsmann. 2011. Recyclable pufs: Logically reconfigurable pufs. *Journal of Cryptographic Engineering* 1, 3 (2011), 177–186.

[21] Joe Kilian. 1988. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing.* ACM, 20–31.

[22] Sandeep S Kumar, Jorge Guajardo, Roel Maes, Geert-Jan Schrijen, and Pim Tuyls. 2008. The butterfly PUF protecting IP on every FPGA. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust.* IEEE, 67–70.

[23] Klaus Kursawe, Ahmad-Reza Sadeghi, Dries Schellekens, Boris Skoric, and Pim Tuyls. 2009. Reconfigurable physical unclonable functions-enabling technology for tamper-resistant storage. In *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on.* IEEE, 22–29.

[24] Keith Lofstrom, W Robert Daasch, and Donald Taylor. 2000. IC identification circuit using device mismatch. In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056).* IEEE, 372–373.

[25] Roel Maes, Vincent Van Der Leest, Erik Van Der Sluis, and Frans Willems. 2015. Secure key generation from biased PUFs. In *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 517–534.

[26] Roel Maes, Anthony Van Herrewege, and Ingrid Verbauwhede. 2012. PUFKY: A fully functional PUF-based cryptographic key generator. In *International Workshop on Cryptographic Hardware and Embedded Systems.* Springer, 302–319.

[27] Alfred J Menezes, Paul C van Oorschot, and Scott A Vanstone. 1996. *Handbook of applied cryptography.* CRC press.

[28] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten van Dijk. 2019. The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2019).

[29] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. 2013. Universally composable secure computation with (malicious) physically uncloneable functions. In *Advances in Cryptology–EUROCRYPT 2013.* Springer, 702–718.

[30] Ravikanth Pappu, Ben Recht, Jason Taylor, and Neil Gershenfeld. 2002. Physical one-way functions. *Science* 297, 5589 (2002), 2026–2030.

[31] Radia J Perlman and Stephen R Hanna. 2001. Methods and systems for establishing a shared secret using an authentication token. US Patent 6,173,400.

[32] Ulrich Rührmair. 2010. Oblivious transfer based on physical unclonable functions. In *Trust and Trustworthy Computing.* Springer, 430–440.

[33] Ulrich Rührmair. 2011. Physical Turing Machines and the Formalization of Physical Cryptography. *IACR Cryptology ePrint Archive* 2011 (2011), 188.

[34] Ulrich Rührmair, Heike Busch, and Stefan Katzenbeisser. 2010. Strong PUFs: models, constructions, and security proofs. In *Towards hardware-intrinsic security.* Springer, 79–96.

[35] Ulrich Rührmair and Daniel E Holcomb. 2014. PUFs at a glance. In *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 347.

[36] Ulrich Rührmair, Christian Jaeger, and Michael Algasinger. 2011. An attack on PUF-based session key exchange and a hardware-based countermeasure: Erasable PUFs. In *Financial Cryptography and Data Security*. Springer, 190–204.

[37] Ulrich Rührmair, Christian Jaeger, Matthias Bator, Martin Stutzmann, Paolo Lugli, and György Csaba. 2011. Applications of high-capacity crossbar memories in cryptography. *Nanotechnology, IEEE Transactions on* 10, 3 (2011), 489–498.

[38] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. 2010. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 237–249.

[39] Ulrich Rührmair and Jan Sölter. 2014. PUF modeling attacks: An introduction and overview. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1–6.

[40] Ulrich Rührmair, Jan Sölter, and Frank Sehnke. 2009. On the Foundations of Physical Unclonable Functions. *IACR Cryptology ePrint Archive* 2009 (2009), 277.

[41] Ulrich Rührmair and Marten van Dijk. 2013. Pufs in security protocols: Attack models and security evaluations. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 286–300.

[42] Ulrich Rührmair. 2020. SoK: Towards Secret-Free Security. In *2020 Workshop on Attacks and Solutions in Hardware Security (ASHES@ CCS 2020)*.

[43] Peter Simons, Erik van der Sluis, and Vincent van der Leest. 2012. Buskeeper PUFs, a promising alternative to D flip-flop PUFs. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 7–12.

[44] François-Xavier Standaert. 2010. Introduction to side-channel attacks. In *Secure integrated circuits and systems*. Springer, 27–42.

[45] G Edward Suh and Srinivas Devadas. 2007. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*. ACM, 9–14.

[46] Shahin Tajik, Enrico Dietz, Sven Frohmann, Jean-Pierre Seifert, Dmitry Nedospasov, Clemens Helfmeier, Christian Boit, and Helmar Dittrich. 2014. Physical characterization of arbiter pufs. In *Cryptographic Hardware and Embedded Systems–CHES 2014*. Springer, 493–509.

[47] Johannes Tobisch, Anita Aghaie, and Georg T Becker. [n.d.]. Combining Optimization Objectives: New Machine-Learning Attacks on Strong PUFs. ([n. d.]).

[48] Pim Tuyls and Boris Škorić. 2007. Strong authentication with physical unclonable functions. In *Security, Privacy, and Trust in Modern Data Management*. Springer, 133–148.

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

[50] Huanyu Wang, Domenic Forte, Mark M Tehranipoor, and Qihang Shi. 2017. Probing attacks on integrated circuits: Challenges and research opportunities. *IEEE Design & Test* 34, 5 (2017), 63–71.

[51] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten van Dijk, and Ulrich Rührmair. 2020. Splitting the interpose PUF: A novel modeling attack strategy. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 97–120.

[52] Wenjie Xiong, André Schaller, Nikolaos A Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. 2016. Runtime accessible DRAM PUFs in commodity devices. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 432–453.

[53] Le Zhang, Zhi Hui Kong, Chip-Hong Chang, Alessandro Cabrini, and Guido Torelli. 2014. Exploiting process variations and programming sensitivity of phase change memory for reconfigurable physical unclonable functions. *IEEE Transactions on Information Forensics and Security* 9, 6 (2014), 921–932.

A BACKGROUND ON AUTHENTICATED SEARCH TREES AND RED-BLACK TREES

An authenticated search tree was introduced in [7] as an undeniable attester. In the context of our GeniePUF, an untrusted Red-Black Tree (RBT) interface is used, which manages LIST of size n. It takes a challenge as input, and generates a proof of non-existence/existence of this challenge in LIST. Notice that, the length of the proof is only $O(log(n))$ long. Upon receiving the non-existence/existence proof, the TCB around the PUF can then verify the proof by checking against a constant-sized ($O(1)$) root hash stored in the TCB. This root hash does not need to be kept secret, i.e., it can be known to adversaries; it must merely be secure against alteration or overwriting by adversaries.

To further improve the performance of an authenticated search tree in the worst case scenario, where a standard search tree will become extremely unbalanced, we merge a red-black tree (RB tree) [5, 9] with the authenticated search tree in the untrusted memory. In short, a red-black tree is one self-balancing binary search tree structure [5, 9], which checks and balances the depth of the tree after every node insertion and deletion. Hence, a Red-Black tree can guarantee searching in $O(log(n))$ time in the average and the worst scenario, where n is the total number of nodes in the tree [9].

In the following, we will describe the necessary procedures of our authenticated red-black trees (e.g., we only describe node insertion, not deletion, because in the GeniePUF application, LIST can only grow). In particular, we present the high level idea of the following basic schemes of our combined tree structure to prepare the readers for understanding this paper.

An authenticated search tree is sorted according to the challenges stored in each node, and it is constructed in such a way that each node consists of a unique challenge c_i in the LIST and a hash value $h_i = F_{\text{Hash}}(c_i, left(c_i).hash, right(c_i).hash)$, where $left(c_i).hash$ and $right(c_i).hash$ are the hash values stored in the left or right child of node c_i, respectively. The hash values of the children of the bottom leaves are considered to be 0 by default. An example tree structure is shown in Figure 6.

Scheme 3: Searching for a Challenge c_i in a RBT

(1) The RBT interface receives a challenge c_i.
(2) The RBT interface searches for c_i, using the RBT as an ordinary binary search tree.
(3) In the end, it results in two cases:
   - If c_i is found, then a pointer to the node associated with c_i is returned.
   - If the binary search for c_i within RBT reaches a leaf node, where no challenge is stored, then the interface returns a pointer to the parent node of the leaf node. (This parent node is the lowest node in the tree whose child c_i would have supposed to be, if c_i was part of the RBT.) In the example in Figure 6, the returned pointer will be pointing to the node containing c_4.

Scheme 4: Generating a PROOF of Existence/Non-Existence of a Challenge c_i in a RBT

(1) After a search for c_i, as described in Scheme 3, is completed in the LIST (either found or not found), the RBT interface gets a node in the tree from the search procedure. It sets this node as the starting node of the PROOF.
(2) The interface adds the challenge of the starting node and the hash values stored in the children nodes of the starting node into the PROOF. Again, taking the example of Figure 6, the information added is c_4 and the hash values of its two children (two *nil* nodes).
(3) Then the RBT interface fetches the challenge of the node and the hash value in the sibling node of each node along the path in the tree from the starting node to the root of the tree to generate the completed PROOF of non-existence/existence

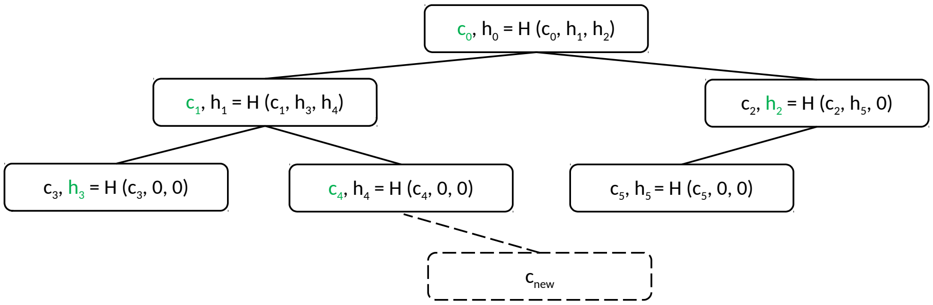

Figure 6: Proof construction in an authenticated search tree. Suppose that one needs to prove that c_{new} does not exist in the authenticated search tree (containing c_0 to c_5). For example, the dashed node shows the location where c_{new} is supposed to be. The green information is included in the proof of non-existence for c_{new}. Note that the hash value stored in the left child of c_4 is also needed in the proof, but it is omitted in the diagram, because it is a *nil* node in the tree.

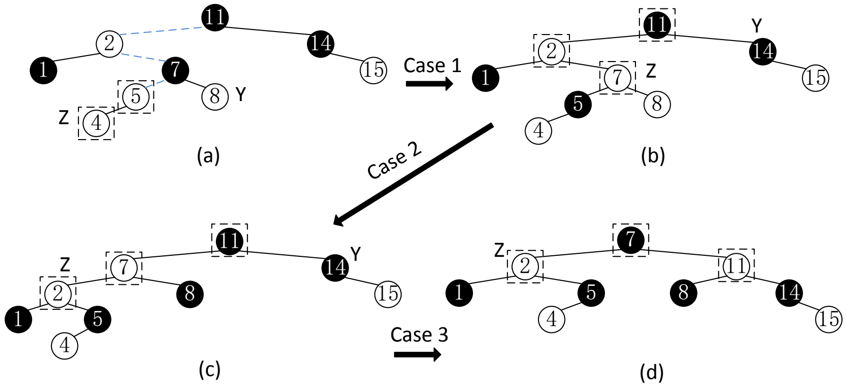

Figure 7: Insertion of a new node 4.

of c_i. E.g., adding (c_1, h_3) and (c_0, h_2) into the PROOF, as shown in Figure 6.

(4) It returns the completed PROOF.

The proof construction process is also illustrated in Figure 6.

Scheme 5: Verifying a PROOF of Existence/Non-Existence

(1) All the proofs generated by the RBT interface have to be verified by the trusted control logic CL. After a proof is received, the CL checks the starting node first. If it is a proof of non-existence, the CL checks whether the left/ right child of the starting node is a leave node based on whether c is smaller/ greater than the challenge in the starting node. In the case of an existence proof, the CL verifies the order of the two children and the starting node. If any of the above check failed, return "⊥".
(2) Then the CL hashes every node from the starting node of the proof all the way to the root, using the challenge value of each node and their sibling hash values provided in the proof. The order of left and right child is determined by comparing two consecutive challenges in the PROOF. The final result is RootHash'.
(3) Check if RootHash' = RootHash stored in the TCB:
   - If yes, we conclude that the PROOF is a valid proof. Based on whether its an existence proof or a non-existence proof, we conclude whether c_i is in the LIST or not.
   - If no, the PROOF is considered as invalid, and we conclude that either the LIST or the RBT Interface has been tampered with by an attacker.

Scheme 6: Adding a New Challenge c_i to the RBT

(1) In the case that a new challenge c_i needs to be added to the LIST, the RBT interface first proves that c_i is not in the LIST using the above schemes.
(2) If the non-existence of c_i gets accepted by the verifier, then c_i is added as a child of the node returned by the search procedure.
(3) After insertion, a red-black tree fix-up is triggered. It may rotate the structure of the tree to re-balance it. More details about the red-black tree fixup can be found in the example in the Appendix B and [9].
(4) After fix-up, a new RootHash will be generated by the trusted control logic CL according to the fixup information of the tree and the proof of non-existence used in Scheme 3.

Note that, based on the way the authenticated search tree is constructed and verified, its security solely relies on the collision resistance of the underlying hash function.

B EXAMPLE ROTATION OF AN AUTHENTICATED RB TREE

Figure 7 depicts an example of consecutive operations in Red-Black Tree Insert-Fixup, see [9]. (a) A new node 4 is inserted. The dashed path in (a) is PROOF. All of the information in nodes 5, 7, 2 and 11 are included in PROOF, together with the hash values of nodes 8, 1 and 14, called the sibling's hash values. In order to verify non-existence, we need to reconstruct the root hash using PROOF and compare with the trusted root hash stored in the TCB. In addition, we need to check whether new node 4 is added at the correct location, which means $2 < 4 < 5$, and node 5 has no left child. Here, case 1 in [9] applies, so node 5 and 7 are recolored but the structure remain the same.

There are six possible cases in a RB tree fixup, in which only case 2, 3, 5 and 6 in [9] will rotate the structure of the tree; this example shows three cases (the other three cases are similar in that they are mirrored versions of the three in the example). In (b),(c) and (d), the nodes in dashed blocks are the nodes which hash values need to be updated; the transition from (b) to (c) is a rotation and the transition from (c) to (d) is a rotation. Note that, PROOF already provides all the information needed for updating these hash values. In this example, in order to compute the hash of node 2, 7 and 11 in (d), we need the hash value of node 5, which was updated in case 1 during the transition from (a) to (b), and the hash values of nodes 1, 8 and 14, which are exactly the sibling's hash values that are contained in PROOF.

C PROOFS OF THEOREMS

In the following proofs we assume that ignoring operations or communication does not increase the original execution time t_{att} of an adversary.

C.1 Proof of Theorem 1

Proof. We will show the contraposition of the above statement, assuming that P is *not* an $(k, t_{\text{att}}, \epsilon)$-secure Strong PUF with respect to some adversary $\mathcal{A}$.

By Definition 2, this implies that there exists an adversary $\mathcal{A}$ who is capable of winning the security game **SecGameStrong** $(P, \mathcal{A}, k, t_{\text{att}})$ of Definition 2 with probability greater than ϵ. This, in turn, means that $\mathcal{A}$ can predict the correct response to one out of k uniformly randomly chosen challenges $c^j \in C_P$ with probability greater than ϵ, whereby the time that $\mathcal{A}$ requires for his physical actions and numeric computations does not exceed t_{att}.

We notice that the very same adversary $\mathcal{A}$ will also win the security game **SecGameErasable** $(P, \mathcal{A}, k, t_{\text{att}})$ with probability greater than ϵ. The reason for this is that the execution of the security game **SecGameErasable** $(P, \mathcal{A}, k, t_{\text{att}})$ with $c^j = c^j_{\text{erase}}$ is identical to the execution of the security game **SecGameStrong** $(P, \mathcal{A}, k, t_{\text{att}})$ because adversary $\mathcal{A}$ in **SecGameErasable** $(P, \mathcal{A}, k, t_{\text{att}})$ never attempts to query an erased challenge $c^j = c^j_{\text{erase}}$. This implies that P is not a $(k, t_{\text{att}}, \epsilon)$-secure Erasable PUF, completing our contraposition argument. □

C.2 Proof Sketch of Theorem 2

Proof Sketch. Let $\mathcal{A}$ be any adversary that is modeled by Definition 5. We define a series of games that reduce

$$\textbf{SecGameErasable}\ (P, \mathcal{A}, k, t_{\text{att}}),$$

with probability of winning denoted by ϵ_{erase}, to

$$\textbf{SecGameStrong}\ (P, \mathcal{A}', k, t_{\text{att}}),$$

where ϵ is the probability of winning as stated in the theorem.

We first modify **SecGameErasable** by assuming an adversary $\mathcal{A}^0$ who is like $\mathcal{A}$ but who cannot produce a valid PROOF for an invalid claim that a challenge was not erased in its interactions with GeniePUF(P). We call this new game **SecGameErasable**^{0} and denote the probability of winning this game by ϵ_0. By the implicit assumptions on the capabilities of the adversary in Definition 5, we know that the control logic CL and PUF P cannot be modified. Therefore, the only way to produce a valid PROOF for an (erased) challenge c in RBT is to find a collision for the hash function. By Theorem 1 in Section 6.2 of [7], the probability of finding a valid PROOF is at most ρ. This shows that

$$\epsilon_{\text{erase}} \leq \epsilon_0 + \rho.$$

Not being able to provide a valid PROOF for an invalid claim in **SecGameErasable**^{0} means that the GeniePUF(P) does not produce responses for erased challenges. This is similar to the same game **SecGameErasable**^{0} where in Step 4a only a challenge c^j_{erase} is chosen at random but not erased, and with the restriction that the adversary is not allowed to query c^j_{erase} after c^j_{erase} is given to the adversary in Step 4b. We call this game **SecGameErasable**^{1}. We now define $\mathcal{A}^1$ as adversary $\mathcal{A}^0$ by discarding any erasure operations which $\mathcal{A}^0$ asks for in Step 2 or Step 4c (these operations cannot lead to feedback from GeniePUF(P) which contains predictive information that can be used in Step 5). For $\mathcal{A}^1$, we can now conclude that game **SecGameErasable**^{1} has winning probability ϵ_1 for which

$$\epsilon_0 = \epsilon_1.$$

Notice that **SecGameErasable**^{1} does not implement any erasure operations. Because **SecGameErasable**^{1} disallows querying any of the c^j_{erase} after being selected in Step 4a and communicated to $\mathcal{A}^1$ in Step 4b of game **SecGameErasable**^{1}, we know that the control logic CL of GeniePUF(P) simply provides direct access to P for the queries by $\mathcal{A}^1$. Therefore, the control logic of GeniePUF(P) provides direct access to P in **SecGameErasable**^{1} and provides no other functionality. This means **SecGameErasable**^{1} results directly in a game for PUF P where we have conceptually stripped away the control logic of GeniePUF(P).

Unrolling all the steps in **SecGameErasable**^{1} for P shows its equivalence with **SecGameStrong**. We now define $\mathcal{A}'$ as $\mathcal{A}^1$ where any attempt by $\mathcal{A}^1$ to read state in RBT or control logic CL is replaced by dummy observations. For $\mathcal{A}'$, we may now conclude that **SecGameStrong** has winning probability

$$\epsilon_1 = \epsilon.$$

By combining all inequalities and equations we have

$$\epsilon_{\text{erase}} \leq \epsilon + \rho.$$

□