A preliminary version of this paper appears in the proceedings of the *31st IEEE Computer Security Foundations Symposium (CSF 2018)*, © IEEE 2018, DOI: 10.1109/CSF.2018.00013. This is the full version.

Self-Guarding Cryptographic Protocols against Algorithm Substitution Attacks

Marc Fischlin Sogol Mazaheri

Cryptoplexity, Technische Universität Darmstadt, Germany
{marc.fischlin, sogol.mazaheri}@cryptoplexity.de

August 20, 2018

Abstract. We put forward the notion of self-guarding cryptographic protocols as a countermeasure to algorithm substitution attacks. Such self-guarding protocols can prevent undesirable leakage by subverted algorithms if one has the guarantee that the system has been properly working in an initialization phase. Unlike detection-based solutions they thus proactively thwart attacks, and unlike reverse firewalls they do not assume an online external party. We present constructions of basic primitives for (public-key and private-key) encryption and for signatures. We also argue that the model captures attacks with malicious hardware tokens and show how to self-guard a PUF-based key exchange protocol.

1 Introduction

Classical security notions in cryptography, such as indistinguishability of encryptions, assume that the involved cryptographic algorithms behave in the specified way. In the real world, however, we have little control over, or insights into, the design criteria or the software implementing the algorithms, even on our own systems. The idea that an adversary may tamper with the implementation, or embed a backdoor in the specification, was suggested already 20 years ago by Young and Yung under the name of *kleptography* [YY96, YY97]. In a kleptographic setting the adversary interacts with an implementation of the cryptographic scheme which may be faulty, or let alone malicious, such that subliminal leakage of confidential data may even go unnoticed. In terms of security, in such settings all bets are off.

1.1 Detecting Substitution Attacks

Recently, the topic of malicious implementations as a potential mean for mass surveillance has received a lot of attention. The formal study of the so called *algorithm substitution attacks* (ASAs) was initiated by Bellare, Paterson, and Rogaway [BPR14] with the example of symmetric encryption schemes. The adversary's goal is roughly formalized as being able to break security while remaining undetectable from the users. Viewed the other way, resistance against ASAs therefore means that either data is not leaked, or the leakage is detectable. The detectors were later given the catchy name *watchdogs* [RTYZ16].

Detecting algorithm substitution attacks can be hard, and sometimes even be impossible. It was shown that randomized symmetric encryption is prone to ASAs that can leak the secret key, while avoiding detection by any efficient watchdog with black-box access to the algorithm [BPR14, BJK15]. Advanced attacks, exploiting techniques such as input-triggered misbehavior and imperfect decryptability, indicate that reliable detection is indeed hard to achieve, as observed by Degabriele, Farshim, and Poettering [DFP15].

Taking into account the impossibility results, some works have suggested to use deterministic schemes with unique ciphertexts such that one can compare against the expected values [BPR14, BH15, BJK15, DFP15].

Even in situations where detection is theoretically possible, it is arguably very difficult to design proper watchdogs in practice. A watchdog gets access to an implementation which, due to the nature of the attack, may be arbitrarily subverted, and the watchdog has to decide if any efficient adversary is able to violate its security. At the same time, different (algorithmic and software) versions do not allow to easily check against a specific code. In other words, it is unclear which kinds of irregularities the watchdog should look for. For instance, deterministic schemes are considered detectable by online watchdogs, since they can compare the output of the possibly subverted algorithm with the expected output at runtime [DFP15]. Aside from being rather inefficient, this assumes that the watchdog has a good implementation of the same algorithm at hand and that scans are performed while the system is active.

Furthermore, the perhaps trickiest attack arises when implementations behave honestly only as long as they are under scrutiny, say, through an offline watchdog. However, malicious behavior can be triggered to wake up at a later point in time. Such attacks are called *time bombs* and become active in some state, or at some point in time. They have been discussed in the context of software and also in the domain of cryptographic hardware backdoors and trojans, e.g., [WS11, DFS16]. To prevent against time bombs in hardware tokens that implement a deterministic function, Dziembowski, Faust, and Standaert [DFS16] make use of a semi-online watchdog that regularly tests the tokens against the specification that they are supposed to implement, while malicious input triggers are prevented via masking, a technique which is also useful in our setting.

Conceptually, a malicious software update can also be thought of as a time bomb. Two prominent testimonies are the heartbleed vulnerability in the open source library OpenSSL and the Juniper Dual EC incident. The heartblead bug has been introduced with version 1.0.1 in 2011 and went unnoticed for approximately two years (see `heartbleed.com`). In 2015, Juniper Networks announced that the source code of ScreenOS, the operating system of their VPN routers, was maliciously modified in 2012 [CMG$^+$16]. Although one can speculate about whether these and similar attacks were inadvertent or not, they showcase the possibility of substitution attacks being performed in the real world as a mean of mass-surveillance.

The problem of checking can even be acute if hardware components are involved. For example, in case of physically unclonable functions (PUFs) this seems to be impossible: A good PUF ideally implements a random and unclonable function (unlike hardware tokens considered by Dziembowski et al. [DFS16] which implement a specified function), but the internal computations are not assessable. It is unclear what the watchdog should check for, maybe except for basic properties such as the absence of collisions. Furthermore, the watchdog may not be able to check output values later if it does not have access to the PUF anymore. The infeasiblility of verifying security of hardware tokens recently motivated Camenisch, Drijvers, and Lehmann [CDL17] to design an anonymous attestation protocol which achieves privacy even with subverted trusted platform modules (TPM).

Another issue with watchdogs is that they need to be trustworthy entities and their implementations also need to be reliable. If the watchdog colludes with a mass-surveillance agency or is subverted itself, then detection may fail. Even worse, in some scenarios the detection algorithm requires access to the secret key to check [BPR14, DFP15], introducing other potential security risks. The latter may make detection also hard in case of hardware components, if keys are not allowed to leave the devices that they are stored on.

1.2 Preventing Subliminal Channels

Considering the difficulty of detecting algorithm substitution attacks, a question that comes to mind is if we can neutralize attacks without requiring to detect them first. While the watchdog approach is reactive, it cannot obscure loss of crucial information, but only allows to detect it, we would expect a solution which proactively prevents leakage in the first place. Although neutralizing algorithm substitution attacks is still a highly challenging task, it can be more promising than detection-based approaches in terms of both security and efficiency.

Indeed cryptographic reverse firewalls, introduced by Mironov and Stephens-Davidowitz [MS15], follow the approach of prevention [MS15, DMSD16]. The idea of reverse firewalls is to distribute the trust between the party and a firewall. The outgoing communication, say, a signature, is first routed through the firewall which may take further cryptographic steps, such as verifying the signature and re-randomizing it, in order to prevent subliminal channels. As long as one of the two parties is trustworthy and has a proper implementation, no information can be transferred through the firewall.

Reverse firewalls may not be readily applicable to every existing protocol. In fact, the goal of [MS15, DMSD16] is to design protocols that can be used with reverse firewalls. How to design amenable protocols for symmetric-key primitives, or when using hardware tokens, without complex detection mechanisms remains open.

Another important work in the line of protection mechanisms instead of detection techniques is the study of backdoored pseudorandom generators (BPRGs) by Dodis et al. [DGG$^+$15]. They showed that BPRGs can be immunized by applying a non-trivial function (e.g., a PRF or a seeded extractor) to the outputs of a possibly backdoored pseudorandom generator. The setting, however, disallows the adversary to replace the PRG algorithm, except for injecting a backdoor.

In two recent works by Russel et al. [RTYZ16, RTYZ17] the watchdog model is combined with some prevention mechanism. Their approach is based on a split-program methodology, where an algorithm is split into deterministic and probabilistic blocks that can be individually tested by the watchdog. This allows prevention of for instance rejection-sampling attacks by combining two independent sources of randomness, and prevention of input-triggered attacks by adding a random value to the input. Despite remarkable improvements regarding the power that a watchdog in the split-program model gains, some of our criticisms, e.g., requiring a good implementation and failing to detect state-dependent attacks, remain.

1.3 Self-Guarding Schemes

Our contribution is to provide an alternative defense mechanism to reverse firewalls which, too, proactively thwarts ASAs, but does not depend on external parties. We focus on a setting where the party at some point holds a genuine version of the algorithm, before the algorithm gets substituted by e.g., a malicious software update, or before a time bomb triggers the malicious behavior of the algorithm. In other words, our “security anchor” is the assumption of having a secure initial phase. We call a cryptographic scheme that is secure despite making black-box use of possibly subverted underlying schemes *self-guarding*. Such a scheme uses information (from now on called *samples*) gathered from its underlying primitives during their good initial phase in addition to basic operations to prevent leakage later on, or to implement a new protocol securely without implementing the required primitives securely from scratch. We emphasize that subverting or exfiltrating samples (e.g., pairs of message and ciphertexts, which depend on users’ keys) would require targeted attacks, which is a considerably more difficult attack to carry out in a large scale than algorithm substitution attacks.

1.3.1 Related Approaches

Our approach shares the idea of a trusted initialization phase with several other methods in the literature. In the area of program self-correction [BLR90] an algorithm can take advantage of a program which computes incorrectly on a small fraction to always output correct answers with high probability. This bootstrapping is similar to our idea here, only that we use temporary correctness (and security) of the program. In self-correction, as well as program checking [BK89], it is important to not trivialize the problem by implementing a trusted program oneself. Instead, one should only use basic operations on top. The same austerity principle applies in our setting.

The concept also appears in the context of digital certificates. A technique called HTTP Public Key Pinning (HPKP) [EPS15], albeit argued about, is a trust-on-first-use technique for checking the validity of certificates. On first usage certificates are declared as trustworthy ("pinned") such that substitution of certificates in subsequent executions becomes infeasible.

Finally, in interactive protocols involving physical unclonable functions (PUFs) or other hardware tokens, the question of security in the presence of malicious tokens has been brought up (e.g., [OSVW13, Rüh16, BKOV17]). Here, the sender of the token typically first holds a genuine version of the token. The adversary may substitute the token later, when in transmission. This corresponds to our setting with a trusted initialization phase and subversion afterwards, only that the protocol involves two parties and hardware tokens.

1.3.2 Comparison to Watchdogs and Reverse Firewalls

As mentioned before, self-guarding schemes, as well as reverse firewalls, prevent leakage by construction. The difference is how the security anchor is provided: In reverse firewalls it is ensured by trust distribution, in self-guarding schemes it is based on a temporary trust phase. Self-guarding applies more smoothly to symmetric primitives and hardware tokens, but for other primitives currently comes with a performance inferior to today's reverse firewall solutions.

At first glance one might think that the initialization phase of self-guarding schemes could simply be executed by the watchdog with the specified program, such that we immediately get a detecting solution. However, our self-guarding schemes will pass state between the phases, whereas watchdogs typically do not forward data to individual users. Furthermore, although one could in principle combine our approach with some detection mechanism, self-guarding does not allow to spot malicious behavior innately. Another noteworthy difference between self-guarding and the watchdog model is that self-guarding schemes do not even need the subverted algorithm in the beginning.

1.4 Constructions

We show how to build self-guarding solutions for some basic primitives, including public-key and private-key encryption, and signatures. To show that our model applies to hardware tokens, too, we also discuss how to self-guard PUF-based key exchange protocols if the adversary can substitute tokens in transmission. While the general idea of passing samples of the primitive in question from the initialization phase to the execution is shared by all solutions, the techniques differ in details and also in terms of security guarantees and efficiency.

We first give a simple and efficient construction for a self-guarding IND-CPA-secure public-key encryption scheme from any homomorphic encryption scheme, e.g., ElGamal encryption. Our scheme is self-guarding even against stateful subversions of the underlying scheme. Yet, the downside is that, if the subverted scheme is stateful, we are limited to only encrypting as many messages securely as we have sample ciphertexts from the first phase. Our construction is inspired by an elegant idea by Russel et al. [RTYZ17] to prohibit input-triggered attacks in encryption schemes. This is achieved by adding a random message to the input of an encryption algorithm and sending the random message along with the ciphertext.

The second construction provides a self-guarding symmetric-key encryption scheme for IND-CPA-security, starting with any regular IND-CPA-secure scheme. Here the number of encryptions is again limited by the number of available samples, and, moreover, the message space is bounded. Despite these limitations, we find this construction quite intriguing, since the only other proposal for subversion-resisting randomized symmetric encryption is in the split-program model [RTYZ17].

Our third construction is a self-guarding signature scheme. It is built upon any deterministic EUF-CMA-unforgeable signature scheme. This time, however, the overhead is bigger than in case of encryption, and it only self-guards against stateless subversion of the underlying scheme. In contrast, it can be securely used to sign arbitrarily often after the substitution took place. Moreover, contrary to re-randomizing reverse firewalls for signatures, as proposed in [AMV15], it does not rely on an honest implementation of the verification algorithm for signing. Moreover, for our self-guarding signatures we neither need to restrict the adversarial queries to random messages, nor do we rely on a (relaxed or perfect) verifiability condition, a property roughly stating that signatures under the subverted signing algorithm must still be verifiable. The latter property has been used in [AMV15]. In our case only basic operations and black-box calls to the signing algorithm are required.

Finally we give a PUF-based key-exchange protocol that is self-guarding against subversion of the PUF with malicious, stateful, and encapsulated PUFs. This is noteworthy as for more complex tasks such as oblivious transfer there are negative results concerning such malicious PUFs [vDR12, DFK$^+$14, Rüh16]. Our key-exchange protocol has 4 rounds and uses only a single genuine sample from the initialization phase for deriving each key. It thus matches the non-self-guarding PUF-based protocols in terms of samples.

2 Security Model for Self-Guarding

2.1 Preliminaries

Notation. A string s is an element of $\{0,1\}^*$. By $|s|$ we denote the length of s, and $s||s'$ is the concatenation of strings s, s'. By $\{0,1\}^\ell$ we denote the set of strings of length ℓ. A special symbol $\perp \notin \{0,1\}^*$ indicates an error. For a finite set S we let $s \xleftarrow{\$} S$ denote a uniformly and independently sampled element s from S. A queue Q is an abstract collection of ordered elements. A new element e can be added to the queue using an enqueue function $\mathsf{enq}(Q, e)$, and the oldest element in the queue can be accessed and removed from it using a dequeue function $e \leftarrow \mathsf{deq}(Q)$. This makes queues a first-in-first-out collection. Using the function $\{0,1\} \leftarrow \mathsf{is\text{-}empty}(Q)$ we can check whether the queue is empty, where we denote an empty queue by $[\,]$.

Syntax. To distinguish genuine from potentially malicious implementations, we use a notation similar to [RTYZ16, RTYZ17], i.e, we use indices $\textsc{genuine}$ for a trusted and genuine implementation, and $\textsc{subv}$ for a possibly malicious implementation.

We are interested in protocols Π which use a possibly subverted primitive Σ. We require Π to obey a specific interface. In particular, it should provide means for generating parameters for the scheme and sampling their algorithm interfaces. More formally, given a cryptographic scheme Σ, we define $\Pi^\Sigma :=$ $(\Pi.\mathsf{Gen}^\Sigma, \Pi.\mathsf{Sample}^\Sigma, \Pi.X_1^\Sigma, \ldots, \Pi.X_n^\Sigma)$ for some $n \in \mathbb{N}$, where

- $\Pi.\mathsf{Gen}^\Sigma(1^\lambda) \xrightarrow{\$} \kappa = (\kappa_s, \kappa_p)$. On input of a security parameter 1^λ, this probabilistic algorithm outputs secret parameters κ_s and public parameters κ_p.
- $\Pi.\mathsf{Sample}^\Sigma(\kappa) \xrightarrow{\$} \Omega$. On input of parameters $\kappa = (\kappa_s, \kappa_p)$, this probabilistic algorithm outputs a collection Ω of N input-output samples of $\Pi.X_i^\Sigma$ for some $1 \leq i \leq n$. The overall number N of samples is determined by the protocol Π and may depend on the security parameter.
- $\Pi.X_i^\Sigma$ are placeholders for other functionalities of Π, for all i with $1 \leq i \leq n$.

We remark here that Π can basically take two different roles. It can either attempt to provide a different, possibly more complex functionality, while remaining secure despite using the subverted algorithm $\Sigma_{\textsc{subv}}$, or it can simply immunize a possible attack in $\Sigma_{\textsc{subv}}$. For the former role, one may think of Π as a key exchange protocol using some cryptographic primitive Σ, and the security game may capture the indistinguishability of the derived keys. Intuitively, the adversary's goal is now to take advantage of the algorithm substitution attack to break the security game. As an example of the latter case, Π and Σ can have a similar functionality, for example providing the usual interfaces for encryption and decryption. In this case, Π's task is to neutralize a potential subversion attack on Σ.

Simplicity. In principle, preventing an attack is trivial to achieve in real executions, simply by having Π deploy its own secure implementation of $\Sigma_{\textsc{genuine}}$, ignoring the potentially substituted implementation $\Sigma_{\textsc{subv}}$. To avoid such issues we assume that Π makes only black-box use of Σ and only implements very basic extra steps for the immunization. In other words, in a practical construction the internal part of Π concerning the immunization, i.e., excluding the queries to Σ and possibly an own functionality, must be as simple as possible. This assumption is important in order to keep the trusted component, i.e., Π, as small as possible, such that it is easy to implement correctly and hence too hard for an adversary to subvert.

Correctness. For a meaningful definition we require the genuine implementations, i.e., $\Pi^{\Sigma_{\textsc{genuine}}}$, to be correct. Since our main objective here is preventing ASAs, we generally do not expect a correct functionality from $\Pi^{\Sigma_{\textsc{subv}}}$, i.e., in the event of subversion. In particular, if Π detects subversion of $\Sigma_{\textsc{subv}}$, it may simply output an error message $\perp$. However, for Π to be able to meaningfully use $\Sigma_{\textsc{subv}}$ as a black box, we do assume that $\Sigma_{\textsc{subv}}$ respects the interface of the corresponding genuine scheme $\Sigma_{\textsc{genuine}}$, i.e., it has the same set of algorithms and the number and spaces of inputs and outputs.

2.2 Cryptographic Games

The advantage of a subverting adversary can be measured against its advantage in breaking the security of a scheme Π (with primitive Σ) with respect to a security game Sec by substituting the original and genuine implementation of the primitive $\Sigma_{\textsc{genuine}}$ by a malicious implementation $\Sigma_{\textsc{subv}}$.

We follow [HH09] and [RTYZ16] in defining the security of standard cryptographic schemes. Our definition will be general enough to capture both regular security games as well as self-guarding games, such that we already include parameters κ and samples Ω as part of the input of the security game. For an ordinary security game one may think of parameters κ_p and κ_s as being the public and secret key, respectively, and Ω being empty.

Definition 2.1 (Cryptographic Game). *A cryptographic game for a scheme Π is defined by a probabilistic algorithm Sec and an associated constant $\delta \in [0,1)$. On input of scheme parameters κ and potentially a set of samples Ω, the algorithm $\mathsf{Sec}(\kappa,\Omega)$ interacts with an adversary $\mathcal{A}(1^\lambda)$ and outputs a Boolean* WON. *We denote the result of this interaction by* WON $\xleftarrow{\$} \mathsf{Sec}^{\Pi}_{\mathcal{A}}(\kappa,\Omega)$. *The advantage of an adversary $\mathcal{A}$ in the game Sec is defined as:*

$$\mathsf{Adv}^{\mathsf{Sec}}_{\Pi,\mathcal{A}}(\kappa,\Omega) = \Pr\left[\mathsf{Sec}^{\Pi}_{\mathcal{A}}(\kappa,\Omega) = \mathsf{true}\right] - \delta.$$

Here the probability is over the random choices of the game, the adversary, and the values $\kappa \xleftarrow{\$} \Pi.\mathsf{Gen}(1^\lambda)$ and $\Omega \xleftarrow{\$} \Pi.\mathsf{Sample}(\kappa)$.

We say that the scheme Π is Sec-secure if for any PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{Sec}}_{\Pi,\mathcal{A}}(\kappa,\Omega)$ is negligible.

As explained in the introduction we assume that users in the security game have access to a genuine version of algorithm Σ in the beginning. Hence, our self-guarding game allows the user to initially query the correct (with respect to the intended behavior) algorithm $\Sigma_{\textsc{genuine}}$ and provide Π with some samples

Ω that can be used to prohibit the adversary from winning Sec. The substitution may only happen after the first phase.

2.3 Self-Guarding Schemes

To proceed to the definition of self-guarding schemes, we need to investigate more closely the role of the parameter and sample generation step. In the self-guarding game SGuard we consider two phases. During the first phase the challenger has access to a genuine version of algorithm Σ_{GENUINE}. There, the challenger can initialize $\Pi^{\Sigma_{\text{GENUINE}}}$ and have N samples from Σ_{GENUINE} be created and stored in a collection Ω (which are meant to be chosen and used by the user in the real execution).

Afterwards, the challenger starts the second phase of the self-guarding game by calling the adversary $\mathcal{A}(\texttt{subst}, \kappa_p)$ on a command $\texttt{subst}$ and the public parameters κ_p, giving $\mathcal{A}$ the opportunity to provide an arbitrary (while still respecting the original interface as explained above) implementation Σ_{SUBV} for Σ and a state $\mathsf{st} \in \{0,1\}^*$, based on the public parameters κ_p and the knowledge about algorithm Σ_{GENUINE}, of course. In the subsequent steps the challenger will use the original algorithm Σ_{GENUINE} or the subverted version Σ_{SUBV}. The choice is made according to a fixed value β. In either case the challenger will have the adversary $\mathcal{A}(\texttt{sec}, \kappa_p, \mathsf{st})$, called on a command $\texttt{sec}$, the public parameters and the state obtained from the subverting adversary, play the security game Sec for the scheme Π^{Σ_β} with parameters (κ, Ω).

The self-guarding ability of Π under subversion of Σ now states that the adversary's success probabilities in winning the game Sec should not increase significantly with the subverted algorithm, i.e., should not depend significantly on the value of β, which indicates if the original or the subverted algorithm is used.

```
$\mathsf{SGuard}^{\Pi^\Sigma,\beta}_{\mathsf{Sec},\mathcal{A}}(1^\lambda)$ with $\beta \in \{\text{GENUINE}, \text{SUBV}\}$

        —— trusted setup phase ——
    $\kappa = (\kappa_s, \kappa_p) \xleftarrow{\$} \Pi.\mathsf{Gen}^{\Sigma_{\text{GENUINE}}}(1^\lambda)$
    $\Omega \xleftarrow{\$} \Pi.\mathsf{Sample}^{\Sigma_{\text{GENUINE}}}(\kappa)$

        —— subversion phase ——
    $\Sigma_{\text{SUBV}}, \mathsf{st} \xleftarrow{\$} \mathcal{A}(\texttt{subst}, \kappa_p)$
    $\text{WON} \xleftarrow{\$} \mathsf{Sec}^{\Pi^{\Sigma_\beta}}_{\mathcal{A}(\texttt{sec},\kappa_p,\mathsf{st})}(\kappa, \Omega)$
    $\textbf{return}$ WON
```

Figure 1: Game for self-guarding of Π against subversion of Σ.

Definition 2.2 (Self-guarding against Subversion)**.** *Let* Σ *and* Π *be cryptographic schemes, and let* Sec *be a security game for* Π*. The advantage of an adversary* $\mathcal{A}$ *in the self-guarding game of Figure 1 is defined by:*

$$\mathsf{Adv}^{\mathsf{SGuard},\mathsf{Sec}}_{\Pi^\Sigma,\mathcal{A}}(\lambda) := \Pr\left[\mathsf{SGuard}^{\Pi^\Sigma,\text{SUBV}}_{\mathsf{Sec},\mathcal{A}}(1^\lambda) = \mathsf{true}\right] - \Pr\left[\mathsf{SGuard}^{\Pi^\Sigma,\text{GENUINE}}_{\mathsf{Sec},\mathcal{A}}(1^\lambda) = \mathsf{true}\right].$$

We say that Π *is* self-guarding *with respect to* Sec *against subversion of* Σ*, if for all* PPT *adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{SGuard},\mathsf{Sec}}_{\Pi^\Sigma,\mathcal{A}}(\lambda)$ *is negligible.*

Intuitively, the above definition requires that the security of a self-guarding scheme Π should not significantly decrease if an adversary subverts the underlying primitive Σ. As discussed before, the simplicity of the guarding performed by Π is crucial in practical applications. In particular, we assume that Π does not implement its own secure version of Σ, nor implement "heavy" detection procedures. This also implies that Π is usually not able to verify correctness of Σ itself, still allowing the adversary to modify Σ at will.

Our definition is (almost) non-adaptive in the sense that the subverted algorithm is chosen before the actual security game starts, but it may depend on the public parameters. This complies with some efforts in the literature, such as the subversion-resistant signature scheme in [RTYZ16] where the subverted algorithm may not even depend on the signer's public key. The subversion attacks on symmetric encryption schemes in [BPR14, DFP15] are also non-adaptive in nature. Such notions provide a basic level of robustness in the setting of mass surveillance where dedicated attacks may be too cumbersome to mount. At the same time, targeted attacks may still be an important aspect, e.g., when the signer is a certification authority such that forging signatures allows to create arbitrary certificates. We stress that there are adaptive notions in the literature, for example for the subversion-resistant signature schemes by Ateniese, Magri, and Venturi [AMV15], but in general the distinction is not explicit.

Our notion is strong enough to capture time bombs and logical bombs in software. The former ones are code parts which get executed at, or after, a certain point in time; the latter ones get executed if some specific condition is met. The input-triggered subversion of Degabriele et al. [DFP15] is a special case of the concept of logical bombs. While we do not have a notion of time in our model, the adversary in our model can in principle provide an algorithm which uses the original algorithm as a subroutine and only enters a special mode after some calls, if the algorithm can be stateful and, say, keep a counter value. Logical bombs, such as input-triggers, can be implemented directly in the subverted algorithm.

3 Self-Guarding Public-Key Encryption

In this section we show how to build a self-guarding encryption scheme from any homomorphic encryption scheme, to achieve substitution resistance against chosen-plaintext attacks. Its guarding mechanism is simple, efficient, and for typical instantiations such as under ElGamal encryption it does not need to perform any modular exponentiation, nor to change anything on the decryptor's side. Another advantage is that the solution enjoys security even in the case that the subverted encryption algorithm keeps state, a property which is usually hard to achieve. The downside is that if the subverted scheme is stateful, we can only encrypt as long as a sufficient number of fresh samples is still available, since a stateful algorithm can store the old samples.

3.1 Preliminaries

Informally, homomorphic encryption schemes allow one to perform operations on encrypted messages by performing efficient computations on their ciphertexts. Here we recall the formal definition. Below we assume that the message space $\mathcal{M}$ with some efficiently computable operation "$\circ$" forms a group (where the message space usually depends on the security parameter or the public key, but we omit this reference for sake of simplicity). Analogously, we assume that the ciphertext space $\mathcal{C}$ forms a group with some efficiently computable operation "$\diamond$". Furthermore, one can efficiently compute inverses in the group.

Definition 3.1 (Homomorphic Encryption Scheme)**.** *A homomorphic public-key encryption scheme* $\mathcal{HE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *with associated message group* $(\mathcal{M}, \circ)$ *and ciphertext group* $(\mathcal{C}, \diamond)$ *consists of three probabilistic polynomial-time algorithms:*

- $\mathsf{Gen}(1^\lambda) \xrightarrow{\$} (\mathsf{sk}, \mathsf{pk})$*: On input the security parameter* 1^λ *this algorithm generates a secret key* sk *and a public key* pk*.*
- $\mathsf{Enc}(\mathsf{pk}, m) \xrightarrow{\$} c$*: On input a public key* pk *and a message* $m \in \mathcal{M}$ *this algorithm outputs a ciphertext* $c \in \mathcal{C}$*.*

- $\mathsf{Dec}(\mathsf{sk}, c) \to m$*: On input a secret key* sk *and a ciphertext* $c \in \mathcal{C}$*, this deterministic algorithm outputs a message* $m \in \mathcal{M}$*.*

For any $\lambda \in \mathbb{N}$*, any* $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$*, any messages* $m, m' \in \mathcal{M}$ *the following conditions hold:*

Correctness: $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Homomorphism: $\mathsf{Enc}(\mathsf{pk}, m \circ m')$ *has the same distribution as* $\mathsf{Enc}(\mathsf{pk}, m) \diamond \mathsf{Enc}(\mathsf{pk}, m')$*.*

The classical example is the ElGamal encryption scheme, where ciphertexts $c = (g^r, \mathsf{pk}^r \cdot m)$ are pairs of elements from a group $\mathcal{G} = \langle g \rangle$ of prime order q, and messages are from $\mathcal{G}$ as well. The operations are multiplication in $\mathcal{G}$ for messages, and component-wise multiplication in $\mathcal{G}$ for ciphertexts.

3.2 Construction

The idea of our generic construction of a self-guarding scheme $\mathcal{HE}^{\text{sg}}$, described formally in Figure 2, is as follows. In the sampling phase we generate multiple ciphertexts of random messages $m_{\$,i}$. At this point, the encryption algorithm still complies with the specification, such that the samples are valid encryptions. Since we do not need any specific order of these samples, we will assume that they are stored in a queue structure and that we access the queue with the usual enq and deq commands, and check if the queue is empty via $\mathsf{is\text{-}empty}$.

Next, when encrypting a given message m, we call the (potentially now subverted) encryption algorithm to encrypt the message $m \circ m_{\$,i}$ for a fresh message $m_{\$,i}$ from the sample list. The idea is that the subverted algorithm then only gets to see a random (i.e., blinded) message for producing the ciphertext. Once we obtain the ciphertext we aim to undo the message blinding via the homomorphic property, dividing out the ciphertext for $m_{\$,i}$. For instance, when we use ElGamal encryption this corresponds to two modular multiplications and an inversion in the group. Remarkably, we do not need to be able to distinguish valid and invalid ciphertexts returned by the encryption algorithm, which saves us for example from performing an exponentiation for such a check for ElGamal encryption. Note that although re-randomizing samples allows for an unlimited number of secure encryptions, such involved techniques, which quasi means to implement one's own encryption algorithm, should be avoided.

```
$\mathcal{HE}^{\text{sg}}.\mathsf{Gen}(1^\lambda)$
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathcal{HE}.\mathsf{Gen}(1^\lambda)$
return $(\mathsf{sk}, \mathsf{pk})$
```

```
$\mathcal{HE}^{\text{sg}}.\mathsf{Sample}(\mathsf{pk})$
$\Omega \leftarrow []$
for $i = 1..N$ do
    $m_{\$,i} \xleftarrow{\$} \mathcal{M}$
    $c_{\$,i} \xleftarrow{\$} \mathcal{HE}.\mathsf{Enc}(\mathsf{pk}, m_{\$,i})$
    $\mathsf{enq}(\Omega, (m_{\$,i}, c_{\$,i}))$
return $\Omega$
```

```
$\mathcal{HE}^{\text{sg}}.\mathsf{Dec}(\mathsf{sk}, c)$
$m \leftarrow \mathcal{HE}.\mathsf{Dec}(\mathsf{sk}, c)$
return $m$
```

```
$\mathcal{HE}^{\text{sg}}.\mathsf{Enc}(\mathsf{pk}, \Omega, m)$
if $\mathsf{is\text{-}empty}(\Omega)$ then
    return $\perp$
$(m_\$, c_\$) \leftarrow \mathsf{deq}(\Omega)$
$c \xleftarrow{\$} \mathcal{HE}.\mathsf{Enc}(\mathsf{pk}, m \circ m_\$)$
$c^{\text{sg}} \leftarrow c \diamond c_\$^{-1}$
return $c^{\text{sg}}$
```

Figure 2: Self-guarding encryption scheme $\mathcal{HE}^{\text{sg}}$ from homomorphic encryption scheme $\mathcal{HE}$.

Correctness. As long as fresh samples are available and the underlying scheme $\mathcal{HE}$ is not under a subversion attack, correctness of our encryption scheme $\mathcal{HE}^{\text{sg}}$ follows immediately from correctness and the homomorphic property of $\mathcal{HE}_{\text{GENUINE}}$. Considering that the encryption algorithm $\mathcal{HE}^{\text{sg}}.\mathsf{Enc}$ practically stops working afterwards, correctness beyond that point is clearly not provided.

```
$\text{IND-CPA}^{\mathcal{E}}_{\mathcal{A}}(\kappa, \Omega)$
    $b \xleftarrow{\$} \{0,1\}$
    $b' \xleftarrow{\$} \mathcal{A}^{\vee(\kappa,\Omega,b,\cdot,\cdot)}(\kappa_p)$
    return $(b = b')$
```

```
$\vee(\kappa, \Omega, b, m_{\text{left}}, m_{\text{right}})$
    if $|m_{\text{left}}| \neq |m_{\text{right}}|$ then return $\perp$
    $m_0 \leftarrow m_{\text{left}}, m_1 \leftarrow m_{\text{right}}$
    $c \xleftarrow{\$} \mathcal{E}.\mathsf{Enc}(\kappa, \Omega, m_b)$
    return $c$
```

Figure 3: IND-CPA game for encryption schemes $\mathcal{E}$.

3.3 Security

The notion of IND-CPA follows the common left-or-right security game and is given in Figure 3 in our terminology. We capture both public-key and private-key encryption simultaneously, by setting $\kappa_p = \mathsf{pk}$ and $\kappa_s = \mathsf{sk}$ resp. $\kappa_p = \perp$ and $\kappa_s = \mathsf{k}$. Looking ahead, the notion for private-key encryption will be used in Section 4. Recall once more that the game basically describes the second phase of substitution attacks. Also, in the self-guarding game of Figure 1, if the adversary always chooses $\mathcal{E}_{\text{SUBV}} = \mathcal{E}_{\text{GENUINE}}$ and we have an empty list Ω we obtain the standard security notions without a substitution attack.

Theorem 3.2. *The encryption scheme $\mathcal{HE}^{sg}$ from Figure 2 is self-guarding with respect to* IND-CPA-*security against subversion of $\mathcal{HE}$, if $\mathcal{HE}$ is an* IND-CPA-*secure homomorphic encryption scheme.*

Proof. Assume that adversary $\mathcal{A}$ plays the self-guarding game $\mathsf{SGuard}^{\mathcal{HE}^{\mathrm{sg}\,\mathcal{HE}},\beta}_{\text{IND-CPA},\mathcal{A}}$ defined in Figure 1. We argue that $\mathcal{A}$'s probabilities for predicting the secret bit b in the two settings, $\beta = \text{GENUINE}$ and $\beta = \text{SUBV}$, are almost equal.

The case $\beta =$ genuine. Consider first the case that the security game uses the actual scheme $\mathcal{HE}_{\text{GENUINE}}$. Then $\mathcal{A}$'s probability of predicting b is negligible close to $\frac{1}{2}$. To see this note that $\mathcal{A}$, upon receiving pk, provides the subverted algorithm $\mathcal{HE}_{\text{SUBV}}$. This encryption algorithm is then ignored. It follows that in each challenge query $m_{\text{left}}, m_{\text{right}}$ of $\mathcal{A}$, where the remaining number of samples is not exhausted yet, the adversary receives an encryption of m_{left} or of m_{right}, only computed as the product of genuine ciphertexts, derived via $\mathsf{Enc}_{\text{GENUINE}}$. But since the homomorphic property says that this has the same distribution as a fresh encryption of the message, it follows immediately that the probability of predicting b is negligibly close to $\frac{1}{2}$ by the IND-CPA-security of $\mathcal{HE}$.

The case $\beta =$ subv. Next, consider the case that the security game now uses the subverted algorithm $\mathcal{HE}_{\text{SUBV}}$ in the challenge queries. In each such query for a pair of messages $m_{\text{left}}, m_{\text{right}}$ we use a random message $m_\$$ to mask the challenge message and encrypt the masked message under the subverted algorithm. The message $m_\$$ has been encrypted under the genuine algorithm $\mathsf{Enc}_{\text{GENUINE}}(\mathsf{pk}, m_\$)$ in the sampling phase. The final ciphertext c^{sg} is derived by multiplying the first ciphertext with the inverse of the second one.

Suppose now that, instead of encryption $m_\$$ under $\mathsf{Enc}_{\text{GENUINE}}$ for the second ciphertext in each challenge query, we encrypt an independent random message $m'_\$$ in the second ciphertext and compute the final ciphertext from that encryption. Then the adversary would only learn the random message $m_\$ \circ m_{\text{left}}$ or $m_\$ \circ m_{\text{right}}$, possibly leaked through the subverted algorithm $\mathsf{Enc}_{\text{SUBV}}$, and the encryption of an independent random message $m'_\$$. This also covers the case that the ciphertext in the challenge phase is malformed, e.g., does not belong to the correct subgroup. In other words, each challenge query yields an answer which is independently distributed from the bit b. The adversary's success probability for predicting b is then at most the guessing probability of $\frac{1}{2}$.

It remains to argue that encrypting $m'_\$$ instead of $m_\$$ does not significantly add to $\mathcal{A}$'s success probability. But this follows straightforwardly from the IND-CPA-security of $\mathcal{HE}$. For this we use $\mathcal{A}$ to build an adversary $\mathcal{B}$ against $\mathcal{HE}$. The adversary $\mathcal{B}$ initially receives a public key pk and forwards it to $\mathcal{A}$. Adversary $\mathcal{A}$ then provides the subverted algorithm $\mathcal{HE}_{\text{SUBV}}$. Then $\mathcal{B}$ simulates the rest of the self-guarding game, picking the bit $b \xleftarrow{\$} \{0,1\}$ itself, but trying to predict an external secret bit b' in its IND-CPA-game.

Adversary $\mathcal{B}$ answers each challenge query $m_{\text{left}}, m_{\text{right}}$ of $\mathcal{A}$ for a message with a ciphertext c^{sg}, computed as follows. Adversary $\mathcal{B}$ picks a random message $m_\$ \xleftarrow{\$} \mathcal{M}$, computes $m_\$ \circ m_{\text{left}}$ (for $b = 0$) resp. $m_\$ \circ m_{\text{right}}$ (for $b = 1$). It encrypts this message under $\mathsf{Enc}_{\text{SUBV}}$ and pk to get a ciphertext c. It picks another random message $m'_\$$ and forwards $m_\$$ and $m'_\$$ to its own challenge oracle to get a ciphertext $c_\$$. It returns $c^{\text{sg}} \leftarrow c \diamond c_\$^{-1}$ to adversary $\mathcal{A}$.

When $\mathcal{A}$ eventually outputs a guess for bit b, adversary $\mathcal{B}$ checks if the guess is correct. If so, it outputs the prediction 0 for bit b', else it outputs 1.

For the analysis note that, if $\mathcal{A}$'s success probability drops significantly from the case that one correctly encrypts $m_\$$ to the case where one encrypts $m'_\$$, then this would immediate a contradiction to the IND-CPA-security of $\mathcal{HE}$. That is, letting $\mathcal{B} = 0$ and $\mathcal{B} = 1$ denote the events that $\mathcal{B}$ outputs 0 and 1, respectively, and $\textsc{won}_\mathcal{A}$ denote the event that $\mathcal{A}$ predicts b correctly, we have:

$$\begin{aligned}
\Pr[\mathcal{B} = b'] &= \frac{1}{2} \cdot \Pr[\mathcal{B} = 0 \mid b' = 0] + \frac{1}{2} \cdot \Pr[\mathcal{B} = 1 \mid b' = 1] \\
&= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[\mathcal{B} = 0 \mid b' = 0] - \Pr[\mathcal{B} = 0 \mid b' = 1]) \\
&= \frac{1}{2} + \frac{1}{2} \cdot (\Pr[\textsc{won}_\mathcal{A} \mid b' = 0] - \Pr[\textsc{won}_\mathcal{A} \mid b' = 1]) .
\end{aligned}$$

The difference in the parentheses is non-negligible, by assumption, such that our algorithm $\mathcal{B}$ has a non-negligibly larger prediction probability than $\frac{1}{2}$. □

4 Self-Guarding Symmetric Encryption

In this section we present a self-guarding mechanism for randomized symmetric encryption. We do not assume any restriction on the attack strategy. In particular, our construction is self-guarding against biased-ciphertext attack (cf. [BPR14]) and stateful subversions. Unlike public-key encryption, a subverted symmetric encryption algorithm has access to the secret key and can potentially leak it. Therefore, simply masking the plaintext cannot fully neutralize the attack. Interestingly, we can thwart leakage by using a random message to mask the output and appending the encryption of this random message so that the resulting ciphertext is decryptable.

The computational overhead of the proposed scheme is small. For encryption we basically need a reliable $\oplus$-operation, and for decrypting we need two calls to the decryption of the underlying scheme, and again a trustworthy implementation of $\oplus$. On the downside, we can only encrypt securely as long as a fresh sample is available. Moreover, the self-guarding decryption algorithm differs from the underlying decryption algorithm, and also we can only encrypt messages that are shorter than the sample messages.

4.1 Construction

Our construction $\mathcal{E}^{\text{sg}}$ from Figure 4 is built upon an arbitrary IND-CPA-secure encryption scheme $\mathcal{E}$ that has a maximum ciphertext expansion e. Here, the ciphertext expansion describes the maximum number of extra bits in the ciphertext to encrypt a message, e.g., to store a random IV.

In the sampling phase we generate multiple ciphertexts of random messages $m_{\$,i}$ of bit length ℓ. At this point, the encryption algorithm still complies with the specification, such that the samples are valid

```
$\mathcal{E}^{\text{sg}}.\mathsf{Gen}(1^\lambda)$
    $\mathsf{k} \xleftarrow{\$} \mathcal{E}.\mathsf{Gen}(1^\lambda)$
    return $\mathsf{k}$
```

```
$\mathcal{E}^{\text{sg}}.\mathsf{Sample}(\mathsf{k})$
    $\Omega \leftarrow []$
    for $i = 1..N$ do
        $m_{\$,i} \xleftarrow{\$} \{0,1\}^\ell$
        $c_{\$,i} \xleftarrow{\$} \mathcal{E}.\mathsf{Enc}(\mathsf{k}, m_{\$,i})$
        $\mathsf{enq}(\Omega, (m_{\$,i}, c_{\$,i}))$
    return $\Omega$
```

```
$\mathcal{E}^{\text{sg}}.\mathsf{Enc}(\mathsf{k}, \Omega, m)$
    $c \xleftarrow{\$} \mathcal{E}.\mathsf{Enc}(\mathsf{k}, m)$
    if $\mathsf{is\text{-}empty}(\Omega)$
        or $|m| > \ell - e - 1$
        or $|c| > \ell - 1$ then
      return $\perp$
    $(m_\$, c_\$) \leftarrow \mathsf{deq}(\Omega)$
    $c^{\text{sg}} \leftarrow [c||10^{\ell-|c|-1}] \oplus m_\$$
    return $(c^{\text{sg}}, c_\$)$
```

```
$\mathcal{E}^{\text{sg}}.\mathsf{Dec}(\mathsf{k}, (c^{\text{sg}}, c_\$))$
    $m_\$ \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{k}, c_\$)$
    $c||10\ldots0 \leftarrow c^{\text{sg}} \oplus m_\$$
    $m \leftarrow \mathcal{E}.\mathsf{Dec}(\mathsf{k}, c)$
    return $m$
```

Figure 4: Self-guarding symmetric encryption scheme $\mathcal{E}^{\text{sg}}$ built from a symmetric encryption scheme $\mathcal{E}$ with maximum ciphertext expansion of e bits for each message.

encryptions $c_{\$,i}$. Similar to the previous section, we store the samples in a queue structure, where we can access the queue with the usual enq and deq commands, and check if it is empty via $\mathsf{is\text{-}empty}$. The sample messages are used as a one-time-pad key to hide the ciphertext produced by a potentially malicious implementation. This is intuitively the reason why we need sample messages that are at least as long as the ciphertexts produced by $\mathcal{E}$. To deal with potentially shorter ciphertexts we use the common padding $10\ldots0$ to expand all ciphertexts to equal length. To make sure that the receiver is able to decrypt, the honest encryption of the sample message is sent along with the actual encryption.

In theory we are able to lift the restriction on the new message space by using a pseudorandom function to expand the sample messages. However we decided to keep the construction simple and minimize the number of trusted components.

Correctness. As long as fresh samples are available and the construction is not under a subversion attack, correctness of our symmetric encryption scheme $\mathcal{E}^{\text{sg}}$ for messages with maximum length of $\ell - e - 1$, follows immediately from correctness of the underlying symmetric encryption scheme $\mathcal{E}_{\text{GENUINE}}$. Since the encryption algorithm $\mathcal{E}^{\text{sg}}.\mathsf{Enc}$ essentially aborts if no more samples are left, correctness is not given afterwards.

4.2 Security

Next we prove that our construction is self-guarding with respect to the IND-CPA game, given that the underlying encryption scheme is IND-CPA during the initial trusted phase.

Theorem 4.1. *The symmetric encryption scheme $\mathcal{E}^{sg}$ from Figure 4 is self-guarding with respect to* IND-CPA*-security against subversion of $\mathcal{E}$, if $\mathcal{E}$ is a* IND-CPA*-secure symmetric encryption scheme.*

Proof. Consider an adversary $\mathcal{A}$ playing the self-guarding game $\mathsf{SGuard}^{\mathcal{E}^{\text{sg}\mathcal{E}},\beta}_{\text{IND-CPA},\mathcal{A}}$ defined in Figure 1. We again show that $\mathcal{A}$'s probability for predicting the secret bit b in the two settings, $\beta = \text{GENUINE}$ and $\beta = \text{SUBV}$, cannot differ significantly.

The case β = genuine. Consider first the case that the security game uses the actual scheme $\mathcal{E}_{\text{GENUINE}}$. Then $\mathcal{A}$'s probability of predicting b is negligible close to $\frac{1}{2}$, because the subverted algorithm $\mathcal{E}_{\text{SUBV}}$ is ignored, such that each in challenge query $m_{\text{left}}, m_{\text{right}}$ of $\mathcal{A}$ (of at most $\ell - e - 1$ bits) the adversary receives an encryption of m_{left} or of m_{right}, where the (padded) ciphertext is xored with a random message $m_\$$, which is at least as long as the ciphertext. The adversary also receives the genuine encryption of $m_\$$ (i.e., $c_\$$), derived via $\mathsf{Enc}_{\text{GENUINE}}$.

For the genuine encryption algorithm $\mathsf{Enc}_{\text{GENUINE}}$ we actually get a "twofold" secure encryption. First, and this suffices for the formal argument, the encryption of the challenge message under the IND-CPA-secure scheme $\mathsf{Enc}_{\text{GENUINE}}$ already hides the secret bit b. This can be straightforwardly formalized by simulating the extra layer of the encryption with $m_\$$ and creating the ciphertext $c_\$$ with the help of the encryption oracle in the IND-CPA game, also keeping track of the number of available samples. At the same time, we could also argue along the security of $c_\$$, hiding $m_\$$, which in turn is then used to mask the ciphertext. Hence, we can conclude that the probability of predicting b in the self-guarding game for $\mathsf{Enc}_{\text{GENUINE}}$ is negligibly close to $\frac{1}{2}$ by the IND-CPA-security of $\mathcal{E}$.

The case $\beta = \mathbf{subv}$. Next, consider the case that the security game uses the subverted algorithm $\mathcal{E}_{\text{SUBV}}$ in the challenge queries. In each such query for a pair of messages $m_{\text{left}}, m_{\text{right}}$ we hence encrypt the message under the subverted algorithm to get a possibly malicious ciphertext c. We check the validity of the length of the ciphertext, and then add the message $m_\$$ to the (padded) ciphertext to obtain c^{sg}. This result, together with the genuine encryption $c_\$ \xleftarrow{\$} \mathsf{Enc}_{\text{GENUINE}}(\mathsf{k}, m_\$)$, which was computed during the sampling phase, is output as the final ciphertext $(c^{\text{sg}}, c_\$)$.

Suppose now that, instead of encryption $m_\$$ under $\mathsf{Enc}_{\text{GENUINE}}$ for the second ciphertext component in each challenge query, we encrypt an independent random message $m'_\$ \xleftarrow{\$} \{0,1\}^\ell$. Then the adversary would only learn the encryption of an independent random message $m'_\$$, since c^{sg} is hidden by another random message $m_\$$. In other words, each challenge query yields an answer which is independently distributed from the bit b. The adversary's success probability for predicting b is then at most the guessing probability of $\frac{1}{2}$.

We are once more left to argue that encrypting $m'_\$$ instead of $m_\$$ does not significantly contribute to $\mathcal{A}$'s success probability. But this follows once more straightforwardly from the IND-CPA-security of $\mathcal{E}$. From adversary $\mathcal{A}$ we build an adversary $\mathcal{B}$ against $\mathcal{E}$. Adversary $\mathcal{A}$ first provides the subverted algorithm $\mathcal{E}_{\text{SUBV}}$. Then $\mathcal{B}$ simulates the rest of the self-guarding game, picking the bit $b \xleftarrow{\$} \{0,1\}$ itself, but playing against an external secret bit b' in its IND-CPA-game.

Adversary $\mathcal{B}$ answers each challenge query $m_{\text{left}}, m_{\text{right}}$ of $\mathcal{A}$ of length at most $\ell - e - 1$ as follows. Adversary $\mathcal{B}$ checks the length restrictions and that the number of samples is not exceeded yet. If so, $\mathcal{B}$ encrypts the message under $\mathsf{Enc}_{\text{SUBV}}$ and xors the padded result with a randomly chosen message $m_\$ \leftarrow \{0,1\}^\ell$ to get a ciphertext c^{sg}. It picks another random message $m'_\$$ of the same length and forwards $m_\$$ and $m'_\$$ to its own challenge oracle to get a ciphertext $c_\$$. It returns $(c^{\text{sg}}, c_\$)$ to adversary $\mathcal{A}$.

When $\mathcal{A}$ eventually outputs a guess for bit b, adversary $\mathcal{B}$ checks if the prediction is correct. If so, it outputs 0 for bit b', else it outputs 1.

The analysis is now identical to the case of the public-key scheme and omitted here. By assumption, our algorithm $\mathcal{B}$ therefore has a non-negligibly larger prediction probability than $\frac{1}{2}$. □

5 Self-Guarding Signatures

A substituted signing algorithm may try to leak information about the secret key, or a different signature. In the domain of reverse firewalls, the idea of Ateniese et al. [AMV15] is to have any signature created by the signer verified by the (trusted) reverse firewall with respect to the public key and, if correct, re-randomized before it is sent out.[1] The combination of re-randomization and verification of signatures prevents against subliminal leakage even when triggered by malicious inputs. For unique signatures, which

[1] Interestingly, Ateniese et al. [AMV15] define re-randomization with respect to the original signature algorithm, but the solution presumably requires re-randomization of maliciously generated signatures under the subverted algorithm.

have only one valid signature for each message under the public key, the re-randomization step is trivial and can be omitted.

It is possible to apply the same idea in our self-guarding setting, *if the verification step and the re-randomization step can be implemented robustly.* In this case, the signer generates a signature, verifies it with the trustworthy verification step, and re-randomizes it securely. This approach may be viable in some settings, e.g., when verifying FDH-RSA signatures with low exponents such as $e = 2^{16} + 1$, where only a few modular multiplications and, more critical, a hash evaluation would need to be carried out safely. Still, in other scenarios implementing the full verification procedure securely may be beyond the signer's capabilities, whereas storing a number of message and signature pairs reliably is usually a much easier task than correctly implementing cryptographic code. We therefore propose an alternative solution below.

5.1 Construction

The idea of our construction is as follows. We will use a regular deterministic signature scheme and consider only stateless subversions. In the initialization phase we sign a random message $m_\$$ under this scheme to get a signature sample $\sigma_\$$. We store this sample and then, if we are supposed to create a signature for a given message m later, then we will have the (now potentially substituted) signing algorithm create one signature for $m_\$$ and one for $m_\$ \oplus [m||\sigma_\$]$. Including the signature $\sigma_\$$ in the second message prevents combination attacks against unforgeability, and requires that $m_\$$ is long enough to range over $m||\sigma_\$$.[2]

We will hand over the two messages $m_\$$ and $m_\$ \oplus [m||\sigma_\$]$ in random order to the subverted signing algorithm such that if the algorithm deviates for one of the two signatures, we will detect this with probability $\frac{1}{2}$ and abort forever. Recall that we assume that the substituted algorithm is stateless such that both messages look equally random to it, even if we re-use the random message across multiple signature creations. To increase the detection probability to overwhelming we will repeat the above λ times with independent key pairs. The independence of the keys ensures that, even if the adversary manages to leak information about some signing keys, the other keys are still fresh. Although the trusted samples are used here to enforce security via detection, the detection strategy is very simple and avoids the problems with usual detectors discussed in Section 1.1.

More formally, our self-guarding signature scheme $\mathcal{S}^{\text{sg}} = (\mathsf{KGen}^{\text{sg}}, \mathsf{Sig}^{\text{sg}}, \mathsf{Vf}^{\text{sg}})$ is based on a deterministic signature scheme $\mathcal{S} = (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ and works as follows. The key generation algorithm $\mathsf{KGen}^{\text{sg}}(1^\lambda)$ creates λ key pairs $(\mathsf{sk}_i, \mathsf{pk}_i) \xleftarrow{\$} \mathcal{S}.\mathsf{KGen}(1^\lambda)$ of the underlying signature scheme. It sets $\mathsf{sk}^{\text{sg}} \leftarrow (\mathsf{sk}_1, \ldots, \mathsf{sk}_\lambda)$ and $\mathsf{pk}^{\text{sg}} \leftarrow (\mathsf{pk}_1, \ldots, \mathsf{pk}_\lambda)$ and outputs them together with a flag *err* initially set to false, indicating that no invalid signature was detected.

In the initialization phase we pick λ random messages $m_{\$,1}, \ldots, m_{\$,\lambda} \leftarrow \{0,1\}^\ell$ and create the signatures $\sigma_{\$,1}, \ldots, \sigma_{\$,\lambda}$ for all messages. We store the pairs $(m_{\$,i}, \sigma_{\$,i})$ in the sample queue Ω. The common bit length ℓ of the messages $m_{\$,i}$ determines an upper bound on the messages m which can later be signed. Namely, any message m can be at most the length of $m_{\$,i}$, minus the bit length for signatures, where we assume without loss of generality that all signatures are of equal length s. Reserving some space for the mandatory padding $10\cdots0$ of shorter messages, we must have that messages are of length $\ell - s - 1$ at most. Longer messages m may be hashed first, outside of the signing algorithm. For sake of a cleaner presentation we assume below that all input messages are tightly of length $\ell - s$. The proof can be transferred to the general case with padding easily.

Later, when signing a message m under the possibly subverted algorithm Sig, with the key sk^{sg} and the samples Ω, do the following. If no invalid signatures were detected so far, i.e., *err* is still false, for each $i = 1, 2, \ldots, \lambda$ pick a random bit $b_i \xleftarrow{\$} \{0,1\}$ and call the signing algorithm twice, one time for $\mathsf{sk}_i, m_{\$,i}$ and the other time for $\mathsf{sk}_i, m_{\$,i} \oplus [m||\sigma_{\$,i}]$. Use this order if $b_i = 0$ and the reverse order if $b_i = 1$. Let σ_i

[2] Note that deterministic signatures can produce shorter signatures, e.g., if first hashing the message; only the signature value for a given message must be deterministically computed.

be the returned signature for the message $m_{\$,i} \oplus [m||\sigma_{\$,i}]$. For each i check that the provided signature for $m_{\$,i}$ equals $\sigma_{\$,i}$. If not, abort after setting *err* to true. Else output $\sigma^{\mathrm{sg}} \leftarrow (m_{\$,1}, \sigma_{\$,1}, \sigma_1, \ldots, m_{\$,\lambda}, \sigma_{\$,\lambda}, \sigma_\lambda)$ as the signature.

Verification is straightforward. For each i build the message $m_{\$,i} \oplus [m||\sigma_{\$,i}]$ from the given message m and the data in the signature, and verify the signature σ_i with respect to pk_i, as well as the signature $\sigma_{\$,i}$ for $m_{\$,i}$. Accept iff all verification steps succeed.

```
$\mathcal{S}^{\mathrm{sg}}.\mathsf{Gen}(1^\lambda)$
for $i = 1..\lambda$ do
    $(\mathsf{sk}_i, \mathsf{pk}_i) \xleftarrow{\$} \mathcal{S}.\mathsf{KGen}(1^\lambda)$
$(\mathsf{sk}^{\mathrm{sg}}, \mathsf{pk}^{\mathrm{sg}}) \leftarrow ((\mathsf{sk}_1, \ldots, \mathsf{sk}_\lambda), (\mathsf{pk}_1, \ldots, \mathsf{pk}_\lambda))$
$err \leftarrow \mathsf{false}$
return $(\mathsf{sk}^{\mathrm{sg}}, \mathsf{pk}^{\mathrm{sg}}, err)$
```

```
$\mathcal{S}^{\mathrm{sg}}.\mathsf{Sample}(\mathsf{sk}^{\mathrm{sg}})$
$\Omega \leftarrow []$
for $i = 1..\lambda$ do
    $m_{\$,i} \xleftarrow{\$} \{0,1\}^\ell$
    $\sigma_{\$,i} \leftarrow \mathcal{S}.\mathsf{Sig}(\mathsf{sk}_i, m_{\$,i})$
    $\mathsf{enq}(\Omega, (m_{\$,i}, \sigma_{\$,i}))$
return $\Omega$
```

```
$\mathcal{S}^{\mathrm{sg}}.\mathsf{Sig}(\mathsf{sk}^{\mathrm{sg}}, m, \Omega, err)$
if $err = \mathsf{true}$ or $|m| \neq \ell - s$ then
    return $\perp$
$\Omega' \leftarrow \Omega$
for $i = 1..\lambda$ do
    $(m_{\$,i}, \sigma_{\$,i}) \leftarrow \mathsf{deq}(\Omega')$
    $b_i \xleftarrow{\$} \{0,1\}$
    if $b_i = 0$ then
        $(m^0, m^1) \leftarrow (m_{\$,i}, m_{\$,i} \oplus [m||\sigma_{\$,i}])$
    else
        $(m^0, m^1) \leftarrow (m_{\$,i} \oplus [m||\sigma_{\$,i}], m_{\$,i})$
    $\sigma^0 \leftarrow \mathcal{S}.\mathsf{Sig}(\mathsf{sk}_i, m^0)$
    $\sigma^1 \leftarrow \mathcal{S}.\mathsf{Sig}(\mathsf{sk}_i, m^1)$
    if $\sigma^{b_i} \neq \sigma_{\$,i}$ then
        $err \leftarrow \mathsf{true}$
        return $\perp$
    $\sigma_i \leftarrow \sigma^{1-b_i}$
    if $|\sigma_i| \neq s$ then return $\perp$
$\sigma \leftarrow (m_{\$,1}, \sigma_{\$,1}, \sigma_1, \ldots, m_{\$,\lambda}, \sigma_{\$,\lambda}, \sigma_\lambda)$
return $\sigma$
```

```
$\mathcal{S}^{\mathrm{sg}}.\mathsf{Vf}(\mathsf{pk}^{\mathrm{sg}}, m, \sigma)$
$\sigma = (m_{\$,1}, \sigma_{\$,1}, \sigma_1, \ldots, m_{\$,\lambda}, \sigma_{\$,\lambda}, \sigma_\lambda)$
$d \leftarrow [|m| = \ell - s]$
for $i = 1..\lambda$ do
    $d \leftarrow d \wedge |m_{\$,i}| = \ell \wedge |\sigma_{\$,i}| = s$
    $d \leftarrow d \wedge \mathcal{S}.\mathsf{Vf}(\mathsf{pk}_i, m_{\$,i}, \sigma_{\$,i})$
    $d \leftarrow d \wedge \mathcal{S}.\mathsf{Vf}(\mathsf{pk}_i, m_{\$,i} \oplus [m||\sigma_{\$,i}], \sigma_i)$
return $d$
```

Figure 5: Self-guarding signature scheme $\mathcal{S}^{\mathrm{sg}}$ with message space $\{0,1\}^{\ell-s}$ built from signature scheme $\mathcal{S}$ producing signatures of length s.

5.2 Security

For the security proof we need another property of the underlying signature scheme, namely, that the (equal length) signature strings are never zero-bitstrings. This can be easily achieved by prepending or

```
$\text{EUF-CMA}^{\mathcal{S}}_{\mathcal{A}}(\kappa, \Omega)$
$M \leftarrow \emptyset$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{Sig}(\kappa_s, \Omega, \cdot)}(\kappa_p)$
if $m^* \notin M$ and $\mathcal{S}.\mathsf{Vf}(\kappa_p, m^*, \sigma^*)$ then
    return true
return false

$\mathsf{Sig}(\kappa_s, \Omega, m)$
$M \leftarrow M \cup \{m\}$
$\sigma \xleftarrow{\$} \mathcal{S}.\mathsf{Sig}(\kappa_s, \Omega, m)$
return $\sigma$
```

Figure 6: EUF-CMA game for signature schemes $\mathcal{S}$.

appending a bit '1' to any signature and verifying that this bit really appears in the signature. We call such signature schemes *zero evading.*

The notion of EUF-CMA unforgeability is given by the standard security game in Figure 6. In the self-guarding game of Figure 1, if the adversary always chooses $\mathcal{S}_{\text{SUBV}} = \mathcal{S}_{\text{GENUINE}}$ and we have an empty list Ω we obtain the standard security notion without a substitution attack.

Theorem 5.1. *The signature scheme $\mathcal{S}^{sg}$ from Figure 5 is self-guarding with respect to EUF-CMA-unforgeability against stateless subversion of $\mathcal{S}$, if $\mathcal{S}$ is a deterministic, EUF-CMA-unforgeable, and zero-evading signature scheme.*

Proof. Consider an adversary $\mathcal{A}$ playing the self-guarding game. We can show that the advantage of $\mathcal{A}$ is negligible by the security of the underlying signature scheme, implying that $\mathcal{A}$ cannot increase its success probability noticeably with the help of substitutions.

In the attack, as well as in the reduction below, we denote the message in the j-th signature query by m_j. The i-th signature component in the j-th query for message $m_{\$,i} \oplus [m_j || \sigma_{\$,i}]$ is denoted as $\sigma_{i,j}$. We assume that $\mathcal{A}$ makes q signature queries. The forgery attempt is denoted by m^* and $(m^*_{\$,1}, \sigma^*_{\$,1}, \sigma^*_1, \ldots, m^*_{\$,\lambda}, \sigma^*_{\$,\lambda}, \sigma^*_\lambda)$.

Reduction to Signature Scheme. We construct an adversary $\mathcal{B}$ against the unforgeability of the underlying signature scheme $\mathcal{S}$ via a black-box reduction. Algorithm $\mathcal{B}$ receives as input a verification key pk. It first picks $k \xleftarrow{\$} \{1, 2, \ldots, \lambda\}$ at random and sets $\mathsf{pk}_k \leftarrow \mathsf{pk}$. It generates all the other key pairs $(\mathsf{sk}_i, \mathsf{pk}_i) \xleftarrow{\$} \mathcal{S}.\mathsf{KGen}(1^\lambda)$ for $i \neq k$ itself. Then $\mathcal{B}$ picks the messages $m_{\$,i}$ and creates the signatures $\sigma_{\$,i}$, for $i \neq k$ with the help of the signing key sk_i, and for $i = k$ by calling the signature oracle. It starts the attack of $\mathcal{A}$.

Whenever $\mathcal{B}$ is supposed to create a signature it executes the same steps as the self-guarding algorithm for any index $i \neq k$. In particular, it checks if the returned signature components for $m_{\$,i}$ match the previously sampled value. For the k-th index it uses the previously obtained oracle value $\sigma_{\$,k}$ and it now calls the external oracle to get a signature for $m_{\$,k} \oplus [m || \sigma_{\$,k}]$; it does not need to check these answers. Algorithm $\mathcal{B}$ uses all these data to assemble the signature in the same way our signing algorithm does.

If the adversary eventually outputs a forgery for message m^* and signature $(m^*_{\$,1}, \sigma^*_{\$,1}, \sigma^*_1, \ldots, m^*_{\$,\lambda}, \sigma^*_{\$,\lambda}, \sigma^*_\lambda)$, then $\mathcal{B}$ does the following:

- If $m^*_{\$,k} = m_{\$,k}$ then output the message $m^*_{\$,k} \oplus [m^* || \sigma^*_{\$,k}]$ together with σ^*_k as the signature.
- Else, if $m^*_{\$,k} \neq m_{\$,k} \oplus [m_j || \sigma_{\$,k}]$ for all $j = 1, 2, \ldots, q$, then output the message $m^*_{\$,k}$ with signature $\sigma^*_{\$,k}$.
- Else, if $m^*_{\$,k} = m_{\$,k} \oplus [m_j || \sigma_{\$,k}]$ for some j, then output the message $m^*_{\$,k} \oplus [m^* || \sigma^*_{\$,k}]$ with the signature σ^*_k.

Analysis. We first argue that the subverted signing algorithm, with overwhelming probability, must output only valid signatures for one of the keys pk_i in the actual attack. To this end, call the i-th entries $\sigma_{\$,i,j}$ and $\sigma_{i,j}$ in the j-th signature reply valid if they correspond to the signature for the messages under the specified signature algorithm. Then we claim that, with overwhelming probability, there must be some index i such that the signature entries in all queries are valid. In case of an abort, this refers to all signatures created up to the aborting query (exclusively), and else this refers to all queries.

Suppose that for each i the subverted algorithm outputs an invalid signature in some query j. This query may vary with the index i. Since the algorithm is stateless, the input messages $m_{\$,i,j}$ and message $m_{\$,i,j} \oplus [m_j||\sigma_{\$,i,j}]$ both look random to the algorithm. Furthermore, the order of the signing request is determined by a random bit b_i, such that the algorithm creates an invalid signature for the sampled message $m_{\$,i,j}$ with probability $\frac{1}{2}$. Hence, our self-guarding signature algorithm will detect this with probability $\frac{1}{2}$ and abort after setting *err* to true.

Any detected invalid signature will lead to an immediate abort and prohibits computing future signatures, and for each key pk_i the detection probability is independent of the other case. Hence, if the adversary tries to output an invalid signature in some query for any key pk_i, our algorithm will detect this, except with probability $2^{-\lambda}$. We can therefore from now on condition on the event that for some index i all signature entries in all queries are valid, losing only a probability $2^{-\lambda}$ in $\mathcal{A}$'s success probability.

If there is a good index i for which the subverted algorithm never outputs a wrong signature, then $\mathcal{B}$ picks this index $k = i$ with probability $\frac{1}{\lambda}$. Given this, all signatures created via the external oracle perfectly mimic the values returned by the subverted algorithm. From now on assume that this is the case. Since any values of invalid length will lead to an abort, we assume that the signature values are of correct length.

It remains to analyze the probability that, in a good simulation, adversary $\mathcal{B}$ creates a valid signature for a fresh message. Note that a valid forgery of $\mathcal{A}$ must be for a new message m^* and must consist of a vector of valid signatures, such that each component carries a valid signature. We distinguish the three cases for $\mathcal{A}$'s output as in the output generation of $\mathcal{B}$:

- If $m^*_{\$,k} = m_{\$,k}$ (and, by determinism, therefore $\sigma^*_{\$,k} = \sigma_{\$,k}$ for a valid signature), then we must have that $\mathcal{B}$'s output message satisfies

  $$m^*_{\$,k} \oplus [m^*||\sigma^*_{\$,k}] = m_{\$,k} \oplus [m^*||\sigma_{\$,k}] \neq m_{\$,k} \oplus [m_j||\sigma_{\$,k}]$$

  for all j, since $m^* \neq m_1, \ldots, m_q$ for a successful forgery of $\mathcal{A}$. Furthermore, since $\sigma^*_{\$,k} = \sigma_{\$,k} \neq 0$ by assumption about the zero-evasion of the signature scheme, $\mathcal{B}$'s output message cannot match $m_{\$,k}$ either. We conclude that $\mathcal{B}$ has never queried its signing oracle about this message, neither in the sampling phase, nor in a signing step. But since this message is checked against σ^*_k under pk_k, adversary $\mathcal{B}$ would also win if $\mathcal{A}$ does.

- Else, if $m^*_{\$,k} \neq m_{\$,k} \oplus [m_j||\sigma_{\$,k}]$ for all $j = 1, 2, \ldots, q$, then the message $m^*_{\$,k}$ is new; it is distinct from all queries in the signing step and also different from the query $m_{\$,k}$ in the sampling step, by the first case.

- Else, if $m^*_{\$,k} = m_{\$,k} \oplus [m_j||\sigma_{\$,k}]$ for some j, the adversary $\mathcal{A}$ has swapped this message part from the j-th query to the other signature position for the k-th entry. But in the signature verification one checks in the other component that the message

  $$m^*_{\$,k} \oplus [m^*||\sigma^*_{\$,k}] = m_{\$,k} \oplus [(m^* \oplus m_j)||(\sigma_{\$,k} \oplus \sigma^*_{\$,k})]$$

  is valid. Since $m^* \neq m_j$ this message cannot match $m_{\$,k}$ for which $\mathcal{B}$ has called the oracle for the sampling step. Moreover, zero evasion implies that $\sigma^*_{\$,k}$ is not zero, and therefore $\sigma_{\$,k} \oplus \sigma^*_{\$,k} \neq \sigma_{\$,k}$.

It follows that $\mathcal{B}$ has not called its oracle about the output message in any of the signing requests either.

In summary, we have

$$\Pr\left[\mathsf{SGuard}^{\mathcal{S}^{\mathrm{sg}\mathcal{S}},\textsc{subv}}_{\text{EUF-CMA},\mathcal{A}}(1^\lambda)\right] \leq \lambda \cdot \mathsf{Adv}^{\text{EUF-CMA}}_{\mathcal{S},\mathcal{B}}(1^\lambda) + \lambda \cdot 2^{-\lambda}.$$

This is negligible if we presume unforgeability of $\mathcal{S}$. □

6 PUF-based Key Exchange

In a key-exchange protocol two parties interact to derive a shared secret key, which they can use subsequently for example for establishing a confidential channel. The interesting aspect of using PUFs for such protocols is that one can achieve information-theoretic security, when the PUF is ideal and one limits the time during which the adversary has access to the PUF. There have been some proposals for PUF-based key-exchange protocols in this line [VD10, Rüh11, BFSK11, vDR12]. These schemes do not withstand substitution attacks as we briefly exemplify for the case of [vDR12] in Appendix A.1.

6.1 Preliminaries

A physically unclonable function (PUF) is a physical entity that is easy to evaluate, if one is in possession of the PUF, and hard to predict otherwise. A PUF can be stimulated with so-called challenges to which it responds with slightly noisy values, called responses. A fuzzy extractor can be applied to the response to eliminate the noise. Hence, in our setting, we assume that PUFs (with a suitable fuzzy extractor) deterministically return consistent answers. Moreover, we only consider PUFs that have exponential challenge and response spaces and hence cannot be learned entirely in a short time. In fact, we assume that the PUF has super-logarithmic input bit size and output size 5λ. If we have a PUF with only λ output bits, then we can expand the output size via domain separation, and evaluate the PUF at points $000||x, \ldots, 100||x$, and concatenates the responses.

Due to uncontrollable variations in the manufacturing process, it is even for the manufacturer practically infeasible to clone a PUF. This property is referred to as *unclonability.* In our scenario we assume an initialization procedure, $\mathsf{create}()$, which creates a new PUF and returns a unique PUF identifier pid. We denote the concrete PUF then as $\mathsf{PUF}_{\mathrm{pid}}$, or following our subversion notations, as $\mathsf{PUF}_{\textsc{genuine}}$.

Only the party (including the adversary) in possession of the PUF (identifier) can evaluate the PUF. Besides the parties, another PUF may be "in possession" of the PUF, called encapsulated PUFs [BKOV17] or PUF-inside-PUFs [Rüh16]. This outer PUF may then exclusively evaluate the inner PUF. The possibility to encapsulate PUFs allows for example to bypass simple checks, such as challenge-response validation, before evaluating the PUF on the actual data; a malicious PUF may switch only to a skewed mode after the checks. We therefore also allow create to be called with a malicious algorithm A in which case the PUF evaluates A on the input, or with an algorithm A and previously created PUF identifiers $\mathrm{pid}_1, \ldots, \mathrm{pid}_n$ in which case algorithm A may also call the PUFs with these identifiers as subroutine. We consequently sometimes refer to such a malicious PUF as $\mathsf{PUF}_{\textsc{subv}}$. We say that a PUF pid' encapsulates a PUF pid if pid' has been created by including pid.

PUFs can have various properties that make them attractive for cryptographic schemes. A property that we take advantage of is *pseudorandomness* of PUF responses [AMSY16]. This means that the PUF approximates a random function. In some works PUFs are also treated as random functions per se, but we prove the result to hold more generally also for (computationally) pseudorandom functions. In Figure 7, we give a simplified and intuitive game for pseudorandomness that suffices for this paper. Note that the uncloneability is basically ensured by allowing the adversary to internally create further PUFs. The PUF is pseudorandom if the probability of predicting the bit b is negligibly close to $\frac{1}{2}$ in the game. A hybrid argument implies that the same is true if we use n challenge-response values instead of only one, where the advantage over $\frac{1}{2}$ grows by the factor n.

```
$\mathrm{IND}_{\mathcal{A}}^{\mathsf{PUF}}(1^\lambda)$
$\mathrm{pid} \leftarrow \mathsf{create}()$
$(\mathsf{done}, \mathsf{st}) \leftarrow \mathcal{A}^{\mathsf{PUF}_{\mathrm{pid}}, \mathsf{create}}(1^\lambda)$
$c \xleftarrow{\$} \mathcal{C}$
$r_0 \leftarrow \mathsf{PUF}_{\mathrm{pid}}(c);\ r_1 \xleftarrow{\$} \{0,1\}^{|r_0|}$
$b \xleftarrow{\$} \{0,1\}$
$b' \xleftarrow{\$} \mathcal{A}^{\mathsf{create}}(\mathsf{st}, c, r_b)$
return $(b = b')$
```

Figure 7: Game for pseudorandomness of PUF responses.

For a more comprehensive and formal definition of PUFs and their security properties we refer to [BFSK11] and [AMSY16].

6.2 Construction

In a simple PUF-based key-exchange protocol, Alice measures the PUF at a random challenge point and sends her PUF to Bob. After assuring that Bob has received the PUF, she sends him the challenge through an authenticated channel.[3] Both parties use the PUF's response on this challenge as their shared secret key. An adversary, not yet knowing the challenge when getting access during transmission of the PUF, may measure the PUF for at most polynomial many challenges. Then the adversary delivers the PUF to the other party, therefore loses access, and only then learns the challenge used by the parties. With high probability this challenge will not be among the ones used by the adversary before, implying that the derived key looks random to the adversary. Instead of sending a fresh PUF for each key derivation, the PUF may also be used multiple times. We denote the number of derived keys by N.

Considering that the physical channel used for transmitting the PUF may not be authenticated, the adversary is now not only able to measure the PUF, but also replace it with a malicious one, potentially even encapsulating the original PUF into the malicious one, e.g., send $\mathsf{PUF}_{\mathrm{pid}^*}$ for $\mathrm{pid}^* \leftarrow \mathsf{create}(A, \mathrm{pid})$. Even with an authenticated physical channel, a more powerful adversary may be able to gain physical access to the PUF while it is in control of one of the parties for a short time, just enough to replace it.

Motivated by the above attack, we draw connections to algorithm-substitution attacks, which in this scenario can be more accurately described as token-substitution attacks. In Figure 8 we propose a PUF-based key-exchange protocol that self-guards against subversion of the PUF. It intuitively does so by splitting the initially derived key y into a test part and an evaluation part. This splitting is done via universal hash functions h_{univ}, h'_{univ}, where $h_{\mathrm{univ}}(y)$ and $h'_{\mathrm{univ}}(y)$ act as authentication codes of the key (towards Bob resp. towards Alice), and an extractor h_{extr} which is used to extract sufficiently many random bits $h_{\mathrm{extr}}(y)$ from the remaining bits. The sending party transmits $h_{\mathrm{univ}}, h'_{\mathrm{univ}}, h_{\mathrm{extr}}$, and $h_{\mathrm{univ}}(y)$ over the authenticated channel, and the receiving party checks that the authentication part matches its initially derived key. The receiver replies with its authentication tag.

[3] See [BFSK11] for a discussion that an authenticated digital channel is necessary for reasonable protocols.

In the protocol we denote by $\mathcal{H}_{5\lambda,\lambda}[p]$ a family of hash functions with input bit size 5λ and output bit size λ, having some property p. Here, p is either being $2^{-\lambda}$-universal, saying that for fixed $x \neq x' \in \{0,1\}^{5\lambda}$ we have $h_{\text{univ}}(x) = h_{\text{univ}}(x')$ with probability at most $2^{-\lambda}$ over the choice of h_{univ} from the family. Similarly, for property p being a $(3\lambda, 2^{-\lambda})$-extractor we have that $(h_{\text{extr}}, h_{\text{extr}}(y))$ has statistical distance $2^{-\lambda}$ from (h_{extr}, z) for uniform $z \in \{0,1\}^{\lambda}$, as long as y has min-entropy at least 3λ. Since the loss of at most 2λ bits through the authentication tags $h_{\text{univ}}(y)$ and $h'_{\text{univ}}(y)$, we still have this min-entropy left in the uniform value y.

Our result holds with respect to malicious, stateful, and encapsulated PUFs. This is not subsumed by any of the previous results, nor does it contradict any of the impossibility results so far. As for positive results note that the oblivious transfer protocol of Brzuska et al. [BFSK11], in the version of Dachman-Soled et al. [DFK+14], from which one could build a key exchange protocol, does not withstand encapsulating and stateful PUFs. The impossibility result to build key exchange protocols by van Dijk and Rührmair [vDR12] only applies to PUFs which are accessible by the adversary after the execution, a property which we do not consider here.

6.3 Security

For our security claim we need to specify the security game. Since we assume authenticated digital transmissions, the adversary may read but not tamper with the transmissions (beyond replacing the PUF). We are interested in key confidentiality, namely, that the adversary cannot distinguish keys from random, and robustness in the sense that, if both parties accept, then they also hold the same key. In the security game we therefore give the adversary a transcript of a run of the key exchange protocol (where the adversary may have replaced the PUF before, however), and hand over the N keys derived by one party, or random values instead. The choice is made according to some secret bit b. We declare the adversary to win if it either manages to predict b, or to make both parties accept with different keys (in which case we hand over b, unifying the threshold to the guessing probability of $\frac{1}{2}$ for both cases).

The game is formally described in Figure 9. Note that this game is needed in the second phase of the self-guarding game. In the first phase, one creates the PUF, then possibly samples challenge and responses, while in the second phase the adversary may subvert the PUF and then plays the robust key indistinguishability game for a key-exchange protocol using either the original or the subverted PUF.

Theorem 6.1. *Our key-exchange protocol $\mathcal{KE}^{sg}$ from Figure 8 is self-guarding with respect to the robust key indistinguishability game* IND-KEY*, against subversion of* PUF*, if the initial* PUF *is pseudorandom.*

Proof. Consider an adversary $\mathcal{A}$ playing the self-guarding game $\mathsf{SGuard}^{\mathcal{KE}^{\text{sg}\mathsf{PUF}},\beta}_{\mathsf{IND\text{-}KEY},\mathcal{A}}$ defined in Figure 1. We argue that $\mathcal{A}$'s success probability in distinguishing keys, established by the protocol, from random strings is negligible, regardless of which value β takes. We let q denote the number of queries which $\mathcal{A}$ makes to the original $\mathsf{PUF}_{\text{GENUINE}}$ itself.

We first note that, instead of using a pseudorandom $\mathsf{PUF}_{\text{GENUINE}}$, we may equally well use a truly random PUF. If this would decrease $\mathcal{A}$'s success probability significantly, then we would immediately derive a contradiction to the pseudorandomness of the PUF.

Next, we argue that, if the adversary does not encapsulate $\mathsf{PUF}_{\text{GENUINE}}$ in $\mathsf{PUF}_{\text{SUBV}}$ then, except with negligible probability, neither party will accept in any of the N runs. To see this note that the probability that $\mathcal{A}$ queries the PUF $\mathsf{PUF}_{\text{GENUINE}}$ on any of the N challenge values x, before *some* $\mathsf{PUF}_{\text{SUBV}}$ is delivered to Bob, is at most $Nq \cdot 2^{-\lambda}$. Condition now on the event that such a query has not happened.

In the moment when $\mathsf{PUF}_{\text{SUBV}}$ is handed over, and by the authenticated *"ok"*-acknowledgement sent by Bob this happens before the adversary gets to learn the challenges, each value $y \leftarrow \mathsf{PUF}_{\text{GENUINE}}(x)$ is distributed independently of the function in $\mathsf{PUF}_{\text{SUBV}}$. Here we use that $\mathsf{PUF}_{\text{SUBV}}$ does not encapsulate $\mathsf{PUF}_{\text{GENUINE}}$. Hence, for each challenge, except with probability $2^{-\lambda}$, the random response y is different from

```
$\mathcal{KE}^{\text{sg}}.\mathsf{Sample}(\text{pid})$
$\Omega \leftarrow []$
for $i = 1..N$ do
    $x_{\$,i} \xleftarrow{\$} \mathcal{C}$
    $y_{\$,i} \leftarrow \mathsf{PUF}_{\text{pid}}(x_{\$,i})$
    $\mathsf{enq}(\Omega, (x_{\$,i}, y_{\$,i}))$
return $\Omega$
```

```
$\mathcal{KE}^{\text{sg}}.\mathsf{Gen}(1^{\lambda})$
$\text{pid} \leftarrow \mathsf{create}()$
    where $\mathsf{PUF}_{\text{pid}} : \{0,1\}^{\lambda} \rightarrow \{0,1\}^{5\lambda}$
return pid
```

```
$\mathcal{KE}^{\text{sg}}.\mathsf{KE}(\text{pid}, \Omega)$
Alice$(\text{pid}, \Omega)$                                                  Bob
                              $\xrightarrow{\mathsf{PUF}_{\text{pid}}}$ (dashed)
                              $\xleftarrow{\text{“ok”}}$
                              for $i = 1..N$ do
$(x, y) \leftarrow \mathsf{deq}(\Omega)$
$h_{\text{univ}} \xleftarrow{\$} \mathcal{H}_{5\lambda,\lambda}[2^{-\lambda}\text{-universal}]$
$h'_{\text{univ}} \xleftarrow{\$} \mathcal{H}_{5\lambda,\lambda}[2^{-\lambda}\text{-universal}]$
$h_{\text{extr}} \xleftarrow{\$} \mathcal{H}_{5\lambda,\lambda}[(3\lambda, 2^{-\lambda})\text{-extractor}]$
$a \leftarrow h_{\text{univ}}(y)$
                              $\xrightarrow{x, h_{\text{univ}}, h'_{\text{univ}}, a, h_{\text{extr}}}$
                                                                            $y \leftarrow \mathsf{PUF}_{\text{pid}}(x)$
                                                                            if $a \neq h_{\text{univ}}(y)$ then abort
                                                                            $a' \leftarrow h'_{\text{univ}}(y)$
                              $\xleftarrow{a'}$
if $a' \neq h'_{\text{univ}}(y)$ then abort
$\mathsf{k} \leftarrow h_{\text{extr}}(y)$                                   $\mathsf{k} \leftarrow h_{\text{extr}}(y)$
```

Figure 8: Self-guarding PUF-based key-exchange protocol $\mathcal{KE}^{\text{sg}}$, where PUF has challenge and response space $\mathcal{C} = \{0,1\}^{\lambda}$ and $\mathcal{R} = \{0,1\}^{5\lambda}$. Solid arrows denote authenticated digital transmissions, while the dashed arrow denotes a physical transmission.

```
$\mathsf{IND\text{-}KEY}^{\mathcal{KE}}_{\mathcal{A}}(\kappa, \Omega)$
$(\mathsf{k}^0_1, \ldots, \mathsf{k}^0_N, \mathsf{transc}) \xleftarrow{\$} \mathcal{KE}.\mathsf{KE}(\kappa, \Omega)$
$b \xleftarrow{\$} \{0, 1\}$
if $\mathsf{k}^0_{i,\text{Alice}} \neq \mathsf{k}^0_{i,\text{Bob}}$ and $\mathsf{k}^0_{i,\text{Alice}}, \mathsf{k}^0_{i,\text{Bob}} \neq \bot$ for some $i$ then $a \leftarrow b$ else $a \leftarrow \bot$ fi
$(\mathsf{k}^1_1, \ldots, \mathsf{k}^1_N) \xleftarrow{\$} \mathcal{K}^N$
$b' \leftarrow \mathcal{A}(\kappa_p, \mathsf{k}^b_1, \ldots, \mathsf{k}^b_N, \mathsf{transc}, a)$
return $(b = b')$
```

Figure 9: $\mathsf{IND\text{-}KEY}$ game with associated constant $\delta = \frac{1}{2}$ for key-exchange protocols $\mathcal{KE}$. Here $\mathsf{k}^0_i = \mathsf{k}^0_{i,\text{Alice}}$ and $\mathsf{k}^0_{i,\text{Bob}}$ denote the key output in the i-th execution by Alice and Bob, respectively. We assume that keys are set to $\bot$ for non-accepting executions; transc denotes the communication between by all parties in all executions; $\mathcal{K}$ denotes the key space.

the response y' computed by $\mathsf{PUF}_{\textsc{subv}}$. In this case, with probability at most $2 \cdot 2^{-\lambda}$ by the property of the universal hash functions $h_{\text{univ}}, h'_{\text{univ}}$ (also chosen independently of y, y'), either of the authentication tags a or a' complies with the expected answer. Summing over all N challenges implies that only with negligible probability $\mathcal{A}$ can afford to not encapsulate $\mathsf{PUF}_{\textsc{genuine}}$ and still make either party accept in any execution.

If, on the other hand, $\mathsf{PUF}_{\textsc{subv}}$ encapsulates $\mathsf{PUF}_{\textsc{genuine}}$, then the adversary cannot determine any of the random value $y \leftarrow \mathsf{PUF}_{\textsc{genuine}}(x)$, since we have already ruled out that it has queried x before. Since the value contains 5λ bits of min-entropy, and we lose at most 2λ bits through the two hash values a, a', the extractor ensures that k is $2^{-\lambda}$ close to uniform, given h_{extr}. The statistical distance of all N independent samples is then given by $N \cdot 2^{-\lambda}$.

The same line of reasoning for the case that the substituted PUF does not encapsulate the original one, shows that the adversary cannot make the parties accept but for different keys, except with negligible probability. For this note that they can only derive distinct keys if they end up with different values $y \neq y'$. Here, the universality of h_{univ} and h'_{univ} and the fact that the responses are determined independently of the choice of the hash function again imply that the probability of such a collision with the expected value a or a' is at most $2 \cdot 2^{-\lambda}$.

In conclusion, we obtain that the success probability of any adversary $\mathcal{A}$ in winning $\mathsf{IND\text{-}KEY}$ against $\mathcal{KE}^{\text{sg}}$ is negligibly close to $\frac{1}{2}$. □

Note that the hashing steps are crucial for security. If one would, say, simply divide y into strings $a||a'||\mathsf{k}$ of lengths λ, λ and 3λ, respectively, then the adversary could send an encapsulated PUF which agrees upon the first 2λ bits but returns a different part k. In this case both parties would accept, but with distinct keys. Even worse, Bob's key part k may be easy to predict for the adversary.

7 Conclusion

Our results show that basic tasks can be made self-guarding. Protection against ASAs is a challenging task. Currently, the biggest concern is to improve the efficiency of constructions. For our self-guarding public-key and symmetric encryption schemes it is less the computational overhead, but rather that one can only perform secure encryption as long as fresh samples are still available. Recall that involved techniques such as sample re-randomization, which quasi means to implement one's own encryption procedure, should be avoided. Thus, a viable option may be to consider restricted subversion attacks, such as stateless algorithms Σ_{SUBV}. One can also study the possibility of reusing samples after each system reboot to protect against stateful subversions that can only use a volatile memory to store their states.

The idea of relaxing the admissible attack strategy works for our self-guarding signature scheme. It can be applied an unbounded number of times for stateless subverted algorithms. Also, it requires many calls to the signature algorithm and produces large signatures. Here, using specific signature schemes may be helpful in overcoming these limitations.

In terms of efficiency, our self-guarding PUF-based key exchange protocol is reasonably fast. It remains an interesting open question if other PUF-based protocols, e.g., for oblivious transfer (OT), can be self-guarded. As for negative results, Rührmair [Rüh16] argues that the strategy of interleaving test and evaluation challenges fails for the oblivious transfer protocol of Dachman-Soled et al. [DFK$^+$14]. But this attack is based on the specific oblivious transfer protocol where the adversary has some control over the input to the PUFs. An option may be to use a different OT protocol where the adversary has less influence on the inputs fed into the PUF.

Acknowledgments

This work has been supported by the German Federal Ministry of Education and Research (BMBF) as well as by the Hessian State Ministry for Higher Education, Research and the Arts, within CRISP.

References

[AMSY16] Frederik Armknecht, Daisuke Moriyama, Ahmad-Reza Sadeghi, and Moti Yung. Towards a unified security model for physically unclonable functions. In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 271–287, San Francisco, CA, USA, February 29 – March 4, 2016. Springer, Heidelberg, Germany. (Cited on pages 18 and 19.)

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 364–375, Denver, CO, USA, October 12–16, 2015. ACM Press. (Cited on pages 5, 8, and 13.)

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 51–70, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. (Cited on pages 18, 19, and 20.)

[BH15] Mihir Bellare and Viet Tung Hoang. Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 627–656, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. (Cited on page 2.)

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15: 22nd Conference on Computer and Communications Security*, pages 1431–1440, Denver, CO, USA, October 12–16, 2015. ACM Press. (Cited on pages 1 and 2.)

[BK89] Manuel Blum and Sampath Kannan. Designing programs that check their work. In *21st Annual ACM Symposium on Theory of Computing*, pages 86–97, Seattle, WA, USA, May 15–17, 1989. ACM Press. (Cited on page 3.)

[BKOV17] Saikrishna Badrinarayanan, Dakshita Khurana, Rafail Ostrovsky, and Ivan Visconti. Unconditional UC-secure computation with (stronger-malicious) PUFs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 382–411, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany. (Cited on pages 4 and 18.)

[BLR90] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *22nd Annual ACM Symposium on Theory of Computing*, pages 73–83, Baltimore, MD, USA, May 14–16, 1990. ACM Press. (Cited on page 3.)

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 1–19, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on pages 1, 2, 8, and 11.)

[CDL17] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted TPMs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 427–461, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. (Cited on page 2.)

[CMG$^+$16] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. A systematic analysis of the juniper dual EC incident. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 468–479, Vienna, Austria, October 24–28, 2016. ACM Press. (Cited on page 2.)

[DFK$^+$14] Dana Dachman-Soled, Nils Fleischhacker, Jonathan Katz, Anna Lysyanskaya, and Dominique Schröder. Feasibility and infeasibility of secure computation with malicious PUFs. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 405–420, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on pages 5, 20, and 23.)

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 579–598, Istanbul, Turkey, March 8–11, 2015. Springer, Heidelberg, Germany. (Cited on pages 1, 2, and 8.)

[DFS16] Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private circuits III: Hardware trojan-resilience via testing amplification. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 16: 23rd Conference on Computer and Communications Security*, pages 142–153, Vienna, Austria, October 24–28, 2016. ACM Press. (Cited on page 2.)

[DGG$^+$15] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc

Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 101–126, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. (Cited on page 3.)

[DMSD16] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 341–372, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. (Cited on page 3.)

[EPS15] C. Evans, C. Palmer, and R. Sleevi. Public key pinning extension for HTTP, April 2015. RFC 7469. (Cited on page 4.)

[HH09] Iftach Haitner and Thomas Holenstein. On the (im)possibility of key dependent encryption. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 202–219. Springer, Heidelberg, Germany, March 15–17, 2009. (Cited on page 6.)

[MS15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 657–686, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany. (Cited on page 3.)

[OSVW13] Rafail Ostrovsky, Alessandra Scafuro, Ivan Visconti, and Akshay Wadia. Universally composable secure computation with (malicious) physically uncloneable functions. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 702–718, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany. (Cited on page 4.)

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 34–64, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany. (Cited on pages 1, 3, 5, 6, and 8.)

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17: 24th Conference on Computer and Communications Security*, pages 907–922, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. (Cited on pages 3, 4, and 5.)

[Rüh11] Ulrich Rührmair. Physical turing machines and the formalization of physical cryptography. Cryptology ePrint Archive, Report 2011/188, 2011. http://eprint.iacr.org/2011/188. (Cited on page 18.)

[Rüh16] Ulrich Rührmair. On the security of PUF protocols under bad PUFs and PUFs-inside-PUFs attacks. Cryptology ePrint Archive, Report 2016/322, 2016. http://eprint.iacr.org/2016/322. (Cited on pages 4, 5, 18, and 23.)

[VD10] M.E. Van Dijk. System and method of reliable forward secret key sharing with physical random functions, January 26 2010. US Patent 7,653,197. (Cited on page 18.)

[vDR12] Marten van Dijk and Ulrich Rührmair. Physical unclonable functions in cryptographic protocols: Security proofs and impossibility results. Cryptology ePrint Archive, Report 2012/228, 2012. http://eprint.iacr.org/2012/228. (Cited on pages 5, 18, 20, and 26.)

[WS11] Adam Waksman and Simha Sethumadhavan. Silencing hardware backdoors. In *2011 IEEE Symposium on Security and Privacy*, pages 49–63, Berkeley, CA, USA, May 22–25, 2011. IEEE Computer Society Press. (Cited on page 2.)

[YY96] Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 89–103, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany. (Cited on page 1.)

[YY97] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 62–74, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany. (Cited on page 1.)

A Supplementary Material

A.1 Example Attack on PUF-based Key Exchange Protocol

Here we briefly argue that the common technique of checking validity of the PUF by verifying a challenge-response pair is vulnerable to substitution attacks. The derived PUF is stateless and encapsulates the original PUF. We describe the attack on the concrete protocol in [vDR12].

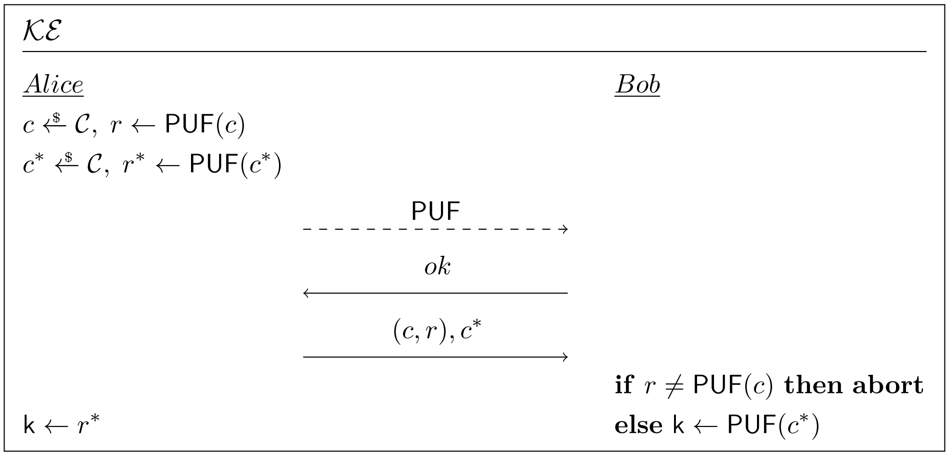
$\mathcal{KE}$
Alice
Bob
$c \xleftarrow{\$} \mathcal{C},\ r \leftarrow \mathsf{PUF}(c)$
$c^* \xleftarrow{\$} \mathcal{C},\ r^* \leftarrow \mathsf{PUF}(c^*)$
PUF
ok
$(c, r), c^*$
if $r \neq \mathsf{PUF}(c)$ **then abort**
$\mathsf{k} \leftarrow r^*$
else $\mathsf{k} \leftarrow \mathsf{PUF}(c^*)$

Figure 10: PUF-based key-exchange protocol from [vDR12], vulnerable to PUF-substitution attack.

The attacker builds the substituted PUF by encapsulating the original PUF. When stimulated, the malicious PUF flips a coin and either returns the original response, or the all-zero string. Then, with probability $\frac{1}{4}$ it will pass the check *and* make Bob output the all-zero key, thus breaking both robustness and key indistinguishability. This process can be derandomized by using a 2-wise independent hash function h outputting a single bit, returning the original response on challenge c if $h(c) = 1$, and the zero string if $h(c) = 0$.

A.2 Security Degradation

Roughly speaking, self-guarding of a scheme states that a scheme Π does not become insecure if the underlying scheme Σ is substituted by a malicious implementation. However, it does not guarantee that the security is not degraded. Consider for instance a substitution attack that manages to leak half of the secret key to the adversary. Although the advantage from Definition 2.2 with respect to a full key recovery game may remain negligible, the success probability grows exponentially. For scenarios, where one needs to quantify an adversary's gain more precisely, we suggest using the security degradation factor defined below.

Definition A.1 (Security degradation under subversion)**.** *Let Σ and Π be cryptographic schemes, and let Sec be a security game for Π with associated constant $\delta \in [0, 1)$ (used in defining the advantage). The* security degradation factor *for an adversary $\mathcal{A}$ in the subversion game of Figure 1 is defined by:*

$$\Theta^{\mathsf{SGuard},\mathsf{Sec}}_{\Pi^\Sigma,\mathcal{A}}(\lambda) := \frac{\Pr\left[\mathsf{SGuard}^{\Pi^\Sigma,\textsc{subv}}_{\mathsf{Sec},\mathcal{A}}(1^\lambda) = \mathsf{true}\right] - \delta}{\Pr\left[\mathsf{SGuard}^{\Pi^\Sigma,\textsc{genuine}}_{\mathsf{Sec},\mathcal{A}}(1^\lambda) = \mathsf{true}\right] - \delta}.$$

We say that subverting Σ does not substantially degrade security of Π with respect to Sec, if for all PPT adversaries $\mathcal{A}$, we have $\Theta^{\mathsf{SGuard},\mathsf{Sec}}_{\Pi^\Sigma,\mathcal{A}}(\lambda) \leq \mathsf{poly}(\lambda)$.