Cliptography: Clipping the Power of Kleptographic Attacks

Alexander Russell[*] Qiang Tang[†] Moti Yung[‡] Hong-Sheng Zhou[§]

August 17, 2015

Abstract

Kleptography, originally introduced by Young and Yung [Crypto '96], studies how to steal information securely and subliminally from cryptosystems. The basic framework considers the (in)security of malicious implementations of a standard cryptographic primitive by embedding a "backdoor" into the system. Remarkably, crippling subliminal theft is possible even if the subverted cryptosystem produces output indistinguishable from a secure "reference implementation." After a long hiatus, interest in such issues was rekindled by the dramatic revelations of Edward Snowden, demonstrating that such deliberate attacks have been deployed and presumably used for massive surveillance. Notably, Bellare, Paterson, and Rogaway [Crypto '14] initiated a formal study of attacks on symmetric key encryption algorithms.

Motivated by the original examples of subverting key generation algorithms in the kleptography papers from Young and Yung [Crypto '96, Eurocrypt '97], we initiate the study of cryptography in the setting where *all* algorithms are subject to kleptographic attacks—we call this **cliptography**. As a first step, we formally study the fundamental primitives of one-way function and trapdoor one-way function in this "complete subversion" model. We describe a general, rigorous immunization strategy to clip the power of kleptographic subversions; concretely, we propose a general framework for sanitizing (trapdoor) one-way function index generation algorithms by hashing the function index, and prove that such a procedure indeed destroys the connection between a subverted function generation procedure and any possible backdoor. Along the way, we propose a split program model for practical deployment.

We then examine two standard applications of (trapdoor) one way functions in this complete subversion model. First, we consider construction of "higher level" primitives via black-box reductions. In particular, we show how to use our trapdoor one-way function to defend against key generation sabotage, and showcase a digital signature scheme that preserves existential unforgeability when *all* algorithms (including key generation, which was not considered to be under attack before) are subject to kleptographic attacks. Additionally, we demonstrate that the classic Blum–Micali pseudorandom generator (PRG), using our "unforgeable" one-way function, yields a backdoor-free PRG. Second, we generalize our immunizing technique to one way functions, and propose a new public immunization strategy to randomize the public parameters of a (backdoored) PRG. This notably contrasts with previous results of Dodis, Ganesh, Golovnev, Juels, and Ristenpart [Eurocrypt '15], which require an honestly generated random key.

Thus, we develop fundamental cryptographic primitives with meaningful security guarantees in a quite adversarial setting, where one cannot rely on private randomness and all associated algorithms, including key and index generation, are under attack.

[*]University of Connecticut, acr@cse.uconn.edu
[†]University of Connecticut, qiang@cse.uconn.edu
[‡]Columbia University and Google, moti@cs.columbia.edu
[§]Virginia Commonwealth University, hszhou@vcu.edu

Contents

1 Introduction

Consider the conventional use of a cryptographic primitive, such as an encryption scheme: To encrypt a desired plaintext, the encryptor simply runs an implementation of the encryption algorithm obtained from a hardware or software provider with the plaintext as input. Although the underlying algorithms may be well-studied and proven secure, malicious implementations could still leak secret information exclusively to the provider/manufacturer without being noticed (through a covert channel that relies on an embedded backdoor). It is notable that *such leakage is possible even if the implementation produces "functionally and statistically clean" output that is indistinguishable from that of a faithful implementation*. While the underlying concept of kleptography was proposed by Young and Yung two decades ago [21,22], the recent Snowden revelations [14,18] provided striking real-world examples that awakened the security community to the seriousness of these issues. As a result, the topic has recently received renewed attention; see, e.g., [2,3,8,16]. In particular, Bellare, Paterson, and Rogaway [3][1] studied algorithm substitution attacks, with focus on symmetric key encryption. Soon after, Dodis, Ganesh, Golovnev, Juels, and Ristenpart [8] studied pseudorandom generators (PRG) in this backdoored setting.

1.1 Our contribution

We continue this line of pursuit. Specifically, we are motivated to develop cryptographic schemes in a *complete subversion model*, in which *all* algorithms of a scheme are potentially subverted by the adversary. This model thus significantly generalizes previously studied settings, which rely on trusted key generation or clean randomness that is assumed private from the adversary. We study two fundamental cryptographic primitives in the complete subversion model—one-way functions and one-way trapdoor funtions—and apply these primitives to construct other cryptographic schemes such as digital signatures and PRGs. Along the way, we identify novel generic defending strategies. We intend to stimulate a systematic study of **cliptography**, to provide a broader class of cryptographic building blocks and a larger set of defending strategies, eventually clipping out potential kleptographic attacks that arise from maliciously implemented components. As mentioned above, prior to our work kleptographic attacks on various primitives have been addressed in weaker models; see *Related work* in Section 1.2. In detail, we show the following:

- We study (trapdoor) one-way functions in the presence of kleptographic attacks. We first introduce a notion of *strong forgeability* that captures natural kleptographic attacks, namely: there is a specification that is proven secure for a (trapdoor) one-way function; the sabotaged function generation algorithm delivers a similar output distribution as the specification distribution; however, with a pre-chosen backdoor, the adversary can invert the entire family of functions that are generated by this subverted algorithm. We then show that such (adversarial) objects can indeed be constructed from (trapdoor) one-way functions by showing that random padding, in particular, renders the cryptosystem vulnerable. We also provide a weaker notion, *forgeability*, for one-way functions, that captures the case where the adversary only sets up the public parameters for which she keeps a backdoor.

- Our main goal is to provide defending mechanisms against kleptographic attacks. We construct *unforgeable* (trapdoor) one-way functions via a general transformation that "sanitizes" arbitrary OWFs by *randomizing the function index*. This transformation clips out all potential correlation between the function and the possible backdoor that the adversary may possess. Additionally, we introduce a *split-program* strategy to make the general method above applicable using only standard hash functions. In the split-program model, the function generation algorithm is composed of two

[1]This paper won the 2015 PET award.

parts: a (randomized) randomness generation algorithm RG that outputs an (ostensibly) uniform bit string, and a (deterministic) function generation algorithm dKG that converts such random string into the function index. We remark that our results even allow the sanitizing algorithm to be implemented by the adversary.

- In Section 4, we investigate how to construct backdoor-free PRGs. Previously, Dodis et al. [8] investigated "backdoored PRG" in which the adversary sets up a PRG instance (i.e., the public parameter), and is able to distinguish the output from uniform with a backdoor. They then proposed immunizing strategies obtained by applying a keyed hash function to the output, *but assuming the key is unknown to the adversary* in the public parameter generation phase.

  We construct backdoor-free PRGs in the complete subversion model. Our first construction is based on the classic Blum-Micali using our strongly unforgeable OWF and the Goldreich-Levin hardcore predicate [9]. In addition, in [8], Dodis et al. show that it is impossible to have a public immunizing strategy for all PRGs by applying a public function to the PRG output. This is because there always exists a PRG (having the immunizing function built in) that reveals the seed to the adversary bit by bit in each iteration. We circumvent their impossibility result via an alternative public immunizing strategy. Instead of randomizing the output of the PRG, we randomize the public parameter of PRG, which gives us a general construction for PRG in the complete subversion model.

- Having constructed unforgeable (trapdoor) one-way functions, we next explore the power of such primitives. In Section 5, we observe that unforgeable trapdoor one-way functions immediately give us a way to construct key generation algorithms (for digital signature schemes and stateless) against kleptographic attacks. We then showcase a concrete example of digital signature scheme in the complete subversion model. More concretely, we achieve this result by (1) using the unforgeable trapdoor one way permutation directly as a key generation algorithm, and then (2) instantiating the unique signature mechanism with the full domain hash. In previous works, [1,3] demonstrated that a unique signature scheme is secure against kleptographic attacks, *assuming that the key generation algorithm is honest* and all the message-signature pairs can be checked by the lab/user. Our result is the first digital signature scheme allowing the adversary to sabotage all algorithms.

We remark that our general defending technique is different from known methods. We here use a potentially subverted hash function to "randomize" the index and public parameter of a randomized algorithm so that steganographic channels can be eliminated. While previously, we have to either use a trusted random source to re-randomize the output of a randomized algorithm, or we consider only deterministic algorithms. We further remark that, the randomized algorithms in this paper will be executed only *once* to generate keys or public parameters for other algorithms; This already improves the state-of-the-art of defending strategy of addressing only deterministic algorithms. We emphasize that eventually we would like to deal with randomized algorithms that can be executed for arbitrary number of times.

1.2 Related work

The concept of *kleptography*—subverting cryptographic algorithms by modifying their implementations to leak secrets covertly, was proposed by Young and Yung [21,22] in 1996. They gave concrete examples showing that backdoors can be embeded into the public keys of commonly used cryptographic schemes; while the resulting public keys appear normal to the users, the adversary is nevertheless capable of learning the secret keys. It may not be surprising that defending against such deliberate attacks is challenging and only limited feasibility results exist. We next briefly describe these existing results.

In [13], Juels and Guajardo suggested the following idea: the user and a trusted certificate authority (CA) jointly generate the public key; as a part of this process, the user proves to the CA that the public key is generated honestly. This contrasts markedly with our setting, where the the user does not have any secret, and every component is provided by the big brother.

Bellare et al. considered a powerful family of kleptographic attacks that they call *algorithm substitution attacks*, and explore these in both symmetric key [3] and public key [2] settings. They first proposed a generic attack, highlighting the relevance of steganographic techniques in this framework: specifically, a sabotaged randomized algorithm can leak a secret bit-by-bit by invoking steganographic rejection-sampling; then an adversary possessing the backdoor can identify the leaked bits from the biased output, which appears unmolested to other observers. The analysis relies on the effectiveness of covert subliminal channels [11,19,20]. They then introduced a framework for defending against such attacks by focusing on algorithms that behave deterministically, determining a unique output for each input: relevant examples of such algorithms include unique ciphertext encryption algorithms (of encryption schemes).Their defending mechanism does not, however, address the (necessarily randomized) process of key generation—it implicitly assumes key generation to be honest. This state of affairs is the direct motivation of the current article: we adopt a significantly amplified *complete subversion model* where *all* cryptographic algorithms—including key generation—are subject to kleptographic (i.e., substitution) attacks. This forces us to manage certain randomized algorithms (such as key generation) in a kleptographic setting. The details of the model, with associated commentary about its relevance to practice, appear below.

Dodis et al. [8] studied an alternative family of kleptographic attacks on pseudorandom generators in order to formalize and study the notorious Dual_EC PRG subversion [6,17]. In their model, the adversary subverts the security of the PRG by opportunistically setting the public parameter while privately keeping some backdoor information (instead of providing an implementation). They prove the equivalence of such a "backdoored PRG" and public key encryption with pseudorandom ciphertexts. Then they proposed immunizing strategies obtained by applying a keyed hash function to the output (of the PRG). Note that the (hash) key plays a special role in their model: it is selected uniformly and is unknown to the adversary during the public parameter generation phase. These results likewise inspire our adoption of the amplified *complete subversion model*, which excludes such reliance on public randomness beyond the reach of the adversary. We remark that our results on strongly unforgeable OWFs can be applied to construct a specific "backdoor-free" PRG following the classic Blum-Micali framework. Moreover, our general immunizing strategy, randomizing the public parameter of a backdoored PRG instead of randomizing the PRG output, permits us to bypass an impossibility result established by Dodis et al. for general public immunization based on the PRG output.

Other works suggest different angles of defense against mass surveillance. For example, in [16], the authors proposed a general framework of safeguarding protocols by randomizing the incoming and outgoing messages via a trusted (reverse) firewall. Their results demonstrate that with a clean trusted random source, many tasks become achievable. As they rely on a "subversion-free" firewall, these results require a more generous setting than provided by our *complete subversion model*.

Recently, Ateniese et al [1] continue the study of algorithm substitution attack on signatures and propose two defending mechanisms, one is using a unique signature scheme assuming the key generation and verify algorithms to be honest; the other is in the reverse firewall model that assumes trusted randomness. Our work remove those assumptions and work in a complete subversion model.

2 Kleptographic Attacks and the Complete Subversion Model

In this section, we explain the kleptographic attacks in more detail and introduce the complete subversion model.

Classical cryptography assumes that the relevant cryptographic algorithms are faithfully implemented and, moreover, that participants have access to truly private randomness. In reality, cryptographic algorithms may be implemented by an adversary (e.g., the "big brother"); this potentially allows the adversary to exert malicious influence on the behavior of the algorithms and, for example, learn secret information that is not supposed to be leaked thru the algorithms. Kleptography studies how to apply these attacks on real-world cryptosystems with the extra condition that the attacks are undetectable. Specifically, while big brother wishes to monitor the system, he does not wish to be exposed.

The detection conditions. We work under the assumption that for every cryptographic scheme of interest, there exists a specification that is rigorously analyzed, proven secure, and designed by honest experts; furthermore, one can actively actively check whether implementations (supplied by the adversary) faithfully meet the specification. This "checking" procedure, however, is only assumed to have oracle access to the potentially sabotaged implementations. Following the formalization of Bellare et al. [3] of the detection condition, the adversary's algorithms must fool *all* PPT checkers.[2] Specifically, we say the adversarial subversion on cryptographic primitive Π is *undetectable*, if there exists a PPT adversary $\mathcal{A}$, for any PPT checker/detecter $\mathcal{D}$ that plays the following detection game (Figure 1), it holds that $|\Pr[b' = b] - 1/2| \leq \epsilon$, where $\epsilon = \mathsf{negl}(\lambda)$, $\mathsf{R}^1, \ldots, \mathsf{R}^k$ are the algorithms implemented for cryptographic primitive Π, and $\mathsf{R}^1_{\text{SPEC}}, \ldots, \mathsf{R}_{\text{SPEC}}$ are the corresponding specifications for Π and the probability is over the coins of the adversary $\mathcal{A}$ (for generating the backdoors or the implementations) and the detecter $\mathcal{D}$.

$$\boxed{\begin{array}{l} \overline{\mathsf{R}}_0 \leftarrow \langle \mathsf{R}^1_{\text{SPEC}}, \ldots, \mathsf{R}^k_{\text{SPEC}} \rangle \\ \overline{\mathsf{R}}_1 = \langle \mathsf{R}^1, \ldots, \mathsf{R}^k \rangle \leftarrow \mathcal{A} \\ b \leftarrow \{0,1\} \\ b' \leftarrow \mathcal{D}^{\overline{\mathsf{R}}_b}(\lambda) \end{array}}$$

Figure 1: The detection game.

We emphasize that the algorithms $\langle \mathsf{R}^1, \ldots, \mathsf{R}^k \rangle$ may be randomized; indeed, the striking historical examples require the adversary to embed a (randomized) backdoor into the algorithms R^i in order to undetectably alter their behavior in a way that can, e.g., leak private data to the adversary. In any case, it follows that the adversary must certainly ensure that the output of any algorithm (randomized or deterministic) is computationally indistinguishable from the output of the specified algorithm, taken over any distribution of inputs of the checker's choice (and the random coins of the adversary).

We first formally present a simple observation that when a *deterministic* algorithm with a *public* input distribution (whether specified by the adversary or not) is subverted in a way that the implementation is inconsistent with the specification at a noticeable fraction of inputs, then it is easily detected.

Lemma 2.1. *If an adversarial subversion on deterministic algorithms $\langle \mathsf{R}^1, \ldots, \mathsf{R}^k \rangle$ (with the corresponding specifications $\langle \mathsf{R}^1_{\text{SPEC}}, \ldots, \mathsf{R}^k_{\text{SPEC}} \rangle$) is undetectable, then for every public input distribution of $X_1, \ldots, X_k$, where X_i corresponds to R^i, it holds that: $\Pr[\mathsf{R}^i(x) \neq \mathsf{R}^i_{\text{SPEC}}(x) : x \leftarrow X_i] \leq \epsilon$, where $\epsilon = \mathsf{negl}(\lambda)$, and the probability is over the coins for sampling inputs.*

Proof. Suppose there exists an implementation R^i for a deterministic algorithm, and a public input distribution X_i, such that, $\Pr[\mathsf{R}^i(x) \neq \mathsf{R}^i_{\text{SPEC}}(x) : x \leftarrow X_i] \geq \delta$, for a non-negligible amount δ; then there is a simple detecter algorithm $\mathcal{D}$ that samples $t = O(1/\delta^2)$ inputs $x^i_1, \ldots, x^i_t$ from X_i, and checks whether $\mathsf{R}^i(x^i_j) = \mathsf{R}^i_{\text{SPEC}}(x^i_j)$ for all $j = 1, \ldots, t$. Following the Chernoff bound, there exists a $j \in [1, t]$, such that $\mathsf{R}^i(x^i_j) \neq \mathsf{R}^i_{\text{SPEC}}(x^i_j)$ is with an overwhelming probability. $\square$

Remark 2.2. *(i.) The above lemma states that for a deterministic algorithm, if the adversary wants to make the subversion undetectable, then for any public input distribution, the implementation will*

[2] Actually, we can even work in a model that relaxes the detection condition for our (trapdoor) one way functions and PRG.

be inconsistent with the specification for at most a negligible fraction of the inputs; In the rest of the presentation, we sometimes use the specification directly for simplicity for deterministic algorithms with public input distribution. (ii.) The above lemma does not require the input distributions to be honestly generated.

Corollary 2.3. *Assume a hash function h_{SPEC} is modeled as a random oracle, and h is a (potentially subverted) implementation. If the subversion of h is undetectable, then for any public input distribution X, $\Pr[h(x) = h_{\text{SPEC}}(x) : x \leftarrow X] \leq \epsilon$, where $\epsilon = \mathsf{negl}(\lambda)$. To put it in another way, h can be still modeled as a random oracle w.r.t to input distribution X.*

With the above detection condition only, it is impossible to achieve meaningful security for several cryptographic primitives. For example, as shown in [1, 12], symmetric key encryption and signature schemes are impossible that the adversary may learn the secret key completely via the so-called "input-triggered subversion" [12]. In order to show feasibility for those primitives, we may require some extra assumption. Bellare et al. introduced the decryptability assumption that all messages encrypted by a subverted encryption algorithm should be decrypted to the original message by the specification decryption algorithm [3]; and similarly Ateniese et al. used a verifiability assumption [1] that every signature produced by the subverted signing algorithm should pass the verification of the specification verification algorithm.

Degabriele et al. relaxed the perfect decryptability condition, and formalized a stronger detection model for symmetric key encryption in [12]. In particular, the detecter $\mathcal{D}$ may have access to the transcripts that the adversary uses to gain advantage. Subversion resistance in this case means that if the subverted algorithms leak information to the adversary, then from the recorded transcript, there exist an efficient detector that can notice the subversion. We choose this formalization for our signature and public key encryption scheme.

More formally, in the following game (Fig 2), the subversion advantage $\delta_s := |\Pr[b' = b] - 1/2|$, and the (strong) detection advantage δ_d is defined by $|\Pr[b'' = b] - 1/2|$. We say the subversion on the algorithms $\langle \mathsf{R}^1, \ldots, \mathsf{R}^k \rangle$ are strongly undetectable if δ_d is negligible; the algorithms $\langle \mathsf{R}^1, \ldots, \mathsf{R}^k \rangle$ are subversion resistant if $\delta_s \leq \delta_d$.

```
$\overline{\mathsf{R}}_0 \leftarrow \langle \mathsf{R}^1_{\text{SPEC}}, \ldots, \mathsf{R}^k_{\text{SPEC}} \rangle$
$\langle (\mathsf{R}^1, z^1), \ldots, (\mathsf{R}^k, z^k) \rangle \leftarrow \mathcal{A}$
$\overline{\mathsf{R}}_1 = \langle \mathsf{R}^1, \ldots, \mathsf{R}^k \rangle$
$\overline{z} = \langle z^1, \ldots, z^k \rangle$
$b \leftarrow \{0, 1\}$
$b' \leftarrow \mathcal{A}^{\overline{\mathsf{R}}_b}(\lambda, \overline{z})$
$b'' \leftarrow \mathcal{D}^{\overline{\mathsf{R}}_b}(\lambda, \tau)$
```

Figure 2: The strong detection and subversion game, τ is the transcript of $\mathcal{A}$ for producing b'.

Remark 2.4. *(i.) The strong detection model essentially describes the situation that the "big brother" is worried about any possible detection. (ii.) When we use the stronger detection condition for signature and public key encryption scheme, the detection is public (verifying the signatures or encrypting the messages using a public key), this is in contrast with [3, 12] that the detecter needs the user secret key.*

Complete subversion model. Significant effort has been invested to defend against such kleptographic attacks. However, the state-of-the-art of defending mechanisms require the participants to have access to various trusted or private sources of randomness. For example, as described above, key generation is assumed to be honest in [3]. Unfortunately, key generation, as shown in the original papers of Young and Yung, [21, 22], can be directly subjected to kleptographic attacks. This motivates our focus on the *complete subversion model*, in which attacks may be launched against *any* of the relevant cryptographic algorithms. This includes, e.g., the *key generation* algorithm, and even the *defending algorithm*. We remark that the immunizing strategy of [8] requires an honestly generated uniform seed unknown to the adversary when she implements the algorithms. We will define security for each primitive we consider in this complete subversion model. Likewise, we analyze our defending mechanisms in this stringent model. We call this general paradigm **cliptography**.

3 One-Way Functions in the Complete Subversion Model

3.1 Formalizing kleptographic attacks on one way functions

Before studying the security for one-way functions (OWF) and trapdoor one-way functions (TDOWF) in the presence of kelptographic attacks, we first recall the conventional definitions of OWF and TDOWF.

One-way function (OWF). A function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i\in I}$ is *one-way* if there are PPT algorithms $(\mathsf{KG}, \mathsf{Eval})$ so that (i) KG, given a security parameter λ, outputs a function index i from $I_\lambda = I \cap \{0,1\}^\lambda$; (ii) for $x \in X_i$, $\mathsf{Eval}(i,x) = f_i(x)$; (iii) $\mathcal{F}$ is one-way; that is, for any PPT algorithm $\mathcal{A}$, it holds that $\Pr[\mathcal{A}(i,y) \in f_i^{-1}(y) \mid i \leftarrow \mathsf{KG}(\lambda); x \leftarrow X_i; y := f_i(x)] \le \mathsf{negl}(\lambda)$.

Trapdoor one way function (TDOWF). A function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i\in I}$ is *trapdoor one-way* if there are PPT algorithms $(\mathsf{KG}, \mathsf{Eval}, \mathsf{Inv})$ such that (i) KG, given a security parameter λ, outputs a function index and the corresponding trapdoor pair (i, t_i) from $I_\lambda \times T$, where $I_\lambda = I \cap \{0,1\}^\lambda$, and T is the domain of t_i; (ii) $\mathsf{Eval}(i,x) = f_i(x)$ for $x \in X_i$; (iii) $\mathcal{F}$ is one-way; and (iv) it holds that $\Pr[\mathsf{Inv}(t_i, i, y) = x \mid i \leftarrow \mathsf{KG}(\lambda); x \leftarrow X_i; y := f_i(x)] \ge 1 - \mathsf{negl}(\lambda)$.

We note that, (trapdoor) one-way permutations can be defined similarly by setting $Y_i := X_i$. For simplicity, we often ignore Eval; for evaluating function with index i on input x, i.e., $\mathsf{Eval}(i,x)$, we write it as $f_i(x)$.

3.1.1 Strong-forgeability and unforgeability for OWF/TDOWF

In this subsection, we define *strong-forgeability* to capture kleptographic attacks on one-way functions. Note that the complemented security notion, *unforgeability*, provides the guarantee that OWF is immune to kleptographic attacks. Similarly, we can define strong-forgeability and the complemented notion, unforgeability, for trapdoor OWF. Next, let's start with expressing the intuition of capturing kleptographic attacks on one-way functions.

We begin with a "laboratory specification" version of the OWF, $(\mathsf{KG}_{\text{SPEC}}, \mathsf{Eval}_{\text{SPEC}})$, which has been rigorously analyzed and certified (e.g., by the experts in the cryptography community).

The adversary then provides an alternate implementation. Note that Eval is deterministic on publicly known input distribution ($i \times U_i$, where i is the output of KG, and U_i is the uniform distribution over X_i). Following lemma 2.1, the event that an input is sampled from KG and the X_i s.t., the implementation Eval is inconsistent with $\mathsf{Eval}_{\text{SPEC}}$ will happen with only a negligible probability. It follows that using $\mathsf{Eval}_{\text{SPEC}}$ directly for Eval will reduce the adversary advantage at most a negligible amount, thus we can consider it as honestly implemented.We therefore focus on KG (the algorithm which generates a function name).

The goal of the adversary is to privately maintain some "backdoor information" z so that the subverted implementation of KG will output functions that can be inverted using z. In addition, the adversary must be sure that the output distributions of $\mathsf{KG}(z)$ and that of specification function generation algorithm $\mathsf{KG}_{\text{SPEC}}$ are computationally indistinguishable, to avoid detection. Formally, we define *strongly-forgeable OWFs*; we note that this immediately allows us to define the complemented notion, *unforgeable OWFs*.

Definition 3.1. *A one-way function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i \in I}$ (with the specification function generation algorithm $\mathsf{KG}_{\text{SPEC}}$) is δ-**strongly forgeable** if there exist PPT algorithms $(\mathsf{BG}, \mathsf{KG}, \mathsf{Inv})$ so that given a backdoor z produced by backdoor generation algorithm BG, i.e., $z \leftarrow \mathsf{BG}(\lambda)$, the function generation algorithm KG generates a function index i that is (1) invertible given z, and (2) computationally indistinguishable from that generated by the specification function generation algorithm $\mathsf{KG}_{\text{SPEC}}$. That is, for every $z \leftarrow \mathsf{BG}(\lambda)$, it holds that:*

(1) $\Pr[x' = x \mid i \leftarrow \mathsf{KG}(\lambda, z); x \leftarrow X_i; y := f_i(x); x' \leftarrow \mathsf{Inv}(z, i, y)] \geq \delta$

(2) $\{i \mid i \leftarrow \mathsf{KG}(\lambda, z)\} \overset{c}{\approx} \{i \mid i \leftarrow \mathsf{KG}_{\text{SPEC}}(\lambda)\}$

*Correspondingly, we say a one-way function family is **unforgeable** if it is* not *δ-strongly forgeable for any non-negligible function δ.*

The notion of strongly-forgeable OWF is closely related to the conventional notion, TDOWF. See the following lemmas. These results state that if we want to use public key cryptography, we have to accept the possibility of kleptographic attacks on OWFs. For detailed proofs of the lemmas, we defer to Appendix A.

Lemma 3.1. *A $(1 - \epsilon)$-strongly forgeable OWF family is also a TDOWF family, where ϵ is a negligible function of the security parameter.*

Next, we show how to construct a strongly-forgeable OWF from any TDOWF. This substantiates the folklore knowledge that sufficient random padding can render cryptosystems vulnerable to backdoor attacks, e.g., [21,22]. Specifically, the random padding in the malicious implementation can be generated so that it encrypts the corresponding trapdoor using the backdoor as a key.

Lemma 3.2. *One can construct a $(1 - \epsilon)$-strongly forgeable OWF from a TDOWF, where ϵ is a negligible function on the security parameter.*

The notions of strongly-forgeable OWF and TDOWF are similar in the sense that they both posit a secret that enables inversion of the OWF. Observe, however, that a strongly-forgeable OWF has a further (critical) property: the distribution of function names, is indistinguishable from a particular specification distribution. The same principle yields a notion of strongly-forgeable TDOWF. Moreover, the adversary can invert using the backdoor, without referring to the regular trapdoor. The formal definition is presented below.

Definition 3.2. *A trapdoor one-way function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i \in I}$ (with the specification algorithms $\mathsf{KG}_{\text{SPEC}}, \mathsf{Inv}_{\text{SPEC}}$) is δ-**strongly forgeable** if there exist PPT adversarial algorithms $\mathcal{A} = (\mathsf{BG}, \mathsf{KG}, \mathsf{Inv})$ so that given a backdoor z produced by backdoor generation algorithm BG, the function generation algorithm KG generates a function index i and the corresponding trapdoor t_i, with the following properties: (1) f_i is invertible given z without providing t_i, and (2) the output of KG is computationally indistinguishable from that generated by the specification function generation algorithm $\mathsf{KG}_{\text{SPEC}}$. That is, for every $z \leftarrow \mathsf{BG}(\lambda)$, it holds that*

(1) $\Pr[x' = x \mid (i, t_i) \leftarrow \mathsf{KG}(\lambda, z); x \leftarrow X_i; y := f_i(x); x' \leftarrow \mathsf{Inv}_{\text{SPEC}}(z, i, y)] \geq \delta$ [3]

(2) $\{(i, t_i) \mid (i, t_i) \leftarrow \mathsf{KG}(\lambda, z)\} \overset{c}{\approx} \{(i, t_i) \mid (i, t_i) \leftarrow \mathsf{KG}_{\text{SPEC}}(\lambda)\}$

*Correspondingly, we say a trapdoor one-way function family is **unforgeable** if it is* not *δ-strongly forgeable for any non-negligible function δ.*

[3]The inversion algorithm of the TDOWF never appears in the security definition, thus we omit the indistinguishability condition for this algorithm, considering it is honestly implemented as $\mathsf{Inv}_{\text{SPEC}}$, and focus only on KG.

3.1.2 Forgeability and strong-unforgeability for OWF

The notion of strong forgeability models an attack where the adversary may provide a subverted implementation of the defining algorithms. In many cases, it may also be interesting to consider a weaker form of attack that the adversary simply provides the function index, this is similar to the notion of backdoored PRG that the adversary sets up the public parameters (as in the Dual_EC PRG example [6]). This is a weaker definition in that the adversary only have to generate a backdoor that can be helpful for inverting one single function instead of a family of functions (as in the case of strongly-forgeable OWF). It is easy to see that this notion is equivalent to the standard notion of TDOWF if the $\mathsf{KG}_{\text{SPEC}}$ outputs the same distribution of the function index as the regular generation algorithm of the TDOWF, thus we only consider this notion for OWF. However, putting into the context of kleptography, it is suggested that when the experts recommend standard for OWF, TDOWF should not be a good candidate.

More importantly, we will later consider unforgeability for OWF, thus the reverse of the weaker notion will yield a stronger definition for unforgeability, and we will give a generic construction that converts any OWF into a strongly unforgeable OWF, which means we can destroy the trapdoor structure generically.

Definition 3.3. *A one-way function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i \in I}$ (with the specification of function generation algorithm $\mathsf{KG}_{\text{SPEC}}$) is δ-**forgeable** if there exist PPT adversarial algorithms $\mathcal{A} = (\mathsf{KG}, \mathsf{Inv})$ so that*

1. $\Pr[x' = x \mid (i, z) \leftarrow \mathsf{KG}(\lambda); x \leftarrow X_i; y \leftarrow f_i(x); x' \leftarrow \mathsf{Inv}(z, i, y)] \geq \delta$
2. *The distribution of i is indistinguishable from the output distribution of $\mathsf{KG}_{\text{SPEC}}$.*

*Correspondingly, we say a one-way function family is **strongly-unforgeable** if it is not δ-forgeable for any non-negligible function δ.*

3.2 Eliminating backdoors

In this section, we discuss methods for safeguarding OWF generation against kleptographic attacks. We first present a general approach that immunizes any OWF generation procedure. We prove that hashing the function index is sufficient to eliminate potential backdoor information. We remark that the hash function can be implemented by the adversary, and will be part of the algorithms (with its own specification). Our defending mechanism can be seen as a suggestion to change the implementation according to a new specification.

However, in many cases of interest, function indices have specific algebraic structure. It is not clear in general how one can guarantee that a public hash function has such "structure preserving" properties. In order to apply our approach in more general settings, we propose a "split-program" model in which the function generation algorithm is necessarily composed of two parts: a random string generation algorithm RG that outputs random bits r, and a deterministic function index generation algorithm dKG which uses r to generate the index.

3.2.1 General feasibility results

We will show below that randomizing the function index (that is, the relationship between names and functions) can provide satisfactory immunization against possibly sabotaged function generation. The intuition behind this idea is that according to the definition of forgeable OWF, each backdoor that frequently appears can only be useful for inverting a sparse subset of one way functions (i.e., the range of $\mathsf{KG}(z)$ is exponentially sparse for every z, otherwise, one can use such backdoor to break the one-wayness of the functions generated by $\mathsf{KG}_{\text{SPEC}}$). Thus, randomizing the function index will map the function index to a "safe" domain, and destroys the possible correlation with any selected backdoor. That said, it is difficult for the adversary to arrange a backdoor that works for a disturbed subset of function index (after hashing), even if she knows the immunizing strategy.

Constructing strongly-unforgeable OWFs. Given any OWF family $\mathcal{F} := (\mathsf{KG}_\mathcal{F}, \mathsf{Eval}_\mathcal{F}) := \{f_i\}_{i \in I}$ (which might be forgeable, and with specification $\mathsf{KG}_{\text{SPEC}}, \mathsf{Eval}_{\text{SPEC}}$) that is secure if $\mathsf{KG}_{\text{SPEC}}(\lambda)$ outputs uniform i from I_λ, we will construct a strongly unforgeable OWF family $\mathcal{G} := (\mathsf{KG}_\mathcal{G}, \mathsf{Eval}_\mathcal{G}) = \{g_i\}$. We assume a (public) hash function $h_{\text{SPEC}} : \{0,1\}^* \to \{0,1\}^*$ modeled as a random oracle.

The key generation algorithm $\mathsf{KG}_\mathcal{G}$ is the same as $\mathsf{KG}_\mathcal{F}$, and $\mathsf{Eval}_\mathcal{G}$ is given as $\mathsf{Eval}_\mathcal{F} * h$ (where h is the implementation of h_{SPEC}), and it is defined as $\mathsf{Eval}_\mathcal{G}(i, x) := \mathsf{Eval}_\mathcal{F}(h(i), x)$. (thus for each i, $g_i(\cdot) = f_{h(i)}(\cdot)$). [4] See also the pictorial illustration in Fig 3.

$$\boxed{\mathsf{KG}_\mathcal{F}} \to i \qquad i \to \boxed{h} \to \tilde{i} \qquad x \to \boxed{\mathsf{Eval}_\mathcal{F}(\tilde{i}, \cdot)} \to y$$

Figure 3: Immunization strategy for OWF.

Remark 3.4. *In the case that we can build hash functions that hash directly onto the index space I_λ, then we can use the above immunization strategy. However, in practice, it is not clear if we can always easily build hash functions on index space. To address this issue, in Section 3.2.2, we introduce a new framework called "split-program model". There, we can use standard hash function, e.g., SHA-256 to work on the random bits directly.*

Theorem 3.5. *The OWF family $\mathcal{G}$ defined above is strongly unforgeable if h_{SPEC} is assumed to be a random oracle model.*

Proof. Suppose that $\mathcal{G}$ is δ-forgeable for a non-negligible function δ, and let $(\mathcal{A}_{\mathsf{KG}}, \mathcal{A}_{\mathsf{Inv}})$ be the adversarial algorithms of Definition 3.3. We will construct a simulator $\mathcal{S}$ which will break the one-way security of $\mathcal{F}$.

Suppose $(f_{\tilde{i}}, y)$ are the challenges $\mathcal{S}$ receives from the one way security challenger $\mathcal{C}$, where $y = f_{\tilde{i}}(x)$ for a randomly selected x. Then (i.) $\mathcal{S}$ first randomly samples a bit b to decide whether to embed j into the answers to the random oracle queries from $\mathcal{A}_{\mathsf{KG}}$ or $\mathcal{A}_{\mathsf{Inv}}$. (W.l.o.g., we assume all the random oracle queries are different.) (ii.) $\mathcal{S}$ runs $\mathcal{A}_{\mathsf{KG}}$. Suppose $\mathcal{A}_{\mathsf{KG}}$ makes q_1 random oracle queries $Q_1 = \{i_1, \dots, i_{q_1}\}$. If $b = 0$, $\mathcal{S}$ randomly selects an index $t_1 \in \{1, \dots, q_1\}$. When answering the random oracle queries $i_1, \dots, i_{q_1}$ from $\mathcal{A}_{\mathsf{KG}}$, $\mathcal{S}$ answers $\tilde{i}$ for $h(i_{t_1})$; for all other queries, $\mathcal{S}$ answers with random elements from the index set I_λ. If $b \neq 0$, $\mathcal{S}$ answers all these queries using random elements from I_λ. $\mathcal{S}$ maintains a list for the query-answer pairs. $\mathcal{A}_{\mathsf{KG}}$ outputs a pair (i, z).

If $[b = 0 \land i \neq i_{t_1}]$, $\mathcal{S}$ aborts; otherwise, $\mathcal{S}$ runs $\mathcal{A}_{\mathsf{Inv}}$ with inputs (i, y, z). Assuming $\mathcal{A}_{\mathsf{Inv}}$ asks q_2 random oracle queries, $\mathcal{S}$ sets $\tilde{i}$ as $h(i)$ (even if i is not asked) and for all others queries, $\mathcal{S}$ answers with random elements from I_λ. $\mathcal{A}_{\mathsf{Inv}}$ outputs x'.

If $[b = 1 \land i \in Q_1]$, $\mathcal{S}$ aborts; otherwise, it returns x' as his answer to $\mathcal{C}$.

Probabilistic analysis. Now we bound the success probability of $\mathcal{S}$. Let us use W to denote the event that $\mathcal{S}$ aborts, W_1 to denote the event that $b = 0 \land i \neq i_{t_1}$, and W_2 to denote the event that $b = 1 \land i \in Q_1$. We have

$$\Pr[x' = x] = \Pr[x' = x | W] \Pr[W] + \Pr[x' = x | \overline{W}] \Pr[\overline{W}] \geq \Pr[x' = x | \overline{W}] \Pr[\overline{W}]$$

[4] Note that the hash function can be implemented by the adversary, since it is deterministic and with a public input distribution (the output distribution of $\mathsf{KG}_\mathcal{F}$). We can also interpret the specifications for $\mathcal{G}$ as $(\mathsf{KG}_{\text{SPEC}}, \mathsf{Eval}_{\text{SPEC}} * h_{\text{SPEC}})$.

We first bound $\Pr[\overline{W}]$ as $1-\Pr[W]$, we have $\Pr[W] = \Pr[W_1 \vee W_2] \leq \Pr[W_1] + \Pr[W_2]$. Assuming $\Pr[i \in Q_1] = \eta$, we bound $\Pr[W_1]$ as follows:

$$\begin{aligned}
\Pr[W_1] &= \Pr[b = 0 \wedge i \neq i_t] = \Pr[b = 0]\Pr[i \neq i_{t_1}] \\
&= \frac{1}{2}(\Pr[i \neq i_{t_1} | i \in Q_1]\Pr[i \in Q_1] + \Pr[i \neq i_{t_1} | i \notin Q_1]\Pr[i \notin Q_1]) \\
&= \frac{1}{2}\left[\left(1 - \frac{1}{q_1}\right)\eta + (1-\eta)\right]
\end{aligned}$$

While $\Pr[W_2] = \Pr[b = 1]\Pr[i \in Q_1] = \eta/2$, we have: $\Pr[W] \leq \frac{1}{2}\left(1 - \frac{1}{q_1}\right)\eta + \frac{1}{2} \leq 1 - \frac{1}{2q_1}$. Thus we can derive that $\Pr[\overline{W}] \geq 1/(2q_1)$.

Furthermore, conditioned on $\mathcal{S}$ not aborting, the input distributions of the adversaries $(\mathcal{A}_{\mathsf{KG}}, \mathcal{A}_{\mathsf{Inv}})$ are identical to that of Definition 3.3. By definition of δ-forgeability, $\Pr[\mathcal{A}_{\mathsf{Inv}}(\tilde{i}, f_{\tilde{i}}(x), z) = x] \geq \delta$, thus $\Pr[x' = x | \overline{W}] \geq \delta$.

Combing these facts, we conclude that if $\mathcal{G}$ is δ-forgeable for some non-negligible δ, then there exists an algorithm $\mathcal{S}$ that breaks the one-way security of $\mathcal{F}$ with probability at least $\delta/(2q_1)$ (which is non-negligible). This completes the proof. □

Remark 3.6. *(i.) Actually, it is not very hard to see from our analysis that we can even relax the detection condition for* KG *that it only has to pass one particular tester. (ii.) We may also prove a similar result in the standard model, that the random oracle is instantiated with a pseudorandom function* PRF*. However, in this case, the adversary* KG *can only have oracle access to the* PRF*, but the* Inv *can have full access to the* PRF *key. Still, this result will not be in the complete subversion model that the* PRF *requires a trusted key. Thus we leave as an open problem that how to establish a general immunizing result in the standard model. Furthermore, the above proof works even if the distribution of i is different with the output distribution of* $\mathsf{KG}_{\text{SPEC}}$*, as long as it still belongs to the index set.*

3.2.2 Practical results in the split-program model

Indices (names) of a one-way function family may have structure. For example, for OWF based on discrete logarithm, $f_{g,p}(x) = g^x \bmod p$, the function index consists of an algebraically meaningful pair (p, g), where p is a prime and g a random generator. This would require that the hash function in the general immunization method above maps (g, p) to (g', p'); note that (g', p') is another algebraically meaningful pair with the same structure. Furthermore, for a TDOWF, the hash function needs to map the function/trapdoor pair to another function/trapdoor pair. It is not clear in general how one can guarantee that a public hash function has such "structure preserving" properties.

To address this problem, we propose a split-program model in which every function generation algorithm is composed of two algorithms, a random string generation algorithm RG that outputs a uniform ℓ-bit random string r, and a deterministic function index generation algorithm dKG that transforms the randomness r into a function index i. In this model, dKG is deterministic with a public input distribution (output distribution of RG). Following lemma 2.1 and the elaboration in section 3.1.1, we can consider it to be honestly implemented and we can focus on "cleaning up" the randomness generated by RG. [5]

Remark 3.7. *It is not hard to see, the split-program model is quite general and can be applied to most practical algorithms. To see this, the user gets the source code of the implementation, which makes calls*

[5] The split-program model essentially forces the adversary to concentrate the parts which may potentially contain backdoors of a malicious implementation into RG. This gives us more flexibility to apply the sanitizing strategy. Furthermore, conceptually, the immunizing strategy itself (e.g., Fig 3) can be seen as a bigger piece of implementation in the split-program model, i.e., the hash function and the actual KG algorithm should be individually implemented and checked.

to some API for generating randomness (e.g., `rand()` *) whenever necessary. The user can hook up the interface with the calls to the API with the separate program* RG *provided by the big brother. In principle, one can always augment a randomized* KG *algorithm to output the function index i together with the randomness r used to generate i, thus the* RG *can be implemented as this augmented* KG*, and discards the function index i from the output.*

We first rephrase the standard OWF/TDOWF definitions in the split-program model.

Definition 3.8. *A function family $\mathcal{F}$ is one way in the split-program model if there exist a pair of algorithms* $(\mathsf{RG}, \mathsf{dKG})$ *where (i.)* RG*, given a security parameter λ, outputs a uniform $\ell(\lambda)$-bit string r; (ii.)* dKG *is deterministic: given the randomness r it outputs a function index $i \in I_\lambda$; and (iii.) $\mathcal{F}$ is one-way under this procedure for generating i.*

Similarly, we can define a TDOWF family in the split program model (In this case, dKG outputs a function index together with a trapdoor). For the ease of presentation, we often use the pair of algorithms $(\mathsf{RG}, \mathsf{dKG})$ to represent the OWF/TDOWF family $\mathcal{F}$ in the split-program model. Next we define δ-forgeable OWF in the split-program model by modifying Definition 3.3; we immediately have the complemented security notion of strongly-unforgeable OWF in the split-program model. [6]

Definition 3.9. *A one-way function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i \in I}$ (with the specification version of function generation algorithm* $\mathsf{RG}_{\mathrm{SPEC}}, \mathsf{dKG}_{\mathrm{SPEC}}$*) is **$\delta$-forgeable** if there exist* PPT *algorithms* $(\mathsf{RG}, \mathsf{dKG}, \mathsf{Inv})$ *such that:*

1. $\Pr[x' = x \mid (r, z) \leftarrow \mathsf{RG}(\lambda); i \leftarrow \mathsf{dKG}_{\mathrm{SPEC}}(\lambda, r, z); x \leftarrow X_i; y := f_i(x); x' \leftarrow \mathsf{Inv}(z, i, y)] \geq \delta\,.$
2. *The distribution of r is indistinguishable from the output distribution of* $\mathsf{RG}_{\mathrm{SPEC}}$.

*Correspondingly, we say a one-way function family is **strongly-unforgeable** in the split-program model if it is not δ-forgeable in the split-program model for any non-negligible function δ.*

Strongly-unforgeable OWF in the split-program model. Given a OWF family $\mathcal{F} := (\mathsf{RG}_\mathcal{F}, \mathsf{dKG}_\mathcal{F}, \mathsf{Eval}_\mathcal{F})$ (in the split program model) whose $\mathsf{RG}_{\mathrm{SPEC}}$ outputs uniform bits, and a public hash function $h(\cdot)$ randomly selected from a hash family $H : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{\ell(\lambda)}$ which is modeled as a random oracle, we construct an strongly-unforgeable OWF family $\mathcal{G}$. $\mathcal{G} := (\mathsf{RG}_G, \mathsf{dKG}_\mathcal{G}, \mathsf{Eval}_\mathcal{G})$ can be described as $(\mathsf{RG}_\mathcal{F}, \mathsf{dKG}_\mathcal{F} \circ h, \mathsf{Eval}_\mathcal{F})$, i.e., $\mathsf{dKG}_\mathcal{F} \circ h(r) = \mathsf{dKG}_\mathcal{F}(h(r))$ the function index i is generated by $\mathsf{dKG}_\mathcal{F}$ using $h(r)$ as randomness, where $r \leftarrow \mathsf{RG}_\mathcal{F}$. See Figure 4 below:[7]

$$\boxed{\mathsf{RG}_\mathcal{F}} \to r \qquad r \to \boxed{h} \to \tilde{r} \qquad \tilde{r} \to \boxed{\mathsf{dKG}_\mathcal{F}} \to \tilde{i} \qquad x \to \boxed{\mathsf{Eval}_\mathcal{F}(\tilde{i}, \cdot)} \to y$$

Figure 4: Immunization strategy for OWF in the split-program model.

The intuition is analogous to that captured in the proof of Theorem 3.5: For a backdoor z that frequently appears in the output of $\mathcal{A}_{\mathsf{RG}}$, the "bad" set of randomness (i.e., for which the dKG algorithm outputs a function that can be efficiently inverted using z) should be sparse in $\{0,1\}^{\ell(\lambda)}$. (Otherwise, one

[6] Note that it is easy to provide a more general definition to capture the fact dKG is also implemented by the adversary, we can simply require the indistinguishability condition hold for the joint output distribution of $(\mathsf{RG}, \mathsf{dKG})$. However as pointed out above, it is fine for us to consider the deterministic dKG as honestly implemented for simplicity, because it has the public input distribution from $h \circ \mathsf{RG}$.

[7] We emphasize that in the split-program model, we can use any regular hash function such as SHA-256. Furthermore, this can be implemented by the adversary, as h has a public input distribution (the output distribution of RG). This in turn guarantees dKG has a public input distribution (output distribution of $h \circ \mathsf{RG}$).

can break the one way security of $\mathcal{F}$ by simply running $\mathcal{A}_{\mathsf{KG}}$ to get z.) In this case, the probability that a uniform random string falls into the "bad" set that z is useful for inverting is negligible. Hashing the random bits will then break the delicate connection between the backdoor and the function index that will be generated using the cleaned randomness. Specifically, it will be challenging for the adversary to design an efficient connection between a backdoor z and the "scrambled" sets of functions backdoored by z.

Theorem 3.10. *The OWF family $\mathcal{G}$ described above is strongly unforgeable in the split-program model if H is modeled as a random oracle.*

Proof. First, the hash function h takes the output distribution of RG as input distribution; thus following corollary 2.3 we can still think it as a random oracle.

Next, we will argue the following: assume $\mathcal{G}$ is δ-forgeable in the split program model, i.e., there exist PPT adversaries $(\mathcal{A}_{\mathsf{RG}}, \mathcal{A}_{\mathsf{dKG}}, \mathcal{A}_{\mathsf{Inv}})$ satisfying Definition 3.9. Then we can construct a simulator $\mathcal{S}$ that breaks the one-way security of $\mathcal{F}$ in the split-program model.

Suppose r^* is the randomness and $y = f_i(x)$ is the challenge value (for a randomly chosen x) received from the one-way security challenger $\mathcal{C}$, where $i = \mathsf{dKG}(r^*)$.

$\mathcal{S}$ first chooses a random bit b and runs $\mathcal{A}_{\mathsf{RG}}$, which asks random oracle queries $\{r_1^0, \ldots, r_{q_0}^0\} := Q_0$. If $b = 0$, $\mathcal{S}$ randomly selects $t_0 \in \{1, \ldots, q_0\}$, and answers $h(r_{t_0}^0) = r^*$; all other queries are answered with uniform ℓ-bit strings. If $b \neq 0$, $\mathcal{S}$ answers all queries with random strings. $\mathcal{S}$ maintains a list for the query-answer pairs.

If $[b = 0 \wedge r \neq r_{t_0}^0]$, $\mathcal{S}$ aborts; otherwise, it sets $h(r) = r^*$ and it runs $\mathcal{A}_{\mathsf{KG}}$ with inputs r^*, and expects $\mathcal{A}_{\mathsf{KG}}$ to output i. $\mathcal{S}$ then runs $\mathcal{A}_{\mathsf{Inv}}$ with inputs (i, y, z) and receives the response x'.

If $b = 1 \wedge r \in Q_0$, $\mathcal{S}$ aborts; otherwise, it sends x' to the challenger $\mathcal{C}$ as his answer.

Similar to the proof of Theorem 3.5, let W denote the event that $\mathcal{S}$ aborts: we can bound $\Pr[\overline{W}] \leq 1 - \frac{1}{q_0}$. Thus,

$$\Pr[x' = x] \geq \Pr\left[x' = x \,|\, \overline{W}\right] \Pr\left[\overline{W}\right] \geq \frac{\Pr\left[x' = x \,|\, \overline{W}\right]}{2q_0}.$$

While following Definition 3.9,

$$\Pr[x' = x|\overline{W}] = \Pr[\mathcal{A}_{\mathsf{Inv}}(z, i, f_i(x)) = x] \geq \delta.$$

To summarize, in the split-program model, if $\mathcal{G}$ is δ-forgeable, $\mathcal{S}$ will break the one way security of $\mathcal{F}$ with probability at least $\delta/(2q_0)$. □

Constructing unforgeable TDOWFs in the split-program model. More interestingly, we can apply the same method to immunize a TDOWF in the split-program model, whose $\mathsf{RG}_{\text{SPEC}}$ outputs uniform bits r and whose (deterministic) $\mathsf{dKG}_{\text{SPEC}}$, given r, outputs a function index and a trapdoor pair. However, in this case, we have to assume the implementation RG to be stateless. The detection condition can guarantee that the output of RG is unpredictable to the adversary. Then, similar to the OWF case, hashing the randomness yields a uniform bitstring, and it ensures that the resulting function index together with the corresponding trapdoor will be "safe." [8] As pointed out in Section 3.1, it is preferable to consider strong forgeability for TDOWF (since a forgeable TDOWF can be seen as itself), thus we adapt Definition 3.2 to the split-program model: [9]

[8]It is easy to see that if RG is stateful and it maintains a counter state, ctr, there is a simple attack that $\mathsf{RG}(z, ctr)$ runs $\mathsf{PRF}_z(ctr)$, where z is the backdoor and used as the seed for the PRF. In this case, the output of RG is completely known to the adversary, and also the trapdoor corresponding to the outputted function index.

[9]Again, since dKG is deterministic with public input distributions, for simplicity we consider they are honestly implemented as $\mathsf{dKG}_{\text{SPEC}}$, even though they are implemented by the adversary.

Definition 3.11. *A trapdoor one-way function family $\mathcal{F} = \{f_i : X_i \to Y_i\}_{i \in I}$ (with the specification algorithms $\mathsf{RG}_{\text{SPEC}}, \mathsf{dKG}_{\text{SPEC}}$) is δ-**strongly forgeable** if there exist* PPT *adversarial algorithms $\mathcal{A} = (\mathsf{BG}, \mathsf{RG}, \mathsf{dKG}, \mathsf{Inv})$ so that given a backdoor z produced by backdoor generation algorithm BG, the random string generation algorithm RG generates an $\ell(\lambda)$-bit random string r, and based on such r the deterministic function generation algorithm dKG generates a function index i and the corresponding trapdoor t_i, with the following properties: (1) f_i is invertible given z without providing t_i, and (2) the output distribution of RG is computationally indistinguishable from that generated by the specification algorithms $\mathsf{RG}_{\text{SPEC}}$. That is, for every $z \leftarrow \mathsf{BG}(\lambda)$, it holds that:*

(1) $\Pr[x' = x \mid z \leftarrow \mathsf{BG}; r \leftarrow \mathsf{RG}(\lambda, z); (i, t_i) \leftarrow \mathsf{dKG}_{\text{SPEC}}(\lambda, r); x \leftarrow X_i; y := f_i(x); x' \leftarrow \mathsf{Inv}(i, y, z)] \geq \delta$

(2) $\{r \mid z \leftarrow \mathsf{BG}; r \leftarrow \mathsf{RG}(\lambda, z)\} \overset{c}{\approx} \{r \mid r \leftarrow \mathsf{RG}_{\text{SPEC}}(\lambda)\}$

Correspondingly, we say a TDOWF family is **unforgeable** *if it is not δ-strongly forgeable for any non-negligible δ.*

The immunization methodology for OWF discussed before can also be used for immunizing stateless TDOWF in the split program model. That is, given any TDOWF family $\mathcal{F}$ whose $\mathsf{RG}_{\text{SPEC}}$ outputs uniform bits r, a hash function h randomly selected from a hash family $H_\lambda : \{0,1\}^\lambda \to \{0,1\}^\lambda$ can be used to determine an unforgeable TDOWF family $\mathcal{G}$, see Figure 5 below:

$$\boxed{\mathsf{RG}(z)} \to r \qquad r \to \boxed{h} \to \tilde{r} \qquad \tilde{r} \to \boxed{\mathsf{dKG}} \to (i, t_i) \qquad x \to \boxed{\mathsf{Eval}(i, \cdot)} \to y$$

Figure 5: Immunization strategy for TDOWF in the split-program model.

Remark 3.12. *We can consider (h, dKG) as one single deterministic algorithm $\mathsf{dKG} \circ h$ which first run h on inputs r to get $\tilde{r}$ and run dKG on $\tilde{r}$. The new specification will be $(\mathsf{RG}_{\text{SPEC}}, \mathsf{dKG}_{\text{SPEC}} \circ h_{\text{SPEC}})$.*

Theorem 3.13. *The TDOWF family $\mathcal{G}$ described above is unforgeable in the split-program model if H is modeled as a random oracle, and the adversarial implementations are stateless.*

First, since the output distribution of a stateless RG is pseudorandom, thus for any PPT adversary $\mathcal{A}$, including the one that produces RG, $\Pr[r' = r : r \leftarrow \mathsf{RG}(z), r' \leftarrow \mathcal{A}(z)] \leq \epsilon$, where $\epsilon = \mathsf{negl}(\lambda)$.

The rest proof is similar to the proof of Theorem 3.10 that if one can break the strong unforgeability of $\mathcal{G}$, then we can construct a simulator that breaks the specification of $\mathcal{F}$ in the split-program model.

4 Pseudorandom Generator in the Complete Subversion Model

As mentioned in the Introduction, our goal is to stimulate a *systematic* study of Cliptography. Having studied the fundamental backdoor-free building blocks, unforgeable OWFs, and unforgeable TDOWFs, we intend to mimic the classic footprints of constructing cryptographic primitives from OWF/TDOWF, and provide solutions to other important backdoor-free building blocks. As our first example, we will show an interesting connection between our notion of unforgeable OWF and the notion of backdoor-free PRG, recently studied by Dodis et al. [8]. Next we first review the basic notions of PRG under subversion attacks. We then provide a specific solution based on the Blum-Micali PRG; this result can be viewed as a mimic of the classic result of Blum-Micali construction in cliptography. Furthermore, we examine how to extend the applicability of our general sanitizing strategy for OWF/TDOWF to more settings. We will demonstrate a general method of public immunizing strategy for PRG. We remark that, all algorithms in our backdoor-free PRG construction, including the sanitizing function (which can be part of the KG algorithm in the specification), can be subverted. Thus we provide the first PRG constructions secure in the complete subversion model.

4.1 Preliminaries: backdoored and backdoor-free PRGs

We adopt the definition from [8], that a pseudorandom generator consists of a pair of algorithms $(\mathsf{KG}, \mathsf{PRG})$, where KG outputs a public parameter pk and $\mathsf{PRG} : \{0,1\}^* \times \{0,1\}^\ell \to \{0,1\}^\ell \times \{0,1\}^{\ell'}$ takes the public parameter pk and an ℓ-bit random seed s as input; it returns a state $s_1 \in \{0,1\}^\ell$ and an output string $r_1 \in \{0,1\}^{\ell'}$. PRG may be iteratively executed; in the i-th iteration, it takes the state from the previous iteration s_{i-1} as the seed and generates the current state s_i and output r_i. We use PRG^q to denote the result of q iterations of PRG with outputs $r_1, \ldots, r_q$ (each $r_i \in \{0,1\}^{\ell'}$).

In a backdoored PRG, the algorithms (in particular KG) are implemented by the adversary (represented by $\mathcal{A}_{\text{INITIAL}}$), outputs a public parameter pk together with a backdoor sk. The output distribution $\mathsf{PRG}(pk, \mathcal{U})$ is still pseudorandom, where $\mathcal{U}$ is the uniform distribution; however, with the corresponding backdoor sk, the adversary (represented by $\mathcal{A}_{\text{DIST}}$) is able to break the PRG security (e.g., the adversary can distinguish the output from a uniform string).

We will rephrase the definition for backdoored PRG in our setting—as in the definition of a forgeable OWF—there exist "specification" versions of the algorithms. In particular, the parameter generation algorithm $\mathsf{KG}_{\text{SPEC}}$ is with the requirement that the distribution of the adversarially generated public parameter must be indistinguishable from the output distribution of $\mathsf{KG}_{\text{SPEC}}$. It is easy to see that the output distribution of $\mathsf{PRG}(pk, s)$ for a uniformly chosen s is pseudorandom even pk is generated by $\mathcal{A}_{\text{INITIAL}}$. (Otherwise, one can easily distinguish the output distribution of $\mathcal{A}_{\text{INITIAL}}$ from $\mathsf{KG}_{\text{SPEC}}$.) Additionally, as the PRG algorithm is deterministic, and its input distribution is public, we may assume that the adversary implements it honestly as the $\mathsf{PRG}_{\text{SPEC}}$ to avoid easy detection so that we can focus on the KG algorithm. The formal definitions are presented as follows:

Backdoored PRG. We re-phrase the definition of (q, δ)-**backdoored PRG** [10] as follows: We define a backdoored PRG game (see Figure 6) with a PPT adversary $\mathcal{A} = (\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{DIST}})$ such that (i) the pk distribution is indistinguishable from that generated by $\mathsf{KG}_{\text{SPEC}}$; and (ii) the adversary wins the backdoored PRG game with probability δ, i.e., $\Pr[b = b'] - \frac{1}{2} \geq \delta$.

```
$(pk, sk) \leftarrow \mathcal{A}_{\text{INITIAL}}$
$s \leftarrow \{0,1\}^\ell$
$r_1^0, \ldots, r_q^0 \leftarrow \mathsf{PRG}^q(pk, s)$
$r_1^1, \ldots, r_q^1 \leftarrow \{0,1\}^{\ell' \cdot q}$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}_{\text{DIST}}(pk, sk, r_1^b, \ldots, r_q^b)$
```

Figure 6: The backdoored PRG game

Backdoor-free PRG. Then we say that a PRG is q-**backdoor free** if, in the above backdoored PRG game, for all PPT adversaries $\mathcal{A} = (\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{DIST}})$, whenever pk is indistinguishable from the specification distribution, the advantage is negligible, i.e., $\left|\Pr[b = b'] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$.

Remark 4.1. *The generation of the seed s is out of the scope of this paper (same as [8]), since even the specification of pseudorandom generators do not cover this part. Our techniques can guarantee a received implementation is as good as the specification. Of course, it would be an important open question to consider the random seed generation for practice.*

[10] We ignore the running time of the adversary here for simplicity. Also, according to the detection principle in section 2, lemma 2.1, the PRG algorithm is deterministic and with a uniform input distribution, thus we can treat it as honestly implemented.

4.2 Constructing backdoor-free PRG from strongly unforgeable OWP

In this subsection, we provide constructions for backdoor-free PRG based on strongly unforgeable one-way permutation. We start with a basic solution based on a (simplified) Blum-Micali PRG, and then extend it to a full-fledged solution. Before going to the details of our constructions, we recall the classic generic construction of Goldreich-Levin (GL), yielding a hardcore predicate [9] for any OWF f. We suppose the input x of f is divided into two halves $x = (x_1, x_2)$ and define the bit $B(x) = \langle x_1, x_2 \rangle$; $B(x)$ is hard to predict given $x_1, f(x_2)$, assuming that f is one-way. Moreover, if there is a PPT algorithm that predicts $B(x)$ with significant advantage δ given $x_1, f(x_2)$, then there is a PPT algorithm I that inverts f with probability $\mathrm{poly}(\delta)$.

Basic construction. We will show that given a strongly unforgeable one-way permutation (OWP) family $\mathcal{F}$ with algorithms $(\mathsf{KG}_{\mathcal{F}}, \mathsf{Eval}_{\mathcal{F}})$, the classic Blum-Micali PRG [5] (using the GL hardcore predicate) is 1-backdoor free. Our basic construction $(\mathsf{KG}, \mathsf{PRG})$ is as follows:

- Parameter generation algorithm $pk \leftarrow \mathsf{KG}(\lambda)$:

  compute $i \leftarrow \mathsf{KG}_{\mathcal{F}}(\lambda)$ and set $pk := i$;

- Bit string generation algorithm $(s', b) \leftarrow \mathsf{PRG}(pk, s)$:

  upon receiving s and pk, where $pk = i$, $s = s_1||s_2$ and $|s_1| = |s_2| = \ell$, compute the following: $s'_1 := s_1$, $s'_2 := f_i(s_2)$ (or $s'_2 := \mathsf{Eval}_{\mathcal{F}}(i, s_2)$), and $s' = s'_1||s'_2$, $b := \langle s_1, s_2 \rangle$.

We can show in the lemma below that the basic construction above is a 1-backdoor free PRG. The intuition is that in the (simplified) Blum-Micali PRG, a distinguisher can be transformed into an OWF inverter (following the GL proof), thus an adversary who can build a backdoor for this PRG implies that she has the ability to make $\mathcal{F}$ (forgeable), which violates the strong unforgeability of $\mathcal{F}$.

Lemma 4.2. *Given a strongly unforgeable one way permutation family $\mathcal{F}$, the basic construction above is 1-backdoor free.*

Proof. The specification $\mathsf{KG}_{\text{SPEC}}$ of the simplified Blum-Micali PRG outputs a random function index from the corresponding index set (by simply running the key generation specification of $\mathcal{F}$).

First, it is easy to see that the OWF function family $\mathcal{G} = \{g_i\}$, where $g_i(x_1||x_2) := x_1||f_i(x_2)$, is strongly unforgeable if $\mathcal{F}$ is strongly unforgeable.

If the above basic construction is a $(1, \delta)$-backdoored PRG for some non-negligible δ, there exist PPT adversaries $(\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{DIST}})$ such that (i.) $\mathcal{A}_{\text{INITIAL}}$ can output a pair (pk, sk), where pk is indistinguishable from a randomly sampled public parameter; and (ii.) $\mathcal{A}_{\text{DIST}}$, with the backdoor sk, can distinguish $s_1||f_i(s_2)||B(s)$ (where $B(s) = \langle s_1, s_2 \rangle$) from a uniform $(2\ell + 1)$-bit string. It is not hard to see that we can then construct algorithms $(\mathsf{KG}, \mathsf{Inv})$ that break the strong unforgeability of $\mathcal{G}$.

In particular, KG runs $\mathcal{A}_{\text{INITIAL}}$ and outputs the received key pair (pk, sk), where pk here corresponds to a function index i that is indistinguishable from a random index, and sk corresponds to the backdoor z. Inv receives a challenge $y = g_i(x)$ and the backdoor z; it first constructs an algorithm $\mathcal{A}_P$. $\mathcal{A}_P$ selects a random bit b, and runs $\mathcal{A}_{\text{DIST}}(pk, sk, y||b)$. By the definition of $\mathcal{A}_{\text{DIST}}$, if $b = B(s)$, $\mathcal{A}_{\text{DIST}}$ (with sk) will output 0 with probability $1/2 + \delta$. It is easy to see that $\mathcal{A}_P$ can predict the GL hardcore predicate B with advantage $\delta/2$, following the GL proof [9], there exists another algorithm $I^{\mathcal{A}_P}(pk, sk, \cdot)$ that can invert y with probability $\delta' = \mathrm{poly}(\delta/2)$. Inv runs $I^{\mathcal{A}_P}(pk, sk, i, y)$ and recovers x'; Inv then outputs x' if it is a valid pre-image of y. It follows that $\Pr[x' = x] \geq \mathrm{poly}(\delta/2) = \delta'$ and $\mathcal{G}$ will be δ'-forgeable for a non-negligible δ', thus contradicts our assumption. $\square$

Full-fledged construction. We now extend our basic construction via iterations to show that the full fledged Blum-Micali PRG construction, using our strongly unforgeable OWF, achieves a q-backdoor free PRG for any $q = \text{poly}(\lambda)$. Our full-fledged construction $(\mathsf{KG}, \mathsf{PRG})$ [11] is as follows:

- Parameter generation algorithm $pk \leftarrow \mathsf{KG}(\lambda)$:
  compute $i \leftarrow \mathsf{KG}_{\mathcal{F}}(\lambda)$ and set $pk := i$;
- Bit string generation algorithm $(s', r) \leftarrow \mathsf{PRG}(pk, s)$:
  upon receiving s and pk where $pk = i$, $s = s_1||s_2$, and $|s_1| = |s_2| = \ell$, compute the following:
  - let $s_1^0 := s_1$ and $s_2^0 := s_2$;
  - for $j = 1, \ldots \ell'$,
    $b_j := \langle s_1^{j-1}, s_2^{j-1} \rangle$;
    $s_1^j := s_1^{j-1}$; $s_2^j := f_i(s_2^{j-1})$; $s^j := s_1^j||s_2^j$;
  - $s' = s^{\ell'} = s_1||f_i^{\ell'}(s_2)$; and $r = b_1 \ldots b_{\ell'}$.

  Please see Figure 7 for pictorial illustration.

$$\begin{tikzcd}
s_1||s_2 \arrow[r] & s_1||f_i(s_2) \arrow[r] \arrow[d] & s_1||f_i^2(s_2) \arrow[r] \arrow[d] & \ldots \arrow[r] & s_1||f_i^{\ell'}(s_2) \arrow[r] \arrow[d] & \ldots \\
 & \langle s_1, s_2 \rangle & \langle s_1, f_i(s_2) \rangle & \ldots & \langle s_1, f_i^{\ell'-1}(s_2) \rangle &
\end{tikzcd}$$

Figure 7: One iteration of BM-PRG

Theorem 4.3. *The full fledged construction above is q-backdoor free (for any polynomially large q), if the underlying OWP family $\mathcal{F}$ is strongly unforgeable.*

Proof sketch. Following Lemma 4.2, $s_1||f_i^j(s_2)||b_j$ is pseudorandom, i.e., it is indistinguishable from "$u_1, \ldots, u_\ell, v_1$", even to the adversaries $(\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{DIST}})$ who set i, where $\{u_i\}, v_1$ are all random bits. While b_{j+1} is only related to $s_1, f_i^j(s_2)$, it follows that to the adversary $\mathcal{A}_{\text{DIST}}$ (who has the backdoor), $b_1 \ldots b_{\ell'}$ satisfies the next-bit unpredictability. Following the classic reduction from pseudorandomness to next-bit unpredictability, we can conclude that $b_1 \ldots b_{\ell'}$ is indistinguishable from uniform bits from $\{0, 1\}^{\ell'}$, even to $\mathcal{A}_{\text{DIST}}$. (This can be shown via the hybrid argument.) Then, inductively, we can conclude that $r_1, \ldots, r_q$ are indistinguishable from $\ell' \cdot q$ uniform bits. $\square$

Remark 4.4. *It is easy to see that if starting from an OWF(not necessarily unforgeables), the full fledged construction can be easily modified by replacing $f_i(\cdot)$ with $f_{h(i)}(\cdot)$.*

4.3 General public immunization strategy for PRG

An impossibility result about public immunization of a backdoored PRG was presented in [8]. However, we observe that this impossibility result only applies to an immunization procedure that operates on the *output* of the backdoored PRG . The application of strongly unforgeable OWF to backdoor-free PRG shown above inspires us to consider a new, general immunizing strategy for backdoored PRG . We suggest

[11] PRG^q can be defined in a straightforward manner that runs PRG for q iterations, each iteration outputs ℓ' bits and updates the state for next iteration.

that—similar to the procedure above for eliminating backdoors in OWFs—one can randomize the public parameters to sanitize the PRG. [12] The intuition for this strategy to be effective in the setting of PRG is similar: if a specification $\mathsf{KG}_{\text{SPEC}}$ that outputs a uniform pk from its domain, no single backdoor can be used to break the security for large amount of public parameters; otherwise, one can use this trapdoor to break the PRG security of the specification.

Consider a PRG implementation $(\mathsf{KG}, \mathsf{PRG})$ (in which the KG algorithm might be backdoored), which is proven secure if the $\mathsf{KG}_{\text{SPEC}}$ outputs uniformly from its range PP. Let h be randomly selected from a hash family $H : PP \to PP$ which is modeled as a random oracle. Then we can construct a backdoor-free PRG, $(\mathsf{KG}, \mathsf{PRG} * h)$, i.e., applying the hash to the given public parameter to derive the actual pk that will be fed into the PRG algorithm (the new deterministic pseudo randomness generation algorithm $\mathsf{PRG} * h$ defined as $\mathsf{PRG} * h(pk, s) = \mathsf{PRG}(h(pk), s)$). Note that in order for this method to work, we must insist that pk can not be null, and is indeed used by the PRG algorithm, as in the case of, e.g., Dual_EC PRG. Also see the pictorial illustration in Fig 8.

$$\boxed{\mathsf{KG}} \to pk \qquad pk \to \boxed{h} \to \widetilde{pk} \qquad s \to \boxed{\mathsf{PRG}(\widetilde{pk}, \cdot)} \to r$$

Figure 8: Public immunization strategy for PRG.

Theorem 4.5. *Assume* $(\mathsf{KG}_{\text{SPEC}}, \mathsf{PRG}_{\text{SPEC}})$ *is a pseudorandom generator if* $\mathsf{KG}_{\text{SPEC}}$ *outputs pk randomly from its domain. Given any implementation* $(\mathsf{KG}, \mathsf{PRG})$*, hashing the public parameters as described above, i.e.,* $(h \circ \mathsf{KG}, \mathsf{PRG}_{\text{SPEC}})$ *yields a q-backdoor-free pseudorandom generator in the random oracle model for any polynomially large q.*

Proof sketch. Suppose $(h \circ \mathsf{KG}, \mathsf{PRG}_{\text{SPEC}})$ is a (q, δ)- backdoored PRG . Then there exist a pair of adversaries $(\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{DIST}})$ that can win the backdoored PRG game defined in Figure 6 with advantage δ. We will transform these adversaries into an adversary $\mathcal{S}$ that breaks the security of $(\mathsf{KG}_{\text{SPEC}}, \mathsf{PRG}_{\text{SPEC}})$.

Suppose the challenger of the specification version of PRG selects the parameter pk^* and the challenge string r^* is either $\mathsf{PRG}(pk^*, s)$ (for a uniform $s \in \{0,1\}^{\ell}$) or a uniform string from $\{0,1\}^{\ell' \cdot q}$ for some ℓ'.

The reduction follows the proof of Theorem 3.5: $\mathcal{S}$ attempts to embed pk^* into the answers to the random oracle queries. In particular, if $\mathcal{A}_{\text{INITIAL}}$ outputs pk, $\mathcal{S}$ wishes to answer the random oracle query about pk using pk^*, i.e., $h(pk) = pk^*$. We then proceed with a similar probabilistic analysis.

It is easy to see that if $\mathcal{A}_{\text{DIST}}$ can distinguish r^* from an uniform string, then $\mathcal{S}$ will be able to distinguish the output of the specification version from random which violates the PRG security of the specification. □

Remark 4.6. *There are several points we would like to stress:*

- *If the public parameter contains only random elements from a group, e.g., the Dual_EC PRG, we may simply encode them into bits and use the regular hash functions like SHA-256, directly and convert the resulting bits back to a group element;*

- *If the public parameters are structured elements, or the* $\mathsf{KG}_{\text{SPEC}}$ *does not output a uniform distribution, we can work on the split-program model that forces the adversarial implementation to explicitly isolate its generation of randomness, and make the randomness public as part of the public parameter.*

[12]To interpret this results, since the solution of [8] requires a trusted seed/key generation and apply the function to the PRG output, thus part of the PRG algorithm can not be subverted. It follows that the construction of PRG in the complete subversion model was still open until our solution. In contrast, our sanitizing strategy does not require any secret, and even the deterministic hash function can be implemented by the adversary as part of the KG algorithm.

- *If we treat the immunizing method as part of the* KG *algorithms, i.e.,* $h \circ \mathsf{KG}$ *is a single algorithm, we can let the adversary sets both* $(pk, \widetilde{pk})$ *as the public parameter, regular user simply uses* $\widetilde{pk}$ *as the actual parameter and the crypto experts can check the validity of it. Similarly, in the split-program model, we can let the big brother set the original randomness* r *and sanitized randomness* $\tilde{r}$*, together with* $\widetilde{pk}$ *as the public parameters.*

5 Signatures in the Complete Subversion Model

As an immediate application, in this section, we will demonstrate how to use unforgeable OWFs and unforgeable TDOWFs as fundamental building blocks to construct backdoor-free digital signature schemes in the complete subversion model.

We build on recent works of Bellare et al. [2, 3], who studied a family of *algorithm-substitution attacks* (ASAs) on cryptographic primitives. In their ASAs, an encryption algorithm is implemented by a saboteur who may have backdoors embedded; then the sabotaged implementation judiciously samples randomness in order to generate the outputs (e.g., ciphertexts) in a way that leaks the secret bit-by-bit via a steganographic channel.

To defend against the ASAs, Bellare et al provided successful mechanisms of using deterministic algorithms (e.g., signing) with unique outputs. However, we note that the ASAs considered in [2, 3] are restricted in the sense that the adversary is not allowed to launch the ASAs on *all* algorithms; In particular, the key generation algorithm is assumed to be implemented honestly and the adversary is not allowed to attack on it.

We are interested in securing cryptographic schemes in the *complete subversion model* where the adversary is allowed to launch algorithm-substitution attacks on *all* components of the schemes. We demonstrate below how our unforgeable TDOWF can be applied to defend against the (stateless) ASAs, even if the key generation algorithm has been sabotaged. Together with the results of [2, 3], we then show a concrete example of a digital signature scheme, achieving security in the case that *all* the algorithms are sabotaged.

Note that our subversion-secure signature scheme will require that the KG implementation to be stateless, and the attack of Bellare et al. implicitly assumes that the sabotaged implementations can persistently maintain a state from one cryptographic call to the next. To justify our result in the complete subversion model, we generalize the ASAs considered in [2, 3] to the stateless setting, that even the implementations are not allowed to maintain states, the (randomized) adversarial implementations can still build a steganographic channel to leak a secret subliminally.

5.1 Subversion Attacks in the Stateless Setting

In [2, 3], Bellare et al studied a family of *algorithm-substitution attacks* (ASAs) on cryptographic schemes, where certain algorithm is implemented by a saboteur who may have backdoors embed, and the subverted implementation, by maintaining state, generate well-structured outputs in a way that the secret can be leaked via a steganographic channel. We consider here the *stateless* ASAs, where the malicious implementations do not maintain state. While stateless ASAs are weaker than those considered in [2, 3], we next demonstrate the evidence that these attacks are very powerful.

Consider a randomized (cryptographic) algorithm R (e.g., encrypting, signing, etc.) which takes a random string r and a message x as inputs, and outputs an n-bit string $y = y_1, \ldots, y_n$ (from the output distribution C, which we assume has high entropy). Suppose s is the secret that R wants to leak covertly with length ℓ; we let z, k denote embedded backdoors—that is, random data known to the saboteur—which are uniform seeds. We let Ext be a randomness extractor that takes a string with enough entropy and a seed as inputs, and outputs a (nearly-) uniform string. Finally, let $\mathbf{C} = C_1, \ldots, C_N$ be the public outputs collected, for $N = \Omega(\ell \log \ell)$. We let Int denote the function that converts a binary string to an integer, and Maj denote the function that outputs the majority of a set containing bits. We describe a simple stateless ASA in Figures 9.

```
Sender algorithm:
$t = 0$
$r \leftarrow \{0,1\}^\lambda$
$c_1, \ldots, c_n \leftarrow R(r, x)$
$y_1, \ldots, y_m \leftarrow \mathsf{Ext}(k, c_1, \ldots, c_n)$
$j \leftarrow \mathsf{Int}(y_1 \ldots y_\ell)$,
$\tilde{y} \leftarrow y_{\ell+1}, \ldots, y_m$
If $s_j \neq \mathsf{PRF}(z, \tilde{y}) \wedge t = 0$:
  $t{+}{+}$;
Else:
  Return $c_1, \ldots, c_n$
```

```
Receiver algorithm:
For $j = 1, \ldots, \ell$ : $Q_j = \emptyset$
For $i = 1, \ldots, N$, do
  $y_1^i, \ldots, y_m^i \leftarrow \mathsf{Ext}(k, C_i)$
  $j \leftarrow \mathsf{Int}(y_1^i \ldots y_\ell^i)$
  $\tilde{y} \leftarrow y_{\ell+1}^i, \ldots, y_m^i$
  If $\mathsf{PRF}(z, \tilde{y}) = b$ : $Q_j = Q_j \cup \{b\}$
For $j = 1, \ldots, \ell$:
  $s_j \leftarrow \mathsf{Maj}(Q_j)$
Return $s_1, \ldots, s_\ell$
```

Figure 9: Stateless ASA

It is easy to see that for each invocation of the algorithm R, it either outputs a random draw from the output range directly, or draws twice and outputs the second. A remarkable feature of such a "1-rejection sampling" algorithm is that—if PRF were a perfectly random function—the resulting distribution is identical to the original distribution on C (cf. [11]). As PRF is merely indistinguishable from a random function, we have:

Claim 5.1. *The output distribution in the stateless attack shown in Figure 9 is indistinguishable from the regular output distribution of R.*

Moreover, we can see that each output corresponds to a random index of the secret, and helps the adversary to predict this particular bit of the secret s with probability close to $3/4$ (again, we here consider PRF to be a perfectly random function and assume that C has negligible collision probability). Thus, after collecting enough outputs of the subverted algorithm, the adversary can create a list of "votes" for each bit of s; when the adversary has collected sufficiently many such votes, he can recover the bit with high confidence by taking the majority.

Claim 5.2. *If the output distribution of R has min-entropy $d \geq \ell + O(\log \frac{1}{\epsilon})$, the receiver algorithm will return the correct value of s with probability $1 - \mathrm{poly}(1/\ell)$.*

Proof sketch. First, from the extractor property, the $y_1, y_2, \ldots, y_m$ may be treated as uniform bits (with negligible statistical distance that can be controlled as a function of the length of the backdoor seeds shared by the adversary and the implementation), then each C_i corresponds to a s_j for a uniformly chosen $j \in \{1, \ldots, \ell\}$.

Let us focus on each index j: after collecting $\Omega(\ell \log \ell)$ outputs of R, the number of outputs that correspond to j will be $\Omega(\log \ell)$ with high probability $(1 - 1/\mathrm{poly}(\ell))$. If PRF is treated as a random function and the collision probability of C is treated as zero, the rejection sampling procedure with provide a correct value of the jth bit with probability $3/4$: that is, $\Pr[s_j = H(\tilde{y})] \geq 3/4$. This is changed only negligibly by using PRF and C with sufficiently small collision probability). Then, following the standard Chernoff bound that s_j appears as majority will be at least $1 - \mathrm{poly}(1/\ell)$. And this is true for all the indices. $\square$

5.2 Digital signature scheme in the complete subversion model

We can define the existential unforgeability in the setting of complete subversion [13] in a way that the public key is generated using the adversarial implementation, and the signing/verification algorithm can also be implemented by the adversary, while all the other steps are the same as the standard definition.

Definition 5.3. *We say a signature scheme* $(\mathsf{KG}, \mathsf{Sign}, \mathsf{Verify})$ *(with the specification version of key generation algorithm* $\mathsf{KG}_{\text{SPEC}}$*, signing algorithm* $\mathsf{Sign}_{\text{SPEC}}$ *and verification algorithm* $\mathsf{Verify}_{\text{SPEC}}$*) is* ***existentially unforgeable in the complete subversion setting****, if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{FORGE}})$*, the following properties hold:*

- *The adversary* $\mathcal{A}$ *wins the unforgeability game defined below, with no more than a negligible probability.*

  UNFORGEABILITY GAME *between a* PPT *adversary* $\mathcal{A} = (\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{FORGE}})$ *and a challenger* $\mathcal{C}$*:*

  1. $\mathcal{A}_{\text{INITIAL}}$ *provides an implementation of* KG, Sign, Verify *(and an implementation of the hash function if needed) to the challenger (which may contain backdoor information* z*).*
  2. $\mathcal{C}$ *queries* KG *to learn the key pair* (pk, sk)*, and sends the public key* pk *to* $\mathcal{A}_{\text{FORGE}}$*.*
  3. $\mathcal{A}_{\text{FORGE}}$*, having the backdoor* z *from* $\mathcal{A}_{\text{INITIAL}}$*, asks signing queries for arbitrarily chosen messages* $m_1, \ldots, m_q$*; for each* m_i *from* $\mathcal{A}_{\text{FORGE}}$*, the challenger* $\mathcal{C}$ *queries* Sign *with input* (sk, m_i) *to learn the corresponding signature* σ_i*, and return* σ_i *to* $\mathcal{A}_{\text{FORGE}}$*.*
  4. $\mathcal{A}_{\text{FORGE}}$ *returns a message-signature pair* (m^*, σ^*) *to the challenger* $\mathcal{C}$*; now* $\mathcal{C}$ *queries* Verify *with input* (pk, m^*, σ^*) *to learn output* b*.*
  5. $\mathcal{A} = (\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{FORGE}})$ *wins the game if* $m^* \notin \{m_1, \ldots, m_q\}$ *and* $b = 1$*.*

- *In the above game, the subversion of* KG, Sign *and* Verify *(and the hash function if there is any) are strongly undetectable. (See section 2 for details.)*

Construction. Following the result of [3], if the key generation algorithm is honest, a unique signature scheme [10, 15] remains existentially unforgeable against the algorithm substitution attack. In this case, the signing and verification algorithms are both deterministic and the detection condition "forces" the adversary to honestly implement these two. To obtain a *complete* subversion-secure solution from a unique signature scheme, we still need to "upgrade" the key generation algorithm so that it is secure against subversion attack.

We may consider the following approach to obtain a subversion-secure key generation algorithm: initially generate an unforgeable OWF f, and then randomly select a secret key sk and compute the public key $pk = f(sk)$. Since the OWF is unforgeable, it seems that we may be able to force the adversarial key generation algorithm to output a "safe" OWF. However, it is not clear how to show that the key pairs are well distributed.

We here consider an alternative approach; we use an unforgeable TDOWF, $(\mathsf{KG}_{\mathcal{F}}, \mathsf{Inv}_{\mathcal{F}})$. More concretely, we use the key generation algorithm of unforgeable TDOWF to generate a function index i together with the corresponding trapdoor t_i, and set the index i as the public key, and the trapdoor t_i as the secret key. To be compatible with the unique signature scheme, we choose to instantiate the unique signature scheme from the full domain hash construction [4, 7]. Details of the construction are as follows:

[13] This definition may be weaker than the general definition of surveillance security defined in [3] that the output distribution of subverted algorithms looks the same as that of the specification even to the adversary who has the backdoor. However, preserving its standard security in this complete subversion model (we think) is good enough for each primitive to be used as normal.

- Key generation $(pk, sk) \leftarrow \mathsf{KG}(\lambda)$:

  compute $(i, t_i) \leftarrow \mathsf{KG}_{\mathcal{F}}(\lambda)$, and set $pk := i$ and $sk := t_i$;

- Signature generation $\sigma \leftarrow \mathsf{Sign}(sk, m)$:

  upon receiving message m, compute $\sigma = \mathsf{Inv}_{\mathcal{F}}(sk, i, h(m))$, where $sk = t_i$.

- Signature verification $b := \mathsf{Verify}(pk, m, \sigma)$:

  upon receiving message-signature pair (m, σ), if $f_i(\sigma_1) = h(m)$ then set $b := 1$, otherwise set $b := 0$; here $pk = i$.

Lemma 5.4. *The algorithms* $\mathsf{Sign}, \mathsf{Verify}$ *and the hash* h *in the full domain hash signature scheme are subversion resistant (in the strong detection model).*

Proof. Suppose $(m_1, y_1), \ldots, (m_q, y_q)$ and $(m_1, \sigma_1), \ldots, (m_{q'}, \sigma_{q'})$ form the transcript of the adversary in the subversion game, trying to distinguish $(\mathsf{Sign}, \mathsf{Verify}, h)$ from $(\mathsf{Sign}_{\text{SPEC}}, \mathsf{Verify}_{\text{SPEC}}, h_{\text{SPEC}})$. The transcript includes all the signing queries and the final forgery.

First, it is easy to see that if there is any $y_i \neq h_{\text{SPEC}}(m_i)$, then there exists a detector simply queries h_{SPEC} and noticed this inconsistency with probability 1.

If there exists $(m_i, (\sigma_i)$ such that $\mathsf{Sign}_{\text{SPEC}}(sk, m_i) \neq \mathsf{Sign}(sk, m_i) = \sigma_i$, then there exists a detecter algorithm $\mathcal{D}$ can always detect it with probability 1. $\mathcal{D}$ uses this pair as an input (part of the transcript) and runs the specification $\mathsf{Verify}_{\text{SPEC}}$ on this message-signature pair. Since the full domain hash is a unique signature, thus if they are not the same as that using the $\mathsf{Sign}_{\text{SPEC}}$, it will not pass the verification.

If there exists a pair (m_i, σ_i) such that $\mathsf{Verify}_{\text{SPEC}}(pk, m_i, \sigma_i) \neq \mathsf{Verify}(pk, m_i, \sigma_i)$, similarly, there exists a detecter $\mathcal{D}$ can always notice it.

It follows that every hash evaluation is consistent with that using the hash specification; every signature generated by the Sign algorithm is consistent with that generated by the specification; every signature pass the Verify algorithm is indeed a valid signature. If the adversary gains any advantage in the subversion game, there must be a pair that is inconsistent with the specifications, then it can be detected with probability 1. We can conclude that $\delta_s \leq \delta_d$. □

Remark 5.5. *The strong detection condition in the signature setting in practice is more reasonable than that in the symmetric key encryption setting as [3, 12]. Since the nature of signature schemes, the transcripts are for public verification anyway, anyone can verify the validity, including the lab. While in the symmetric key encryption, the detection requires the detecter to have access to the secret key. This essentially states that the every user is a detecter who has access to the specification, which weakens the motivation for considering the implementation subversion (since they have the specification anyway).*

Theorem 5.6. *Given an unforgeable TDOWF* $\mathcal{F}$*, the full domain hash signature scheme is existentially unforgeable (under complete subversion) in the random oracle model, assuming all the subverted implementations are stateless.*

Proof. First, following lemma 5.4, the $h, \mathsf{Sign}, \mathsf{Verify}$ algorithms we use are subversion resistant. It follows that the adversary behaves the same as sending honest implementations for those algorithms (otherwise got detected), i.e., consistent with the specifications. In particular, the hash function implementation can still be modeled as a random oracle for queries of the adversary.

We will show that if there is a subversion attack, then one can break the unforgeability of $\mathcal{F}$. Suppose $(\mathcal{A}_{\text{INITIAL}}, \mathcal{A}_{\text{FORGE}})$ are the adversaries who break the existential unforgeability of the full domain hash signature scheme in the complete subversion model with a non-negligible probability δ, we will construct $\mathcal{S}$ to simulate the adversaries $(\mathsf{BG}, \mathsf{KG}, \mathsf{Inv})$ as in Definition 3.2.

BG first runs $\mathcal{A}_{\text{INITIAL}}$ to receive the backdoor z and the implementation KG_0 for the signature key generation. KG simply runs KG_0 and outputs a function index $i := pk$ and the corresponding trapdoor t_i. $\mathcal{S}$ discards t_i and sends pk to $\mathcal{A}_{\text{FORGE}}$. Suppose $y = f_i(x)$ is the challenge that Inv receives from the unforgeable TDOWF challenger, Inv (constructed by $\mathcal{S}$) will feed $\mathcal{A}_{\text{FORGE}}$ with z and runs it as follows to try to invert.

W.l.o.g, we assume $\mathcal{A}_{\text{FORGE}}$ asks a random oracle query for a message before she asks for the signing query, (if not, the simulator can ask instead of her), and $\mathcal{A}_{\text{FORGE}}$ asks the random oracle query for her final forgery m^*.

Suppose $\mathcal{A}_{\text{FORGE}}$ asks q random oracle queries $m_1, \ldots, m_q$. Inv randomly choose $j \in \{1, \ldots, q\}$, and answers the query with $y = h(m_j)$; He then chooses $q-1$ random elements $\sigma_1, \ldots, \sigma_{j-1}, \sigma_{j+1}, \ldots, \sigma_q$, and answers the random oracle queries as $h(m_k) = f_i(\sigma_k)$ for $k \neq j$. Inv maintains a list.

When $\mathcal{A}_{\text{FORGE}}$ asks a signing query m_k, if $m_k = m_j$, Inv aborts, otherwise, Inv checks the list and returns the corresponding σ_k. (Note that Sign algorithm can only have one output for each message, thus the above procedure perfectly simulates the signing oracle implemented by the subverted Sign algorithm). $\mathcal{A}_{\text{FORGE}}$ outputs m^*, σ^*. If $m^* \neq m_j$, Inv aborts, otherwise, Inv outputs σ^*.

Let us use W to denote the event that Inv aborts. Following the classic proof of security of the full domain hash signature scheme (e.g., [7]), $\Pr[W] \leq 1 - \text{poly}(\frac{1}{q})$, thus Inv successfully inverts with probability at least $\delta_0 = \text{poly}(\frac{1}{q})\delta$ (as we assume $\mathcal{A}_{\text{FORGE}}$ has δ advantage, and the Verify has to be honestly implemented to avoid detection). This means $\mathcal{F}$ is δ_0-forgeable, which contradicts our condition. □

6 Conclusion and Open Problems

We initiate the systematic study of defending mechanism against kleptographic attacks of cryptographic primitives when *all* algorithms are subject to the subversion, we call **cliptography**.

We start from the fundamental primitives of (trapdoor) one way functions. We formalize the notions of forgeable (trapdoor) OWF to capture the possibility of embedding backdoors, in particular, into the function generation algorithms, and show how to launch such attacks. More interestingly, we suggest a general sanitization method employing randomization of the function index to destroy the possibility of embedding backdoor information. To instantiate our method in practice, we propose a split-program model in which the function generation algorithm consists of two components, a randomized component RG and a deterministic component dKG; here, the first component RG generates a uniform random string for the second component dKG, and then the second component generates an index based on such random string. In such split-program model, we can directly apply our general method of immunizing one way function generation to the randomness generated by RG.

We then pursue the possibility of building cliptography from unforgeable (trapdoor) OWFs. In particular, we show how to construct a signature scheme and a pseudorandom generator that preserves its security in the complete subversion model. These are done by using our unforgeable trapdoor OWF and strongly unforgeable OWF as the key/parameter generation algorithm. Finally, we show how to apply our immunizing technique directly to the setting of PRG and present a general public immunizing strategy for PRG.

Many important problems remain open about defending against kleptogaphic attacks, and in general against mass surveillance. The immediate open questions left by our paper would be: to what extend the results and techniques developed in this paper can be used to build cliptography. In particular, can we construct other cryptographic primitives that preserves their security in the setting that all algorithms are subject to kleptographic attacks? There are also many other related open questions, to name a few: (1) how to sanitize the OWF generation in the standard model? (2) how to eliminate the subliminal channel in general? (3). Can we show some results in the complete subversion model so that we can preserve the security for multiparty protocols? and finally, (4) how to design an accountable mechanism for surveillance, such that key escrow is provided but not abused?

References

[1] G. Ateniese, B. Magri, and D. Venturi. Subversion-resilient signature schemes. *IACR Cryptology ePrint Archive*, 2015:517, 2015. 2, 3, 5

[2] M. Bellare and V. T. Hoang. Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 627–656. Springer, Apr. 2015. 1, 3, 18

[3] M. Bellare, K. G. Paterson, and P. Rogaway. Security of symmetric encryption against mass surveillance. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Aug. 2014. 1, 2, 3, 4, 5, 6, 18, 20, 21

[4] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 399–416. Springer, May 1996. 20

[5] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo random bits. In *23rd FOCS*, pages 112–117. IEEE Computer Society Press, Nov. 1982. 15

[6] S. Checkoway, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, H. Shacham, and M. Fredrikson. On the practical exploitability of dual EC in TLS implementations. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 319–335, 2014. 3, 8

[7] J.-S. Coron. On the exact security of full domain hash. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Aug. 2000. 20, 22

[8] Y. Dodis, C. Ganesh, A. Golovnev, A. Juels, and T. Ristenpart. A formal treatment of backdoored pseudorandom generators. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Apr. 2015. 1, 2, 3, 6, 13, 14, 16, 17

[9] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989. 2, 15

[10] S. Goldwasser and R. Ostrovsky. Invariant signatures and non-interactive zero-knowledge proofs are equivalent (extended abstract). In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 228–245. Springer, Aug. 1993. 20

[11] N. J. Hopper, J. Langford, and L. von Ahn. Provably secure steganography. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 77–92. Springer, Aug. 2002. 3, 19

[12] P. F. Jean Paul Degabriele and B. Poettering. A more cautious approach to security against mass surveillance. In *Fast Software Encryption 2015.*, 2015. 5, 21

[13] A. Juels and J. Guajardo. RSA key generation with verifiable randomness. In D. Naccache and P. Paillier, editors, *PKC 2002*, volume 2274 of *LNCS*, pages 357–374. Springer, Feb. 2002. 2

[14] J. Larson, N. Perlroth, and S. Shane. Revealed: The NSA's secret campaign to crack, undermine internet security. Pro-Publica, 2013. http://www.propublica.org/article/the-nsas-secret-campaign-to-crack-undermine-internet-encryption. 1

[15] A. Lysyanskaya. Unique signatures and verifiable random functions from the DH-DDH separation. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 597–612. Springer, Aug. 2002. 20

[16] I. Mironov and N. Stephens-Davidowitz. Cryptographic reverse firewalls. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Apr. 2015. 1, 3

[17] NIST. Special publication 800-90: Recommendation for random number generation using deterministic random bit generators. National Institute of Standards and Technology, 2012. http://csrc.nist.gov/publications/PubsSPs.html. 3

[18] N. Perlroth, J. Larson, and S. Shane. N.S.A. able to foil basic safeguards of privacy on web. The New York Times, 2013. http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html. 1

[19] G. J. Simmons. The prisoners' problem and the subliminal channel. In D. Chaum, editor, *CRYPTO'83*, pages 51–67. Plenum Press, New York, USA, 1983. 3

[20] G. J. Simmons. A secure subliminal channel (?). In H. C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 33–41. Springer, Aug. 1986. 3

[21] A. Young and M. Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Aug. 1996. 1, 2, 6, 7

[22] A. Young and M. Yung. Kleptography: Using cryptography against cryptography. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, May 1997. 1, 2, 6, 7

A Omitted Proofs

Lemma 3.1. *A $(1-\epsilon)$-strongly forgeable OWF family is also a TDOWF family, where ϵ is a negligible function of the security parameter.*

Proof. Suppose $\mathcal{F} := (\mathsf{BG}_{\mathcal{F}}, \mathsf{KG}_{\mathcal{F}}, \mathsf{Inv}_{\mathcal{F}})$ is a $(1-\epsilon)$ strongly-forgeable OWF family. We can define $(\mathsf{KG}, \mathsf{Eval}, \mathsf{Inv})$ for the TDOWF family as follows. The generation algorithm KG is as follows: first run $\mathsf{BG}_{\mathcal{F}}(\lambda)$ and receive a string z; then run $\mathsf{KG}_{\mathcal{F}}$ with input (λ, z) and receive a function index i; finally output (i, z). The inversion algorithm Inv for the TDOWF is simply $\mathsf{Inv}_{\mathcal{F}}$, and the evaluation algorithm is defined as $\mathsf{Eval}(i, x) = f_i(x)$.

We can easily see that $(\mathsf{KG}, \mathsf{Eval}, \mathsf{Inv})$ is invertible once trapdoor is given: since $\mathcal{F}$ is a $(1-\epsilon)$ strongly-forgeable OWF family, it holds that by definition, $\Pr[\mathsf{Inv}_{\mathcal{F}}(z, i, y) = x \mid x \leftarrow X_i; y := f_i(x)] \geq 1-\epsilon$; therefore, $\Pr[\mathsf{Inv}(i, y, z) = x \mid x \leftarrow X_i; y := f_i(x)] \geq 1-\epsilon$.

We can also show the one-wayness: without z, no PPT algorithm can invert y for a random x; otherwise assume $(\mathsf{KG}, \mathsf{Eval}, \mathsf{Inv})$ is not one-way, then there exists an adversary $\mathcal{A}$ who for $i \leftarrow \mathsf{KG}_{\mathcal{F}}$, can invert $y := f_i(x)$ with non-negligible probability. We note that the specification function generation algorithm is one-way, i.e., for $i \leftarrow \mathsf{KG}_{\text{SPEC}}$, no one can invert $y := f_i(x)$ except negligible probability. Now, one can distinguish the output distribution of $\mathsf{KG}_{\mathcal{F}}(z)$ from the output distribution of $\mathsf{KG}_{\text{SPEC}}$, simply by trying inversion using $\mathcal{A}$. □

Lemma 3.2. *One can construct a* $(1-\epsilon)$*-strongly forgeable OWF from a TDOWF, where* ϵ *is a negligible function on the security parameter.*

Proof. Consider a TDOWF $\mathcal{F} = \{f_i\}$ with the associated algorithms $(\mathsf{KG}_{\mathcal{F}}, \mathsf{Inv}_{\mathcal{F}})$. Assuming the trapdoors can be represented using $\ell(\lambda)$ bits, we construct a strongly forgeable OWF family $\mathcal{G} = \{g_{i,r}\}$, where $g_{i,r}(x) = f_i(x)||r$ and $r \in \{0,1\}^{\ell(\lambda)}$.

The specification version of the sfunction generation algorithm $\mathsf{KG}_{\text{SPEC}}$ is defined as follows: run the $\mathsf{KG}_{\mathcal{F}}$ algorithm and receive a function index/trapdoor pair (i, t_i); then discard t_i and sample randomly $r \leftarrow \{0,1\}^{\ell(\lambda)}$; finally output (i, r). It is easy to see that $g_{i,r}$ is one way because f_i is one way (without t_i).

While for the backdoored implementations, BG first outputs a random key k for a symmetric key encryption scheme $\mathsf{SE} = (\mathsf{SE.Enc}, \mathsf{SE.Dec})$ which is assumed to be a pseudorandom permutation (PRP). $\mathsf{KG}(k)$ is defined as follows: it first runs $\mathsf{KG}_{\mathcal{F}}$, and receives an index i together with the corresponding trapdoor t_i; the second part $\tilde{r}$ is generated by encrypting t_i using k, i.e., $\tilde{r} = \mathsf{SE.Enc}(k, t_i)$. $\mathsf{KG}(k)$ outputs $(i, \tilde{r})$. ($g_{i,\tilde{r}}(x) = f_i(x)||\mathsf{SE.Enc}(k, t_i)$). It is easy to see that with the backdoor k, one can define the Inv as follows: it first decrypts $\tilde{r}$ using k to retrieve t_i, and then inverts $f_i(x)$ by running $\mathsf{Inv}_{\mathcal{F}}$ with t_i as an input.

Furthermore, since $\mathsf{SE.Enc}$ is modeled as a PRP, the distributions of (i, r) returned by $\mathsf{KG}_{\text{SPEC}}$ and $(i, \tilde{r})$ returned by $\mathsf{KG}(k)$ for any k are computationally indistinguishable. □