Generic Semantic Security against a Kleptographic Adversary

Alexander Russell
University of Connecticut
acr@cse.uconn.edu

Qiang Tang
New Jersey Institute of Technology
qiang@njit.edu

Moti Yung
Snap. Inc, & Columbia University
moti@cs.columbia.edu

Hong-Sheng Zhou
Virginia Commonwealth University
hszhou@vcu.edu

ABSTRACT

Notable recent security incidents have generated intense interest in adversaries which attempt to subvert—perhaps covertly—cryptographic algorithms. In this paper we develop (IND-CPA) Semantically Secure encryption in this challenging setting.

This fundamental encryption primitive has been previously studied in the "kleptographic setting," though existing results must relax the model by introducing trusted components or otherwise constraining the subversion power of the adversary: designing a Public Key System that is kletographically semantically secure (with minimal trust) has remained elusive to date.

In this work, we finally achieve such systems, even when all relevant cryptographic algorithms are subject to adversarial (kleptographic) subversion. To this end we exploit novel inter-component randomized cryptographic checking techniques (with an offline checking component), combined with common and simple software engineering modular programming techniques (applied to the system's black box specification level). Moreover, our methodology yields a strong generic technique for the preservation of any semantically secure cryptosystem when incorporated into the strong kleptographic adversary setting.

CCS CONCEPTS

• **Security and privacy → Public key encryption**;

KEYWORDS

Kleptography; Semantic security; Modular design

1 INTRODUCTION

Despite the remarkable advances in modern cryptography, applying cryptographic tools to provide robust security *in practice* is a notorious challenge. One implicit assumption in typical cryptographic modeling is that the implementations of cryptographic algorithms actually realize their "official specifications." In practice, however, implementations may diverge from their specifications for a variety of reasons, including programming bugs or malicious tampering. In particular, in the *kleptographic* setting, one considers the pathological possibility of *fully adversarial implementations* of cryptographic algorithms. The goal of such an adversary is to produce implementations of cryptographic algorithms which compromise security while appearing to be correct even in the face of fairly intensive (black-box) testing. The concrete possibility of these kleptographic attacks arise whenever a "third-party" software library or hardware device is relied upon for cryptographic purposes.

The consequences of such attacks are rather surprising: It turns out that—in wide generality—adversarial implementations of *randomized algorithms* may leak private information while producing output that is indistinguishable from the specification. The threat was first identified over two decades ago by Young and Yung [26, 27]. Recently, starting with [4], the topic has received renewed formal attention [1–3, 6, 10, 11, 17, 20] motivated by startling evidence from the Snowden whistleblowing revelations of past deployment of kleptographic attacks in the USA. One of the most striking recent discoveries [3, 4] establishes that a malicious side channel (also known as a subliminal channel or steganographic channel) can be embedded in the output of a subverted randomized (encryption) algorithm so that secret information can be *exclusively* leaked to the adversary via the ciphertext. Such kleptographic attacks can even be applied in settings where the subverted algorithms are stateless [3, 20].

This kleptographic setting features an adversary who may provide malicious implementations of intended cryptographic algorithms and modules. The minimal countermeasure for such strong attack is to perform testing by a trusted referee, hoping to ensure that the provided implementation has not been subverted. The trusted checking party (systematically explored, and called "watchdog" in [20]) may *test* the (adversarially-provided) implementation of each module against the specification—this testing can provide a measure of safety to the final users of the implementation. The major question is whether a combination of careful specification (a software engineering tool, in general) and testing (new cryptographic tools) can, in combination, preserve security despite such a powerful adversary. The broad challenge is to rework the classical framework of cryptographic primitives and constructions so that they provide security in this new environment.

Recent efforts have partially explored the landscape and clearly identified the critical role played by randomized algorithms in the kleptographic setting. Indeed, existing defense strategies roughly fall into the following three categories: (i). Systematically abandon randomized algorithms by adopting deterministic counterparts, e.g., use deterministic encryption with a unique ciphertext property (coupled with trusted key generation component) as suggested

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/http://dx.doi.org/10.1145/3133956.3133993

in [2–4, 6] for encryption schemes. (ii). Clipping the power of a subverted key generation by hashing the output [20] to preserve security for certain cryptographic primitives. (iii). Use a very powerful trusted on-line independent module to re-randomize all incoming/outgoing communication generated by malicious randomized algorithms [11, 17] (this notion is, in fact, adopted from the notion of "active warden" in the literature about subliminal channels [7], which was considered a much stronger assumption than a passive one). In turn each of the past defending strategy has some shortcomings (demonstrating the difficulty of the open question):

- While insisting on deterministic algorithms has favorable properties, it necessarily places many central notions of security entirely out of reach: in particular, IND-CPA (semantic) security for public-key encryptions is unattainable. Furthermore, the process of key generation is inherently stochastic; thus, it demands a setting where key generation is independent trusted method not under the adversary's control (to allow pseudorandomness to take care of security throughout).
- Hashing the output of the key generation indeed restrains the adversary's power, however, it does not destroy the subliminal channel (an explicit attack will be shown in section 3.5). The subverted algorithm can still leak some (at least one bit) information, thus, while being sufficient for other security purposes it does not solve the open problem: the IND-CPA security for public key encryptions is still out of reach.
- The reverse firewall (active warden) model can provide impressive feasibility results, but at the cost of assuming an active trusted component (in particular, it requires a source of trusted randomness not controlled by the adversary, which is not clear how we can obtain). Additionally, the reverse firewall model requires some "re-randomizable" structure of the scheme. (That means, such approaches cannot be generic.)

These recent results and the importance of semantic security, motivated us to consider the following question:

> *Can we compile any IND-CPA encryption scheme to the kleptographic setting with strictly weaker assumptions?*

We give an affirmative answer to the question. Our principal techniques are cryptographic testing, combined with modular programming methodology which specifies the cryptographic well defined functions and components to be designed and implemented separately, so that they can run independently. The former is a new procedure, whereas the latter is a regular software engineering requirement (modular programming where a straight line simple programs simply call these procedures in order). This configuration is further supported in reality and augmented by the availability of modern tools for isolated executions of various modules by various software architectures (virtual machines) as well as hardware architectures (e.g., Intel's SGX). We further remark that modularity is already part of crypto systems' own definition (e.g., a key generation module is separated from an encryption module and separated from a decryption module); we note further that in the anti subversion literature, *all* solutions employ separate modules. Brief description of the model and the split program techniques (as adopted from software engineering), see below.

Cliptographic model and split program methodology. We will mostly follow the definitional models from [20] that to capture the security against a kleptographic adversary: in particular, the adversary first supplies implementations of all components of the cryptographic functionalities. We say a cryptographic scheme is subversion resistant (clipto-secure), if *either* the security of the underlying cryptographic primitive can be preserved even if the potentially subverted implementations are used; *or* there is a watchdog that can detect the subversion via black-box testing on the components.

Moreover, the cliptographic model is pretty flexible to reflect some adjustment. Throughout the paper, we will use the module design principle and bring it into the algorithm level. Each algorithm may be decomposed into several functional components, and they will be executed independently and re-assembled correctly. [1] Sometimes, we may need one more basic trusted operation such as an addition (e.g., needed only for encryption of large size message). Those can be reflected in the model that the challenger will carry out all those trusted operations. In practice, they can be done by the user or the user's computing base. We remark that those modular design principle is implicit in all existing deployments, see Fig. 1. Also, see more detailed discussions in Sec. 3.1 and Appendix. A.

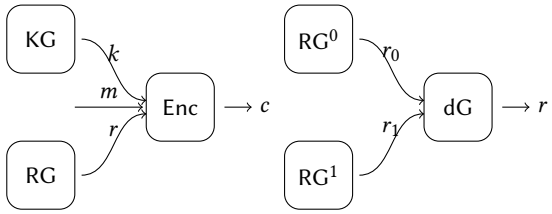


Figure 1: Modular design v.s. split program.

Our contributions. We construct symmetric-key and public-key encryption schemes preserving the semantic security when all the cryptographic algorithms (including key generation and randomness) are subject to subversion. To achieve these, we develop general techniques that completely eliminate subliminal channels introduced by malicious implementations. In more detail,

(1) We define a property of "stego-freeness" by adjusting the models of [4, 20] to characterize whether an implementation of an algorithm can be considered to be following its specification in the kleptographic setting. The notion will be a stepping stone on the way to our main result.
(2) We then consider defending against steganographic channel attacks by employing the simple module design principle. We demonstrate a "double-splitting" strategy in which randomness generation is carried out by two components $\mathrm{RG}_0, \mathrm{RG}_1$ (which may be subverted and even contain the same backdoor). We prove that when r_0, r_1 are independently sampled from RG_0 and RG_1, mixing them with an immunization function Φ can indeed destroy subliminal channels in the

[1] Note that an environment of independent modules that are isolated is encouraged in modern computer security teachings, and is supported by a variety of modern computer architectures (See [9, 12, 14, 16, 18, 23–25]). Furthermore, under the name of "modular programming" it is a very widely accepted specification and design paradigm that is simple to implement, and is known to accelerate software development and software testing.

	[4]	[6]	[3]	**Our SE**	**Our PKE**	[2]	[5, 11]
Symmetric key	√	√	√	**√**	**n/a**	n/a	√
Public key	n/a	n/a	n/a	**n/a**	**√**	√	√
Generic	×	×	×	**√**	**√**	×	×
IND-CPA security	√	√	√	**√**	**√**	×	√
Untrusted key gen	×	×	×	**√**	**√**	×	√
Untrusted randomness	n/a	n/a	n/a	**√**	**√**	n/a	×
Offline Watchdog	×	×	×	**√**	**√**	×	√
Dependent execution	×	×	×	**×**	**×**	×	×

Table 1: "Generic" refers to whether the technique can be used to immunize any encryption scheme. Only deterministic encryption was considered in [2–4, 6], thus we treat it as "do not apply" in the "untrusted randomness" item. We treat the decryptability assumption as requiring at least an online watchdog [20] to ensure it. Regarding "Dependent execution" item, we note that all existing work require some kind of isolated execution of components, and none achieves it.

implementation of a wide class of randomized algorithms in the random oracle model.[2] To necessitate "double" splitting, we also extend the attacks of [3, 4, 10] to present impossibility of CPA secure public key encryption with blackbox implementation of randomness generation (even its output is hashed by a untampered random oracle).

(3) We apply this general technique to immunize each algorithm of any symmetric-key (single-bit) encryption scheme, including key generation. Our construction preserves the IND-CPA security. We then immunize symmetric key encryption schemes for large message spaces. To defend against input-trigger-style attacks [6], we further assume the user's computing base contains a trusted addition function (e.g., in its computing base). We also consider correctness (previously, it was mostly *assumed* as a decryptability assumption) in the cliptographic setting. These techniques can be applied directly to immunize any public-key encryption, and, in turn, our generic construction gives the first IND-CPA secure scheme in the cliptographic setting without relying on trusted randomness.

Table 1 summarizes our results and the state of the art.

We note that designing cryptosystems immune to kleptographic subversion is a very active area which has led to remarkable new models and important techniques; many of these are realizable by systems and can reduce the threat of subversion attacks under various relaxation of the setting, or under setup assumptions (all, justified by the very strong adversary). These earlier works have motivated us, and further, we felt that semantic security of encryption – perhaps the most important security notion– deserves to be better understood theoretically and advanced practically.

[2]Same as [20], we only assume the specification Φ_{SPEC} to be an random oracle, while the implementation Φ_{IMPL} can be arbitrarily subverted.

2 MAIN RESULTS, ROADMAP AND RELATED WORK

Following the definitional framework of [20], we describe the definition of subversion resistant public key encryption, (i.e., IND-CPA in the cliptographic setting, sometimes we also call it cliptographically IND-CPA secure) first. As we explained the intuition in introduction, here we only recall that one can arrive at a variety of different definitions based on the order of quantification for the watchdog $\mathcal{W}$ and adversary $\mathcal{A}$, and whether $\mathcal{W}$ is given any further information (such as a transcript of the security game). We refer to [20] for further discussion. In this paper we will adopt the strongest of the definitions of [20] (which gives the watchdog the least power): in their terminology, we will consider a *universal* and *offline* watchdog. In such a definition, the watchdog only tests the implementation once with only oracle access. In particular, $\mathcal{W}$ has no access to the actual communications during the security game. Moreover, the description of the watchdog is quantified before the adversary. To make it concrete for IND-CPA secure public key (bit) encryption, we present the definition as follows:

Definition 2.1. For any public key bit encryption scheme, consider a specification $\mathbf{E}_{\text{SPEC}} := (\mathsf{KG}_{\text{SPEC}}, \mathsf{Enc}_{\text{SPEC}}, \mathsf{Dec}_{\text{SPEC}})$. We say specification $\mathbf{E}_{\text{SPEC}}$ is *subversion resistant in the offline watchdog model* if there exists an offline PPT watchdog $\mathcal{W}$, for any PPT adversary $\mathcal{A}$, such that, at least one of the following conditions hold in the game defined in Fig. 2:

$$\mathbf{Det}_{\mathcal{W},\mathcal{A}} \text{ is non-negligible, or, } \mathbf{Adv}_{\mathcal{A}} \text{ is negligible,}$$

where the detection probability of the watchdog is defined as:

$$\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda) = |\Pr[\mathcal{W}^{\mathbf{E}_{\text{IMPL}}}(1^\lambda) = 1] - \Pr[\mathcal{W}^{\mathbf{E}_{\text{SPEC}}}(1^\lambda) = 1]|,$$

and, $\mathcal{A}$'s advantage is defined as:

$$\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = |\Pr[b_C = 1] - \frac{1}{2}|.$$

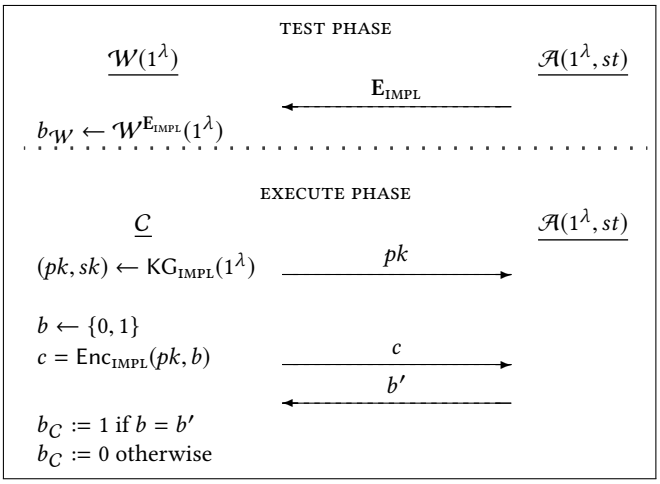


Figure 2: Subversion-resistant public key bit encryption, where $\mathbf{E}_{\text{IMPL}} := (\mathsf{KG}_{\text{IMPL}}, \mathsf{Enc}_{\text{IMPL}}, \mathsf{Dec}_{\text{IMPL}})$.

Note that the definition requires only non-negligible detection probability on the part of the watchdog, but it can be directly amplified by repetition if we are willing to relax the model and allow the watchdog depends on the running time of the adversary.[3]

Main Theorem(*informal*): *For any IND-CPA secure public key bit encryption, there exists a specification such that it is subversion resistant and correct in the split program model. Furthermore, for any IND-CPA public key encryption (supporting large input space), there exists a specification such that it is subversion resistant and correct assuming further a trusted $\oplus$ operation.*

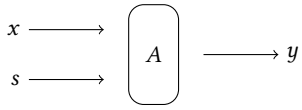
Roadmap towards constructing a cliptoraphically IND-CPA secure PKE. In the rest of the paper, we will first formalizing models for immunizing a subverted randomized algorithm. We then propose a general technique of of destroying subliminal channels in the output of a subverted randomized algorithm, via modular design and isolated execution. With those tools at hand, we demonstrate how to immunize every subverted algorithm in a bit encryption scheme one-by-one, we further consider how to achieve secure encryption schemes supporting large messages. Finally, as a corollary, we obtain subversion resistant public key encryption and finishes the proof for our main theorem above.

Related works. Kleptography introduced by Young and Yung [26, 27], primarily highlighted the possibility of subverting key generation, and left open the problem of defending against such subversion. Recent work of Russell, Tang, Yung and Zhou [20] has made initial progress on protecting key generation for specific cryptographic algorithms (trapdoor one-way permutations, pseudorandom generators, and digital signature scheme). However, these techniques do not remove steganographic channels needed for semantic security.

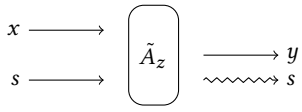
Also recently, several research threads have studied the kleptographic setting, developing both new attacks and defense mechanisms. In particular, Bellare, et al. [4] studied subverted randomized encryption algorithms, building a steganographic channel that leaks secrets bit by bit; t Indeed, subliminal channel attacks turn out to be the major obstacle in this area, and have been further explored by Ateniese et al. [1], Bellare et al. [2, 3], Degabriele et al. [6], and Dodis et al. [10]. A common feature of these works [2–4, 6] is to adopt *deterministic* algorithms and to assume honest key generation. Additionally, these works do not rely merely on testing: Most, in fact, require an a priori "decryptability" condition which demands that every message encrypted using the implementation should be decrypted correctly using the specification. A notable exception is [6]; however, they rely on a watchdog that possesses access to the actual challenger–adversary communication transcript (including the internal state of the challenger).

The authors of [5, 11, 17] considered defense mechanisms with a "reverse firewall" that is trusted to generate good randomness and faithfully "re-randomize" incoming and outgoing communication. This model is attractive as it may permit quite general feasibility results; on the other hand, it relies on an independant component which is a source of trusted randomness, and "re-randomizable" structure of the underlying algorithms (it generalized such a trusted component called "trusted warden" [8] used to eliminate subliminal channels in authentication protocols).

[3]Trivial amplification transforms a gap of ϵ to $1-\delta$ with $k = \epsilon^{-1}\log(\delta^{-1})$ repetitions. As the watchdog's running time is independent of the adversary, however, amplification cannot be adapted to a particular non-negligible function. However, if the watchdog is permitted a number of samples that depends on the adversary, then one can amplify non-negligible detection probability to $1 - o(1)$ for an infinite set of inputs.



(a) A randomized algorithm which leaks no information about s.



(b) A subverted algorithm depending on a hidden random string z. Its output y is indistinguishable from A, but with knowledge of z it leaks s.

Figure 3: Embedding a subliminal channel in a randomized algorithm A.

In contrast to all previous work, our goal is to develop semantic secure encryptions in a stricter model that does not require clean keys, trusted randomness, strong watchdogs, or decryptability assumptions (i.e., all cryptographic components can be subverted, only the very basic computing base is trusted, e.g., the hardware and the operating system's basic functions). We stress again, some trusted component was needed in earlier works, an independent execution assumption has to be *implicitly* made (separating the subverted and the trusted components). This holds for all previous work which actually employ independent separate modules (without explicitly claiming this). See the comparisons in Table 1.

3 FUNDAMENTAL BUILDING BLOCKS: ELIMINATING SUBLIMINAL CHANNELS IN SUBVERTED RANDOMIZED ALGORITHMS

In this section, we will introduce the technical building blocks towards proving our main theorem.

3.1 Definitions

As explained in the introduction, our focus will be the challenge of *generically destroying* subliminal channels which may have been adversarially embedded in a subverted algorithm. We briefly recall the notion of a subliminal channel to set the stage for the basic definitions below.

Consider an (honest) randomized algorithm A which takes an input x and has additional access to a "secret" bit $s \in \{0,1\}$. The algorithm produces a random output y, which we assume leaks no information about s. A fundamental result in steganography [13, 21, 22] asserts that is is possible to construct a subverted algorithm $\tilde{A}_z$, whose behavior is determined by a hidden random string z, s.t., (i.) for all inputs x and s, the distribution produced by $\tilde{A}_z(x,s)$ (including the random selection of z) is *identical* to the distribution produced by $A(x,s)$, and hence leaks no information about s; but, (ii.) with knowledge of z, the output of $\tilde{A}_z$ is highly correlated with s. In particular, an adversary who owns z can use the output of $\tilde{A}_z$ to infer s with high probability. See Figs. 3a and 3b.

In the cliptographic setting, our goal of *destroying* a subliminal channel means that the adversary cannot learn any extra information from the output of a subverted implementation, so long as it has passed the checking of the watchdog. We adopt the general definitional framework of [20], and generalize the notion defined by the surveillance game in [4] (which was defined only for symmetric key encryption) to formulate this intuition, and it defines a new notion that we call "stego-freeness". Specifically, we compare the information leaked by the implementation with that leaked by the specification (or, equivalently, an honest implementation). Stego-free specifications for algorithms will be the stepping stones to our cliptographically IND-CPA secure PKE. We note that our new notion of "stego-freeness" is defined mostly at the algorithm level, while the definition of subversion resistence from [20] was defined at the cryptographic primitive level.

We also remark here that throughout the paper, we consider only stateless algorithms. While it is easy to see that a subverted randomness generation can be simulated using a (deterministic) pseudorandom function if a global state can be synchronized between the implementation and the adversary, thus no entropy will be generated. We can still demonstrate feasibility in the setting of restricted stateful cases, and we will defer those discussions to the full version.

Defining stego-freeness. We now formally define stego-freeness for any (randomized) algorithm G under subversion. Following the basic cliptographic models, the adversary $\mathcal{A}$ prepares a (potentially subverted) implementation G_{IMPL} of the algorithm G; we let G_{SPEC} denote the specification of the algorithm. Stego-freeness means **either** the adversary $\mathcal{A}$ cannot learn any extra information from the outputs of G_{IMPL} (in comparison with that of G_{SPEC}), **or** the subversion can be detected by the watchdog $\mathcal{W}$ (using oracle access to G_{IMPL}). Depending on how communication is generated and whether the randomized algorithm can takes rich inputs, we have a variety of definitions; We emphasize here that different type of inputs in current setting correspond to different algorithms, and lead to dramatically different outcome.

We begin with the following elementary version for randomized algorithms—such as key generation—that rely *only* on a length parameter.

Definition 3.1 (stego-free, basic form). Consider a (randomized) algorithm G with specification G_{SPEC}. We say such G_{SPEC} is *stego-free in the offline watchdog model* if there exists a PPT watchdog $\mathcal{W}$ so that for any PPT adversary $\mathcal{A}$ playing the following game (see Fig. 4), it satisfies that

either $\mathbf{Adv}_{\mathcal{A}}$ is negligible, or $\mathbf{Det}_{\mathcal{W},\mathcal{A}}$ is non-negligible

where the detection probability is

$$\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda) = \left|\Pr[\mathcal{W}^{\mathsf{G}_{\text{IMPL}}}(1^\lambda) = 1] - \Pr[\Pr[\mathcal{W}^{\mathsf{G}_{\text{SPEC}}}(1^\lambda) = 1]\right| .$$

and adversary's advantage is defined as:

$$\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = |\Pr[b_C = 1] - 1/2| .$$

More general definitions of stego-freeness. In the above game, G only takes as input a fixed security parameter (often ignored later in the paper); Besides the security parameter, we can consider algorithms which take richer inputs. Such extensions will be important for applications. But they significantly complicate the task

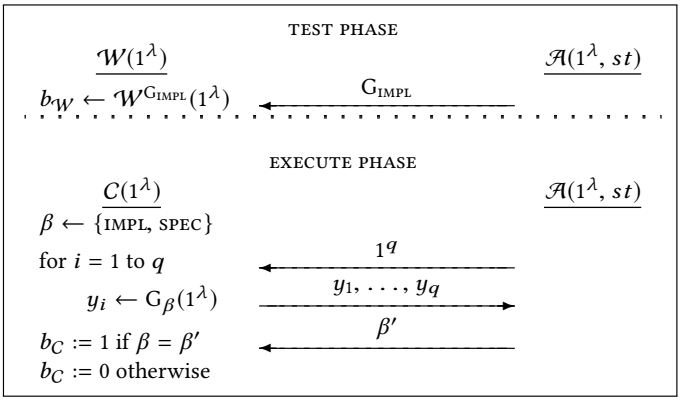


Figure 4: A game for stego-freeness.

of destroying an embedded steganographic channel. One note is that for input taken from a small domain, (for example, $\{0, 1\}$), we simply allow the adversary to query the evaluation on all inputs.

Beyond the previous cases, we may consider algorithms taking inputs from a large domain. The most straightforward adaptation permits the adversary to sample $\mathsf{G}_{\text{IMPL}}(x_i)$ at inputs x_i of her choice. However, this model suffers from a crippling "input trigger" attack [6] (where the adversary hides some secret at a "trigger" location x, which can be impossible for an offline watchdog to detect); we discuss this in detail later. However, there is a compromise setting that captures many cases of actual interest and permits strong feasibility results. In this setting we permit the adversary to determine inputs to a randomized algorithm G by specifying a randomized *input generator* IG: The input generator may be an arbitrary PPT algorithm with the condition that given 1^λ it produces (typically random) outputs of length λ. This implicitly defines the randomized algorithm $\mathsf{G}(1^\lambda, \mathsf{IG}(1^\lambda))$. (*See remark below for concrete use case.*) In our setting, the watchdog is provided (oracle access) to IG, which it may use during its testing of G_{IMPL}. Note that IG is chosen by the adversary during the security game; it is by default could be "subverted". Revisiting the security game in this new setting, challenges $\{y_i\}$ are generated by first sampling $m_i \leftarrow \mathsf{IG}(1^\lambda)$, and then obtaining $y_i \leftarrow \mathsf{G}_\beta(1^\lambda, m_i)$ by calling G_β using inputs 1^λ and m_i. Note that the adversary could use IG to produce some specific input "triggers" where G_{IMPL} deviates from G_{SPEC}.

Definition 3.2 (stego-free, general form). We say that a specification G_{SPEC} (of a randomized algorithm G) is stego-free in the offline watchdog model if it satisfies Def. 3.1 with the security game of Fig. 5. Note that the PPT input generator IG may be determined by $\mathcal{A}$ during the game.

Remark: use cases of the above definitions. Definition 3.1 can capture algorithms like randomness generation and key generation when we instantiate G to be the corresponding functionality. While the more general notion (Def. 3.2) of stego-freeness (allowing adversary to supply an input generator IG) captures algorithms that take the output of other implementation as input. For instance, $\mathsf{Enc}_{\text{IMPL}}$ will take the output of $\mathsf{KG}_{\text{IMPL}}$ as an input. Which of the definitions

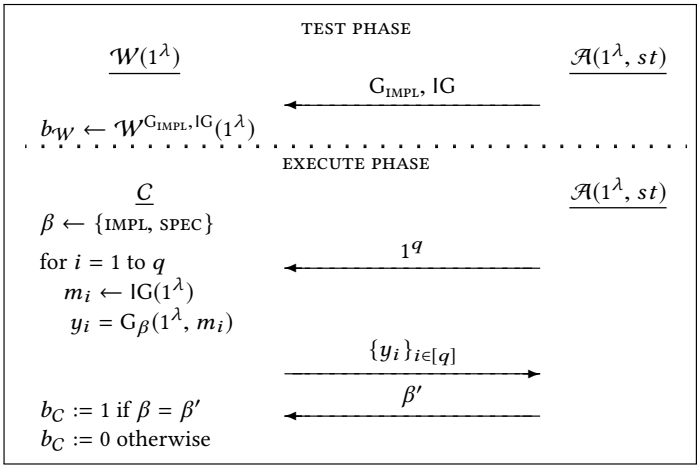


Figure 5: The stego-freeness game with input distribution $\{1^\lambda\} \times IG$.

(3.1 or 3.2) is appropriate for a given randomized algorithm can be determined from context, depending on whether IG is specified.

As mentioned above, an even stronger definition is obtained by permitting the adversary to simply choose the input m_i for each y_i directly. This notion reflects stego-freeness for algorithms with adversarially chosen inputs. Such a subverted implementation may have a hidden "trigger" that was randomly drawn during the (adversarial) manufacturing process and can permit the adversary to easily win the stego-freeness distinguishing game. In fact, such a trigger attack does not even require that G be randomized: for example, consider the algorithm $G_{SPEC}(1^\lambda, x) := x$, defined for $x \in \{0,1\}^\lambda$. The adversary then uniformly draws $z \leftarrow \{0,1\}^\lambda$ and defines $G_{IMPL}(1^\lambda, x) = 0^\lambda$ if $x = z$. As the placement of the trigger (z) is random, the watchdog cannot detect disagreement between G_{IMPL} and G_{SPEC}, while the adversary can distinguish these algorithms easily by querying z. In a practical setting, an algorithm with such an input trigger can leak arbitrary private data to an adversary in a way undetectable to an offline watchdog. This was formally demonstrated in [6]. Nevertheless, we will discuss in Section 4.1 a method for sidestepping this impossibility with only an offline watchdog by assuming some small piece of extra trusted operation, such as "one trusted addition."[4]

Discussions about stego-freeness and steganography. We emphasize two properties of these definitions. First, if a proposed specification satisfies such definitions, direct use of the implementation preserves the typical security guarantees originally possessed by the specification. This enables us to provide fairly modular security proofs by designing Stegofree specifications for each algorithm.

The second, and more critical, issue pertains to the feasibility of achieving these notions of stego-freeness: at first glance they appear hopeless. It is known that general steganography is always possible over a channel with sufficient entropy [13, 21, 22]. This implies that the subverted algorithm G_{IMPL} can always produce a sequence of messages that enable the adversary to retrieve secrets from the (public) outputs $y_1, \ldots, y_q$. As a simple example of such subversion in our setting, consider the algorithm $G_{SPEC}(1^\lambda)$ which outputs a uniformly random element of $\{0,1\}^\lambda$. Consider then the subverted implementation $G^z_{IMPL}(1^\lambda)$ whose behavior is determined by a uniformly random string $z \leftarrow \{0,1\}^\lambda$ chosen by the adversary: the algorithm $G^z_{IMPL}(1^\lambda)$ outputs a uniformly random element from

$$H = \{w \in \{0,1\}^\lambda \mid \mathsf{lsb}(F_z(w)) = 0\},$$

where $\mathsf{lsb}(x)$ denotes the least-significant bit of x and $F_z(\cdot)$ denotes a pseudorandom function (PRF) with key z. (Note that elements of H can be drawn by rejection sampling.) Of course, it is easy for the adversary to distinguish G_{IMPL} from G_{SPEC} (as G_{IMPL} only outputs strings with a particular property that is easily testable by the adversary who has z). On the other hand, no watchdog can distinguish these algorithms without breaking the PRF.

The above suggests that if the user makes only *black-box* use of the subverted implementation of randomized algorithms, it is hopeless to achieve stego-freeness. This again motivates the *non-black-box* style of the split-program model.

The split-program methodology. To overcome the steganographic attacks, we allow the specification of an algorithm to be split into several components. Fortunately, the definitional framework of [20] is quite general and flexible, that enables us to describe a more fine-grained module design principle. In this *split-program model*, the adversary has to provide implementation for each of the component, otherwise she violates the specification in a trivial way. The watchdog can thus check them individually. And the components will be executed independently and then composed correctly by the user or his computing base. See also Appendix A. [5] To not frustrate reader too much, we choose not to present a model for each different splitting, instead, one can follow the above general principle to infer how the model should be tuned implicitly.

One example of this framework is the simple method proposed in [20] to study certain randomized algorithms: they begin by specifying a (general) randomized algorithm G as a pair (RG, dG) where RG is the *randomness generation* algorithm, responsible for generating a uniform random string of appropriate length, and dG is a deterministic algorithm that takes the random coins produced by RG to produce the final output. They then add to this specification a third deterministic algorithm Φ which acts as a kind of "immunization function" for the random bits generated by RG. Specifically, given the implementations (RG_{IMPL}, dG_{IMPL}, Φ_{IMPL}), the challenger *amalgamates* them by first querying $r_0 \leftarrow RG_{IMPL}$, "sanitizing" this randomness by passing it into Φ_{IMPL} to receive $r \leftarrow \Phi_{IMPL}(r_0)$ and, finally, running $y \leftarrow dG_{IMPL}(r)$. They show that in several contexts such an "immunization" can preserve security even under subversion. We remark that a simple decomposition of this form cannot destroy steganography in general, and we show an explicit attack; see Sec. 3.5.

[4] All previous works either simply assume it won't happen (the decryptability assumption) or employ at least an online watchdog who has access to the transcript between the challenger C and the adversary $\mathcal{A}$, (sometimes the secret key of C). And trusted addition is still weaker than the re-randomizer needed in the reverse firewalls.

[5] To reflect the re-assembly is done properly, the **challenger** will execute the components independently and amalgamate the outputs to yield the fully functional implementation in the security game. In general, if we need to power the user with some basic trusted operation–which may come from user's computing base–trusted operation can be reflected in the game that the **challenger** carries out such operation. We refer to Sec. 4.1 for one such example of assuming a trusted $\oplus$.

In this paper, we will show that this general methodology has remarkable power against subversion: with further decomposition, we show it is possible to generically destroy steganographic channels. This provides us a family of tools for developing cliptographyically-secure cryptography without abandoning randomized algorithms.

Remark: random oracles. In some settings, we establish results in the conventional random oracle model which requires some special treatment in the model above. In a kleptographic setting with complete subversion, we must explicitly permit the adversary to tamper with the "implementation" of the random oracle supplied to the challenger. In such settings, we provide the watchdog – as usual – oracle access to both the "specification" of the random oracle (simply a random function) and the adversary's "implementation", which may deviate from the random oracle itself. For concreteness, we permit the adversary to "tamper" with a random oracle h by providing an efficient algorithm $T^h(x)$ (with oracle access to h) which computes the "implementation" $\tilde{h}$ – thus the implementation $\tilde{h}(x)$ is given by $T^h(x)$ for all x. Likewise, during the security game, the challenger is provided oracle access only to the subverted implementation $\tilde{h}$ of the random oracle. As usual, the probabilities defining the security (and detection) games are taken over the choice of the random oracle. In this sense, the random oracle assumption used in our complete subversion model is weaker than the classical one, since we can allow even *imperfect* random oracles. Fortunately, when the random oracle is applied to a known input distribution, an offline watchdog can ensure that the implementation is almost consistent with its specification (see Lemma 3.3).

3.2 Purifying subverted randomness via double splitting

Next, we present our main result: a generic transformation that destroys subliminal channel which relies on "double-splitting" the randomness generation coupled with a public immunizing function. We also present simple results that "single" splitting already work if randomness is trusted, however, if the randomness generator is implemented by the adversary, we present an explicit attack, thus our double splitting is necessary. See sections. 3.4, 3.5.

Basic guarantees provided by an offline watchdog $\mathcal{W}$: First, $\mathcal{W}$ can at least guarantee that the output r_0 of an implementation of randomness generation $\mathsf{RG}_{\mathrm{IMPL}}$ is *unpredictable* to the adversary $\mathcal{A}$. Otherwise, the distribution given by $\mathsf{RG}_{\mathrm{IMPL}}$ would have significant (non-negligible) collision probability,[6] which can be easily tested by $\mathcal{W}$ who simply draws two samples and rejects if it observes a collision. (As with the other tests we discuss, the success of this test can be amplified by repetition.) On the other hand, the collision probability of $\mathsf{RG}_{\mathrm{SPEC}}$ is negligible.

Second, $\mathcal{W}$ can ensure that the implementation of a *deterministic* algorithm disagrees with its specification ($G_{\mathrm{SPEC}}(x) \neq G_{\mathrm{IMPL}}(x)$) with negligible probability when inputs are drawn from a known input distribution IG. To see this, $\mathcal{W}$ can simply draw samples from IG and compare the evaluations using $\mathsf{G}_{\mathrm{IMPL}}$ and $\mathsf{G}_{\mathrm{SPEC}}$.This also follows directly from Lemma 3.3 from [20], (see below). That said, deterministic algorithms with a public input distribution satisfy stego-freeness in a straightforward fashion. Throughout, we use the term "public" distribution to refer to any efficiently sampleable source that the watchdog can construct, either using IG supplied by the adversary or some honest distribution.

Lemma 3.3 ([20]). *Consider $\Pi_{IMPL} := (F^1_{IMPL}, \ldots, F^k_{IMPL})$, an adversarial implementation of a specification $\Pi_{SPEC} = (F^1_{SPEC}, \ldots, F^k_{SPEC})$, where $F^1, \ldots, F^k$ are deterministic algorithms. Additionally, for each security parameter λ, public input distributions $X^1_\lambda, \ldots, X^k_\lambda$ are defined respectively. If $\exists j \in [k], \Pr[F^j_{IMPL}(x) \neq F^j_{SPEC}(x) : x \leftarrow X^j_\lambda]$ is δ, this can be detected by a PPT offline watchdog with probability at least δ.*

Intuitions. From a first look, a specification of an algorithm G with form of $(\mathsf{RG}_{\mathrm{SPEC}}, \mathsf{dG}_{\mathrm{SPEC}}, \Phi_{\mathrm{SPEC}})$, where $\mathsf{RG}_{\mathrm{SPEC}}(1^\lambda) \to r_0; \Phi_{\mathrm{SPEC}}(r_0) \to r$, and $\mathsf{dG}_{\mathrm{SPEC}}(r)$ generates the final output, may already provide security in a kleptographic setting if the immunization function Φ can suitably interfere with generation of biased output by the implementation of RG. To simplify our presentation, we assume throughout that $\mathsf{RG}_{\mathrm{SPEC}}$ produces at least λ bits of randomness; this does not affect the generality of the results. (Our techniques can be adapted to a low-entropy setting with some changes to running time of the watchdog.[7]) This suggests the intuition that $\Phi_{\mathrm{SPEC}}(r_0)$ may appear to $\mathcal{A}$ to be a randomly drawn value if Φ_{SPEC} is a random oracle. Unfortunately, $\mathcal{A}$ also holds the backdoor z which may contain information about the final output $r = \Phi_{\mathrm{IMPL}}(r_0)$ generated by the sampling and "cleaning" process. In particular, we will show an attack in Sec. 3.5, the subverted implementation has full access to Φ_{SPEC} and may thus bias the output $r = \Phi_{\mathrm{SPEC}}(r_0)$ as a function of z, which can be noticed by $\mathcal{A}$.

To circumvent the above obstacle, we introduce a new technique that further splits RG into *two* random algorithms, $\mathsf{RG}^0_{\mathrm{SPEC}}$ and $\mathsf{RG}^1_{\mathrm{SPEC}}$, and combines their outputs using an immunization function Φ_{SPEC}; we shall see that this mechanism can destroy subliminal channels. In general, in the split program model, the user simply runs $\mathsf{RG}_{\mathrm{IMPL}}$ twice independently to simulate two separate $\mathsf{RG}^0_{\mathrm{IMPL}}$ and $\mathsf{RG}^1_{\mathrm{IMPL}}$ and passes the joint outputs to Φ_{IMPL}; the final output will have the form $r = \Phi_{\mathrm{IMPL}}(r_0 \circ r_1)$ (where $\circ$ denotes concatenation). The main idea behind this strategy is that it frustrates attempts by $\mathsf{RG}^0_{\mathrm{IMPL}}$ and $\mathsf{RG}^1_{\mathrm{IMPL}}$ to launch sophisticated rejection-sampling because the final output is not fully determined by either output. (Neither can evaluate $\Phi_{\mathrm{IMPL}}(r_0 \circ r_1)$ during the generation of r_0 or r_1.) In this way, if Φ_{SPEC} is modeled as a random oracle, the final output $\Phi_{\mathrm{SPEC}}(r_0 \circ r_1)$ will be uncorrelated with $\mathcal{A}$'s state (which includes both $\mathcal{A}$'s random oracle queries and z). Now we can safely claim that r looks uniform even to $\mathcal{A}$.[8]

With this approach, we demonstrate a qualitative advantage of the split-program methodology. We first describe a stego-free specification of randomness generation in Fig. 6, and then proceed to

[6] Observe that if D is a probability distribution on a set X, the optimal strategy for predicting the result of drawing an element of X according to D is simply to guess $\max_x D(x)$. If this maximum probability is ϵ, then the collision probability of D, equal to $\sum_x D(x)^2$, is at least ϵ^2.

[7] Observe that if $\mathsf{RG}_{\mathrm{SPEC}}$ produces only $O(\log n)$ random coins then an offline watchdog, by a suitable regimen of repeated sampling, can empirically approximate (with high probability) the distribution of $\mathsf{RG}_{\mathrm{IMPL}}$ with high accuracy. This can be directly compared with $\mathsf{RG}_{\mathrm{SPEC}}$ using distance in total variation. Note that such a watchdog requires a number of samples polynomial in the resulting error.

[8] We stress again that Φ_{IMPL} can be subverted, but it is a deterministic function with a known input distribution, the inconsistency can be ensured to be at only a negligible fraction of places due to the watchdog, see Lemma 3.3.

give an immunization strategy for arbitrary randomized algorithms so long as they have a public input distribution (or a small input domain). We apply these tools in next section to compile encryption schemes that retains semantic security.

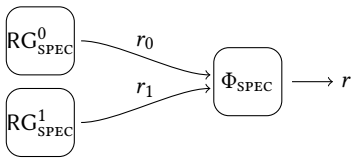


Figure 6: A stego-free spec for randomness generation.

Theorem 3.4. *Consider randomness generation* RG *with specification* $(\mathrm{RG}^0_{SPEC}, \mathrm{RG}^1_{SPEC}, \Phi_{SPEC})$ *as described in Fig. 6:*

- RG^0_{SPEC} *and* RG^1_{SPEC}*, given* 1^λ*, output uniformly random strings of length* λ*;*
- Φ_{SPEC} *is a hash function so that* $\Phi_{SPEC}(w)$ *has length* $\lceil |w|/2 \rceil$*;*
- *the specification for* $\mathrm{RG}(1^\lambda)$ *is the trusted composition:* $\Phi_{SPEC}(\mathrm{RG}^0_{SPEC}(1^\lambda), \mathrm{RG}^1_{SPEC}(1^\lambda))$.

Then RG_{SPEC} *is stego-free in the split program model if* Φ_{SPEC} *is modeled as a random oracle.*

Proof. If the specification of Fig. 6 is not stego-free, then for any offline watchdog $\mathcal{W}$ there is an adversary $\mathcal{A}$ that can prepare an implementation $\mathrm{RG}_{\mathrm{IMPL}} := (\mathrm{RG}^0_{\mathrm{IMPL}}, \mathrm{RG}^1_{\mathrm{IMPL}}, \Phi_{\mathrm{IMPL}})$ satisfying the following: (1.) $\mathcal{W}$ cannot distinguish $\mathrm{RG}_{\mathrm{IMPL}}$ from $\mathrm{RG}_{\mathrm{SPEC}}$ via oracle access; (2.) The adversary $\mathcal{A}$ can distinguish output of $\mathrm{RG}_{\mathrm{IMPL}}$ from $\mathrm{RG}_{\mathrm{SPEC}}$. We will then define an offline watchdog such that these two conditions cannot hold simultaneously for any adversary.

An offline watchdog. The watchdog $\mathcal{W}$'s strategy is as follows: $\mathcal{W}$ first draws a constant number of samples from $\mathrm{RG}^0_{\mathrm{IMPL}}$ and $\mathrm{RG}^1_{\mathrm{IMPL}}$; if $\mathcal{W}$ observes a collision in either distribution, it rejects the implementation outright (as collisions are negligible in SPEC). Next, $\mathcal{W}$ draws pairs of samples (again) from $\mathrm{RG}^0_{\mathrm{IMPL}}$ and $\mathrm{RG}^1_{\mathrm{IMPL}}$ and evaluates Φ_{IMPL} on (the concatenation of) each pair to ensure that the result is consistent with Φ_{SPEC}.

Next, we will show, for any PPT $\mathcal{A}$, if the detection probability $\mathbf{Det}_{\mathcal{W},\mathcal{A}}$ is negligible (this is the sum of detection probability for collision denoted by δ_c, and the detection probability for inconsistency δ_i), then the advantage $\mathbf{Adv}_{\mathcal{A}}$ will also be negligible, thus the two conditions cannot hold simultaneously.

Game transitions. We will go through the security game part of Def. 3.1 step by step. Without loss of generality, we assume the challenge r contains only one element.

In Game-0, $\mathcal{A}$ prepares subverted implementations $\mathrm{RG}_{\mathrm{IMPL}} := (\mathrm{RG}^0_{\mathrm{IMPL}}, \mathrm{RG}^1_{\mathrm{IMPL}}, \Phi_{\mathrm{IMPL}})$; we let Q be the set of random oracle queries $\mathcal{A}$ made during preparation of $\mathrm{RG}_{\mathrm{IMPL}}$.

The challenger C samples from $\mathrm{RG}^0_{\mathrm{IMPL}}$ and $\mathrm{RG}^1_{\mathrm{IMPL}}$ respectively and receives r_0 and r_1; then C evaluates Φ_{IMPL} at $r_0 \circ r_1$ and sends the output r as the challenge to $\mathcal{A}$. Let Q_b (for $b = 0, 1$) be the set of random oracle queries made by $\mathrm{RG}^b_{\mathrm{IMPL}}$ before outputting r_b. All random oracle queries will be (consistently) answered with randomly chosen values.

Game-1 is identical to Game-0, except that Φ_{IMPL} is replaced with Φ_{SPEC}; Game-2 is identical to Game-1, except that C simply chooses a uniform r and directly sends it to $\mathcal{A}$ as the challenge; Game-3 is identical to Game-2, except that $\mathrm{RG}_{\mathrm{IMPL}}$ is completely replaced with $\mathrm{RG}_{\mathrm{SPEC}}$; Game-4 is identical to Game-3, except that r is generated as in Game-0, but the challenger uses $\mathrm{RG}_{\mathrm{SPEC}}$ instead.

Probabilistic analysis. We will analyze the gaps of each game transition parametrized by the quantity $\mathbf{Det}_{\mathcal{W},\mathcal{A}}$.

First, since Φ_{SPEC} is deterministic and with a public input distribution (which is $\mathrm{RG}^0_{\mathrm{IMPL}} \times \mathrm{RG}^1_{\mathrm{IMPL}}$), following Lemma 3.3,

$$\Pr\left[\begin{array}{l}\Phi_{\mathrm{IMPL}}(r_0 \circ r_1) \neq \Phi_{\mathrm{SPEC}}(r_0 \circ r_1) : \\ r_0 \leftarrow \mathrm{RG}^0_{\mathrm{IMPL}}, r_1 \leftarrow \mathrm{RG}^1_{\mathrm{IMPL}}\end{array}\right] \leq \delta_i.$$

Otherwise, the watchdog $\mathcal{W}$ will notice the inconsistency with probability at least δ_i (by choosing 1 random sample from $\mathrm{RG}^0_{\mathrm{IMPL}} \times \mathrm{RG}^1_{\mathrm{IMPL}}$, evaluate on both $\Phi_{\mathrm{SPEC}}, \Phi_{\mathrm{IMPL}}$ and compare). It follows that replacing Φ_{IMPL} with Φ_{SPEC} would incur a $\mathbf{Det}_{\mathcal{W},\mathcal{A}}$ difference, thus:

$$|\Pr[b_C = 1 \text{ in Game-0}] - \Pr[b_C = 1 \text{ in Game-1}]| \leq \delta_i.$$

Second, we will argue that the probability that $r_0 \circ r_1$ is ever queried (falling in $Q \cup Q_0 \cup Q_1$) is negligible, and now we are in Game-1 using Φ_{SPEC} which is a random oracle.

It is easy to see that $\Pr[r_0 \circ r_1 \in Q] \leq \sqrt{\delta_c} \cdot \mathrm{poly}(\lambda)$; otherwise, the watchdog will observe a collision in $\mathrm{RG}^i_{\mathrm{IMPL}}$ with probability δ_c. To see this, let $R_0 = \{r_0 \mid \exists r_1, r_0 \circ r_1 \in Q\}$, note that Q, R_0 are only polynomially large. If the probability that $r_0 \circ r_1$ falls into Q (thus also the probability that r_0 falls into R_0) is δ, that means the value r_0 will be generated by $\mathrm{RG}^0_{\mathrm{IMPL}}$ with probability at least $\delta_0 = \delta/|R_0|$. It follows that the collision probability that $\mathrm{RG}^0_{\mathrm{IMPL}}$ produces the same output r_0 would be δ_0^2. While on the other hand, $\mathrm{RG}^0_{\mathrm{SPEC}}$ produces uniform bits, the collision probability (that $\mathrm{RG}^0_{\mathrm{IMPL}}$ produces the same uniform output string r_0) would be negligible. Thus the watchdog can easily distinguish $\mathrm{RG}^0_{\mathrm{IMPL}}$ from $\mathrm{RG}^0_{\mathrm{SPEC}}$ when drawing, say 2 samples. It follows that $\delta_0^2 \leq \delta_c$, and we can obtain $\delta \leq \sqrt{\delta_c}|R_0|$.

Similarly, we bound the probability for Q_0, Q_1. Let $R_{0,1} = \{r_1 \mid \exists r_0, r_0 \circ r_1 \in Q_0\}$. Since $\mathrm{RG}^0_{\mathrm{IMPL}}, \mathrm{RG}^1_{\mathrm{IMPL}}$ are independently run, the probability that r_1 falls into the polynomially large set $R_{0,1}$ would be $\sqrt{\delta_c} \cdot |R_{0,1}|$; otherwise, if $\mathrm{RG}^1_{\mathrm{IMPL}}$ outputs r_1 with probability more than δ_c, then $\mathcal{W}$ can notice the difference between $\mathrm{RG}^1_{\mathrm{IMPL}}$ and $\mathrm{RG}^1_{\mathrm{SPEC}}$ by identifying collisions. Thus $\Pr[r_0 \circ r_1 \in Q_0] \leq \Pr[r_1 \in R_{0,1}] \leq \sqrt{\delta_c} \cdot \mathrm{poly}(\lambda)$. The same holds for Q_1.

The adversary $\mathcal{A}$ is holding the set of random oracle queries Q, and a backdoor z. The only way r may correlate with z is that $r_0 \circ r_1$ is queried during the execution of $\mathrm{RG}^0_{\mathrm{IMPL}}$, or $\mathrm{RG}^1_{\mathrm{IMPL}}$. If $r_0 \circ r_1 \notin Q \cup Q_0 \cup Q_1$, $r_0 \circ r_1$ will be independent with $\mathcal{A}$'s view (Q, z), thus $r = \Phi_{\mathrm{SPEC}}(r_0 \circ r_1) = \mathrm{RO}(r_0 \circ r_1)$ looks uniform to $\mathcal{A}$. We can claim that:

$$|\Pr[b_C = 1 \text{ in Game-1}] - \Pr[b_C = 1 \text{ in Game-2}]| \leq \sqrt{\delta_c} \cdot \mathrm{poly}(\lambda).$$

Next, it is easy to see that $\Pr[b_C = 1 \text{ in Game-2}] = \Pr[b_C = 1 \text{ in Game-3}]$ since the adversary receives the identical challenge. Also, $\Pr[b_C = 1 \text{ in Game-3}] = \Pr[b_C = 1 \text{ in Game-4}]$ since querying $\mathrm{RG}_{\mathrm{SPEC}}$ yields a uniform output $\mathrm{RO}(u_0 \circ u_1)$, where u_0, u_1 are uniformly chosen.

To conclude, we have

$$|\Pr[b_C = 1 \text{ in Game-0}] - \Pr[b_C = 1 \text{ in Game-4}]| \leq \delta_i + O(\sqrt{\delta_c}) \cdot \mathrm{poly}(\lambda).$$

Observe that Game-0 corresponds to the case that challenger flips a coin to be 0, i.e., C uses $\mathrm{RG}_{\mathrm{IMPL}}$ to generate the challenge messages, while Game-4 corresponds to the case that $b = 1$, when C uses $\mathrm{RG}_{\mathrm{SPEC}}$. Since we assume $\mathbf{Det}_{\mathcal{W},\mathcal{A}}$ is negligible, It follows that:

$$\mathbf{Adv}_{\mathcal{A}} = |\Pr[b_C = 1] - 1/2| \leq \mathrm{negl}(\lambda)\,.$$

Combine all, we conclude the theorem. □

Implementation considerations. Practical deployment of such splitting—especially as it requires independence—clearly requires detailed consideration of the particular computational environment. There exist a lot of research in this domain: e.g., [12, 16, 18, 24]. Here we list some lightweight examples. Modern lightweight virtualization layers such as Docker [9], insulate individual copies of the adversary's code; we remark that this also permits state duplication (which may be necessary for the watchdog in the stateful case). Even more lightweight solutions such as the *AppArmor* [14] or *TOMOYO* [23] infrastructure also have been deployed. These provide on-the-fly trapping of system calls, and have been successfully applied to isolate code for security purposes. In particular, these solutions can be applied to externally compiled executables. More aggressive complete virtualization is also possible. An alternate approach relies on constraining the source code (or the compiler) to directly limit I/O and system calls; this has the advantage that the components can be run efficiently in the native environment. Finally, there may also be settings where it is possible to isolate the program in the architectural/hardware layer or physically separate various components (e.g., using Intel SGX [25], or even move one RG^i outside the user's computer, use a random beacon, etc.).

Transition to the standard model. We also consider how to achieve stego-freeness without a random oracle. The main observation is that the watchdog can guarantee that each copy of $\mathrm{RG}^i_{\mathrm{IMPL}}$ provides at least $\log n$ bits of (min-)entropy. If we are willing to have more components (or simply executes one piece of implementation independently for more times) for randomness generation, we can accumulate entropy using a simple immunizing function and stretch the result using a PRG. We defer the detailed discussion about standard model constructions in the full version.

3.3 Stego-free specifications for randomized algorithms; a general transformation

Now we are ready to establish the general result yielding a stego-free specification for any randomized algorithm that is with a public input distribution. Furthermore, it can be generalized directly to the setting with an extra small input. As discussed in Sec. 3.1, those randomized algorithms already cover many of the interesting cases such as key generation and bit encryption.

The transformation. Consider a randomized algorithm G which uses $\lambda = \lambda(n)$ random bits for inputs of length n. Let $(\mathrm{dG}, \mathrm{RG})$ denote the natural specification of G that isolates randomness generation, so that $\mathrm{RG}(1^\lambda)$ produces λ uniformly random bits and $\mathrm{dG}(r, x)$ is a deterministic algorithm so that for every x, $\mathrm{G}(x)$ is equal to $\mathrm{dG}(\mathrm{RG}(1^\lambda(n), x)$ for $n = |x|$. ("Equal" here simply means these have identical output distributions.) As described above, we consider the transformed specification for G of the form $(\mathrm{RG}_1, \mathrm{RG}_2, \Phi, \mathrm{dG})$ where dG is as above, both $\mathrm{RG}_1(1^\lambda)$ and $\mathrm{RG}_2(1^\lambda)$ produce λ uniformly random bits, and Φ is a hash function that carries strings of length $2k$ to strings of length k. (See Fig. 7 below, which shows the transformation applied to an algorithm with a public input distribution generated by IG.) We will prove that when Φ_{SPEC} is given by a random oracle, this is a stego-free specification of G.

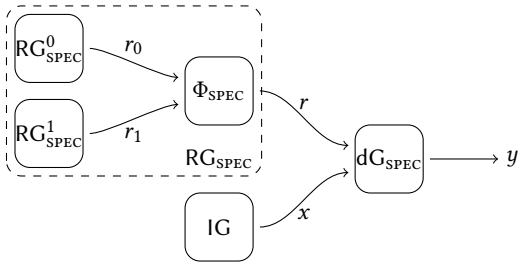


Figure 7: A stego-free specification for randomized algorithm G, where $x \leftarrow \mathsf{IG}$.

Security analysis. Following Lemma 3.3, we know that a subverted implementation $\mathrm{dG}_{\mathrm{IMPL}}$ of a deterministic algorithm will be consistent with its specification with an overwhelming probability when the inputs are sampled from some public distribution, which is $\mathsf{IG} \times \mathrm{RG}_{\mathrm{IMPL}}$. Thus $\mathrm{dG}_{\mathrm{IMPL}}$ can be considered as good as $\mathrm{dG}_{\mathrm{SPEC}}$ when restricted to this public input distribution. Furthermore, $\mathrm{RG}_{\mathrm{SPEC}}$ is stego-free, as discussed above. Thus all implementations can be replaced with their specifications, and the security follows easily. We record this in the theorem below, and we defer the detailed proof to Appendix D.

Theorem 3.5. *For any randomized algorithm* G*, consider the specification* $\mathrm{G}_{SPEC} := (\mathrm{RG}_{SPEC}, \mathrm{dG}_{SPEC})$*, where* RG_{SPEC} *generates* $\lambda = \lambda(n)$ *bits of uniform randomness and* dG_{SPEC} *is deterministic. We define* RG_{SPEC} *to be* $(\mathrm{RG}^0_{SPEC}, \mathrm{RG}^1_{SPEC}, \Phi_{SPEC})$ *as above. If (1.)* $\mathrm{RG}^0_{SPEC}(1^\lambda)$ *and* $\mathrm{RG}^1_{SPEC}(1^\lambda)$ *output* λ *uniform bits; (2.)* Φ_{SPEC} *takes* $r_0 \circ r_1$ *as input, and outputs* r *(so it maps strings of length* 2λ *to strings of length* λ*) (see Fig. 7), then* G_{SPEC} *is stego-free with a trusted combining. Here* Φ_{SPEC} *is modeled as a random oracle, and* $\mathrm{RG}^0_{IMPL}, \mathrm{RG}^1_{IMPL}$ *are executed independently.*

It is straightforward to generalize the result to support algorithms with an extra small size (polynomially large q) input as they essentially expand the input distribution to be q distributions, which can all be checked by the watchdog.

Corollary 3.6. *For any randomized algorithm* G*, consider a specification* $\mathrm{G}_{SPEC} := (\mathrm{RG}_{SPEC}, \mathrm{dG}_{SPEC})$*, where* RG_{SPEC} *is defined as in Fig. 6: If* dG_{SPEC} *takes* r*,* x*,* m *as input, where* x *is generated by a sampler* IG*, and* m *is taken from a polynomial size public domain* D*. Then the specification* G_{SPEC} *is stego-free with a trusted combing, if* Φ_{SPEC} *is modeled as a random oracle.*

3.4 Eliminating subliminal channels with trusted randomness

For completeness, we also briefly explain how we can immunize subverted randomized algorithms *generically* if we assume that the user has access to trusted randomness. This is as assumed in the reverse firewall model [11, 17]. However, using our split-program strategy, we do not need to rely on specific structure

(e.g., homomorphic property) of the underlying algorithm or some further trusted operations for re-randomization.

It proceeds simply as follows: every randomized algorithm G can be decomposed into RG, dG, where RG generates random coins r, and dG is a takes r and some samples from a public distribution IG as inputs. The specification will require the implementation to be in this form explicitly, thus the implementation provided by the adversary will be $(\mathsf{RG}_{\mathrm{IMPL}}, \mathsf{dG}_{\mathrm{IMPL}})$. Now that the user has trusted randomness, he can simply ignore $\mathsf{RG}_{\mathrm{IMPL}}$, and use his own trusted randomness R instead. The problem of immunizing $\mathsf{G}_{\mathrm{IMPL}}$ simply becomes achieving stego-freeness of $\mathsf{dG}_{\mathrm{IMPL}}$. It follows from Lemma 3.3 directly that when a deterministic algorithm is with a public input distribution, it is almost always consistent with its specification. As $\mathsf{dG}_{\mathrm{SPEC}}$ will take inputs from a known distribution $\mathsf{IG} \times \mathsf{R}$, thus will be stego-free when restricted to $\mathsf{IG} \times \mathsf{R}$.

3.5 Impossibility of publicly immunizing subverted random source

One may wonder whether the "double-splitting" strategy is necessary. Unfortunately, the steganographic attacks of [3, 4] can still be carried out in the "single" split-program model above, if the user is only given one subverted randomness generator $\mathsf{RG}_{\mathrm{IMPL}}$ (instead of trusted randomness) together with the corresponding deterministic part $\mathsf{dG}_{\mathrm{IMPL}}$.

Here we point out that such an attack exists even if we hash the output of $\mathsf{RG}_{\mathrm{IMPL}}$, as described in [20]. In fact, this approach can fail even in the most generous setting when the hash Φ is given by a untampered random oracle and the adversary only subverts RG.

The attack is a straightforward adaptation of the techniques from [3, 4]: the subverted implementation $\mathsf{RG}_{\mathrm{IMPL}}$ can evaluate $\mathsf{dG}_{\mathrm{SPEC}}$ and appropriately query the random oracle Φ_{SPEC} during the procedure of rejection sampling. It is easy, then, to arrange for the the output of $\mathsf{RG}_{\mathrm{IMPL}}$ to be biased in a way only detectable by the adversary. While a generic attack is possible, for concreteness we present an attack on a subverted public-key cryptosystem which permits the adversary to effortlessly determine the (plaintext) message bit. This indicates that more sophisticated non-black-box techniques such as ours are necessary to remove steganographic channels in general. A detailed description appears in Sec. C.

4 SUBVERSION-RESISTANT ENCRYPTION

Now we are ready to immunize encryption schemes.

4.1 Subversion-resistant symmetric encryption

We first construct subversion-resistant symmetric-key bit encryption in the case when all algorithms are subject to subversion. We then discuss correctness in the setting of large size messages and how to remove the "decryptability" assumption in previous works, it requires that every ciphertext generated by the subverted implementation can be correctly decrypted using the specification. While this helps to achieve security, it seems difficult to justify; see the criticism in [6]. In general, then, correctness is not placed on the same footing as security (which is established via the specification in tandem with watchdog testing), but is rather provided by fiat.

Defining subversion resistance and correctness. Similar as Def 2.1, we follow the definitional framework of [20] to define a subversion-resistant symmetric-key encryption scheme. As mentioned in Sec. 3.1, we focus on stateless encryption algorithms and defer the discussions about special stateful encryption in the full version.

Definition 4.1. A (stateless) symmetric-key (bit) encryption scheme with specification $\mathbf{E}_{\mathrm{SPEC}} := (\mathsf{KG}_{\mathrm{SPEC}}, \mathsf{Enc}_{\mathrm{SPEC}}, \mathsf{Dec}_{\mathrm{SPEC}})$ is subversion-resistant in the offline watchdog model if there exists a PPT watchdog $\mathcal{W}$ so that, for any PPT adversary $\mathcal{A}$ playing the game described in Figure 8, either

$$\mathbf{Adv}_{\mathcal{A}} \text{ is negligible,} \quad \text{or} \quad \mathbf{Det}_{\mathcal{W},\mathcal{A}} \text{ is non-negligible.}$$

where $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = |\Pr[b_C = 1] - 1/2|$, and,

$$\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda) = \left|\Pr[\mathcal{W}^{\mathbf{E}_{\mathrm{IMPL}}}(1^\lambda) = 1] - \Pr[\Pr[\mathcal{W}^{\mathbf{E}_{\mathrm{SPEC}}}(1^\lambda) = 1]\right|.$$

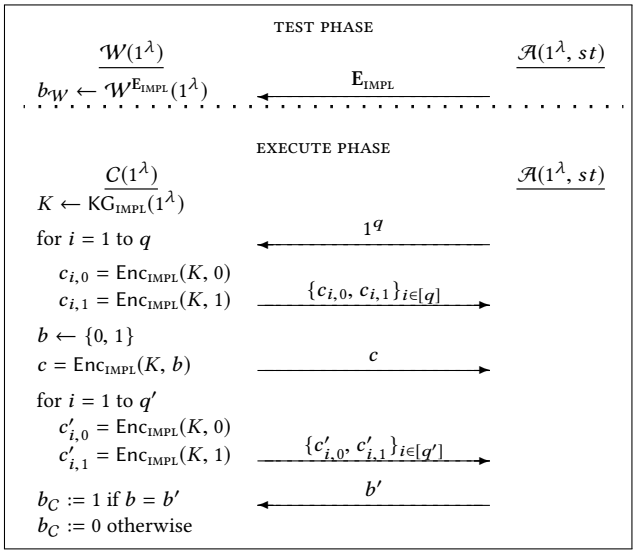


Figure 8: Subversion-resistant symmetric-key bit encryption. (The stateless case.), where $\mathbf{E}_{\mathrm{IMPL}} := (\mathsf{KG}_{\mathrm{IMPL}}, \mathsf{Enc}_{\mathrm{IMPL}}, \mathsf{Dec}_{\mathrm{IMPL}})$

Correctness under subversion is an overlooked (or simply assumed, e.g., in [4], as the "decryptability" assumption), but fundamental property. One can imagine that the adversary "hates" a certain message m that is unknown to the watchdog (e.g., "your cryptosystem is subverted"). The subverted implementation can then check whether the plaintext matches m and, if so, $\mathsf{Dec}_{\mathrm{IMPL}}$ outputs an arbitrary value other than m. This can be used by the adversary to effectively implement censorship. We say a public key encryption scheme is *correct* under subversion if the following holds: $\forall m$,

$$\Pr\left[\begin{array}{l}\mathsf{Dec}_{\mathrm{IMPL}}(sk, c) \neq m : \\ C \leftarrow \mathsf{Enc}_{\mathrm{IMPL}}(pk, m), (pk, sk) \leftarrow \mathsf{KG}_{\mathrm{IMPL}}(1^\lambda)\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is over the coins used in $\mathsf{KG}_{\mathrm{IMPL}}$ and $\mathsf{Enc}_{\mathrm{IMPL}}$.

Subversion resistant symmetric-key bit encryption. We proceed to design a specification for any secure symmetric-key bit encryption scheme so that the subliminal channels in all of the algorithms can be destroyed. With the general tools we developed in Sec. 3.2, we will "immunize" the algorithms one by one.

First, the (symmetric-key) key generation algorithm simply takes the security parameter as input and produces a random element in the key space. Following Thm. 3.4 directly, the specification $\mathsf{KG}_{\text{SPEC}} := (\mathsf{KG}^0_{\text{SPEC}}, \mathsf{KG}^1_{\text{SPEC}}, \Phi^{\mathsf{KG}}_{\text{SPEC}})$ (see Fig. 9) is stego-free; here $\mathsf{KG}^0_{\text{SPEC}}, \mathsf{KG}^1_{\text{SPEC}}$ both output random elements in the key space $\mathcal{K}$, and $\Phi^{\mathsf{KG}}_{\text{SPEC}} : \mathcal{K} \times \mathcal{K} \to \mathcal{K}$, is modeled as a random oracle. As discussed above, the random oracle can be removed if we allow randomness generation to be further subdivided; see appendix. **??**.

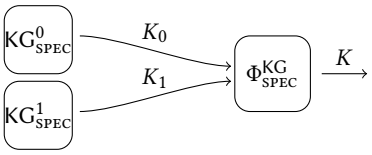


Figure 9: Stego-free spec for symmetric key generation

Next, consider the encryption algorithm; we focus first on bit encryption. In this case, the encryption algorithm takes three inputs: a security parameter, a small domain input (a bit), and a pair given by the random coin and the key, which come from a public input distribution $\mathsf{RG}_{\text{IMPL}} \times \mathsf{KG}_{\text{IMPL}}$. From Corollary 3.6, the specification $\mathsf{Enc}_{\text{SPEC}} := (\mathsf{RG}_{\text{SPEC}}, \mathsf{dEnc}_{\text{SPEC}})$ (as described in Fig. 10), where $\mathsf{RG}_{\text{SPEC}} := (\mathsf{RG}^0_{\text{SPEC}}, \mathsf{RG}^1_{\text{SPEC}}, \Phi^{\mathsf{RG}}_{\text{SPEC}})$ defined as in Fig. 6, is stego-free if Φ_{SPEC} is assumed to be a random oracle.

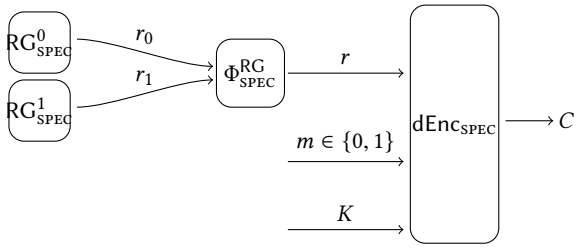


Figure 10: Stego-free specification for encryption algorithm in a symmetric-key bit encryption scheme.

Finally, consider the decryption algorithm. It does not influence CPA security, but directly influences correctness. Fortunately, when considering bit encryption only, the decryption algorithm is deterministic with a public input distribution. To see this, the decryption algorithm will input a key K (generated by $\mathsf{KG}_{\text{IMPL}}$) and a ciphertext C (generated by encrypting 0 or 1 using K with uniform coins). The watchdog can sample from the input distribution of $\mathsf{Dec}_{\text{IMPL}}$ to check the correctness, i.e., the consistency with $\mathsf{Dec}_{\text{SPEC}}$.

With all of the algorithms handled individually, we present the first general immunizing result for randomized encryption.

Theorem 4.2. *Given any stateless IND-CPA secure symmetric bit encryption scheme, the specification described above is subversion resistant and correct according to Def. 4.1.*

Proof. (sketch) The specification is described above (with the randomized algorithm split into randomness generation and a deterministic part, and the randomness generation split into two components together with an immunizing function as in Figure 6.).

Security. The watchdog is the combination of those described above: it guarantees that $\mathsf{RG}_{\text{SPEC}}$ is stego-free (making samples to observe collisions) and guarantees that $\mathsf{dEnc}_{\text{IMPL}}$ is consistent with the specification on inputs sampled from $\mathsf{KG}_{\text{IMPL}} \times \mathsf{RG}_{\text{IMPL}}$ (with 0 and 1); cf. Lemma 3.3. Here we only sketch the game changes and explain the negligible differences arising during the game transitions conditioned the watchdog's result.

Game-0 is the original game as described in Figure 8 with a trusted combining; Game-1 is the same as Game-0 except $\mathsf{KG}_{\text{IMPL}}$ is replaced with $\mathsf{KG}_{\text{SPEC}}$; Game-2 is the same as Game-1 except $\mathsf{Enc}_{\text{IMPL}}$ is replaced with $\mathsf{Enc}_{\text{SPEC}}$.

The adversary's advantage in Game-0 and Game-1 are negligibly close because of the stego-freeness of $\mathsf{KG}_{\text{SPEC}}$. To see this, a simulator can simulate the rest of the games in Game-0 (or Game-1) after receiving K sent by the challenger in the game defining stego-freeness for $\mathsf{KG}_{\text{SPEC}}$. If one can distinguish Game-0 from Game-1, then the simulator can easily tell apart $\mathsf{KG}_{\text{IMPL}}$ from $\mathsf{KG}_{\text{SPEC}}$ (even if the challenger does the composition). Similarly, we can argue Game-1 and Game-2 are indistinguishable because $\mathsf{Enc}_{\text{SPEC}}$ is stego-free. The fact that $\mathsf{Enc}_{\text{SPEC}}$ is stego-free follows from Corollary 3.6.

Every algorithm used in Game-2 is faithfully implemented; thus the IND-CPA security of the underlying encryption scheme holds.

Regarding correctness. As described above, the decryption algorithm for a bit encryption scheme has a public input distribution. The watchdog can sample random keys and check whether $\mathsf{Dec}_{\text{IMPL}}$ works properly for bit encryption. Following Lemma 3.3, we conclude that: $\forall b \in \{0,1\}$,

$$\Pr\left[\mathsf{Dec}_{\text{IMPL}}(K, \mathsf{Enc}_{\text{SPEC}}(K,b)) \neq b : K \leftarrow \mathsf{KG}_{\text{IMPL}}\right] \leq \mathsf{negl}(\lambda),$$

while for the encryption algorithm, $\mathsf{Enc}_{\text{IMPL}}$ can be used interchangeably with $\mathsf{Enc}_{\text{SPEC}}$. Thus, $\forall b \in \{0,1\}$,

$$\Pr[\mathsf{Dec}_{\text{IMPL}}(\mathsf{Enc}_{\text{IMPL}}(K,b)) \neq b : K \leftarrow \mathsf{KG}_{\text{IMPL}}] \leq \mathsf{negl}(\lambda)\,.$$

Combining these yields the statement of the theorem. □

Subversion resistant symmetric encryption with large message spaces. For large message spaces, the security game must allow the adversary to query; see Fig 13 in appendix. B. As mentioned in Sec 3.1, this immediately invites an input-trigger attack. Specifically, for a particular query m_i (chosen randomly by A during subversion), the subverted encryption algorithm may directly output the secret key; the same threat exists for stateful *bit* encryption, where a particular sequence of encryption bits may act as a trigger. Note that this simply cannot be detected by a PPT watchdog (making polynomially many samples). Furthermore, the same attack can be launched on $\mathsf{Dec}_{\text{IMPL}}$ to ruin correctness (as $\mathsf{Dec}_{\text{IMPL}}$ can output a different value). This suggests the principle that a subverted implementation should never be given a "naked" input generated by the adversary, e.g., the queried message.

However, observe that if the input message is forced to come from a known distribution U (e.g., the uniform distribution), the watchdog can check consistency between $\mathsf{dEnc}_{\text{IMPL}}$ and $\mathsf{dEnc}_{\text{SPEC}}$ on U (we ignore other inputs for simplicity, as they are from a public input distribution). Indeed, the watchdog can guarantee that with an overwhelming probability $\mathsf{dEnc}_{\text{IMPL}}$ is as good as $\mathsf{dEnc}_{\text{SPEC}}$ when the input is sampled according to U. Now, to defend against an input-trigger attack, we must ensure that the probability that any *particular* m causes an inconsistency is negligible. This sort of "worst-case vs. average-case" relationship arises in the theory

of *self-correcting programs* [19], where one wants to transform a program with a negligible fraction of bad inputs into a negligible error probability for every input. With these observations, we return to the large-message space challenge.

Constructions. First, we consider bit-wise encryption. When encrypting a message $m = m_1 \dots m_\ell$ for $\ell = |m|$, the user generates the ciphertext by calling $\mathsf{Enc}_{\text{IMPL}}$ ℓ times, yielding $C := (c_1, \dots, c_\ell)$, where $c_i = \mathsf{Enc}_{\text{IMPL}}(K, m_i)$. Since the bit encryption scheme we developed above preserves IND-CPA security; IND-CPA security follows via a simple hybrid argument. (Note, however, that in this case it is important that the encryption algorithm is stateless.)

To develop a more efficient scheme that can further handle long message (also stateful encryption) we augment the model by permitting the user (i.e., the challenger) to carry out *one trusted addition* operation [9] for each encryption and decryption, see Fig. 11. (We continue to assume trusted combining, as before.) We augment the specification of the encryption algorithm with a random message generator $\mathsf{MG}_{\text{SPEC}}$. Specifically, the specification of the encryption algorithm $\mathsf{Enc}_{\text{SPEC}}$ has the form $(\mathsf{RG}_{\text{SPEC}}, \mathsf{dEnc}_{\text{SPEC}}, \mathsf{MG}_{\text{SPEC}})$, where $\mathsf{MG}_{\text{SPEC}}$ has the specification $(\mathsf{MG}^0_{\text{SPEC}}, \mathsf{MG}^1_{\text{SPEC}}, \Phi^{\mathsf{MG}}_{\text{SPEC}})$, and $\mathsf{RG}_{\text{SPEC}}$ is as before (as in Fig. 6). When encrypting a message m, the user first runs $\mathsf{MG}_{\text{IMPL}}$ (the implementation) to sample a random message u, and computes $m' = m \oplus u$. The user will call $\mathsf{Enc}_{\text{IMPL}}$ to encrypt the new message m' and obtains the ciphertext c'. This includes calls to $\mathsf{KG}_{\text{IMPL}}$, $\mathsf{RG}_{\text{IMPL}}$ and passing the corresponding outputs K, r together with m' to $\mathsf{dEnc}_{\text{IMPL}}$; see Figure 11. Observe that m' is a *uniformly chosen message* (as the watchdog can ensure that u is safely generated). The new ciphertext C now includes u together with the ciphertext c'. For decryption, the user first runs $\mathsf{Dec}_{\text{IMPL}}$ on c' and obtains m'; then the user computes $m = m' \oplus u$.

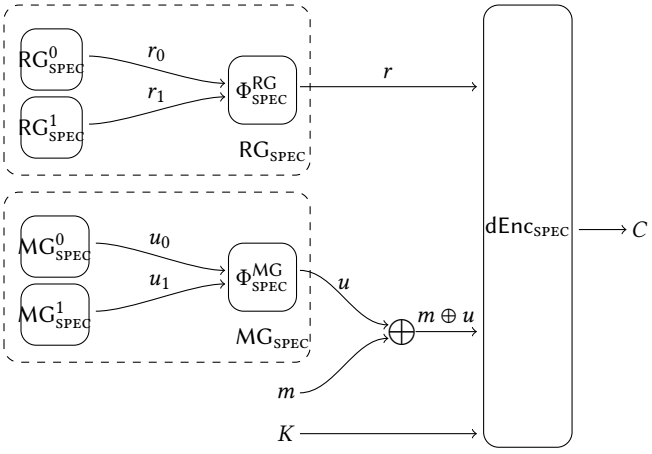


Figure 11: Stego-free encryption specification supporting large messages, where $K \leftarrow \mathsf{KG}_{\text{IMPL}}$.

Security analysis. The intuition that this simple method works is as follows: First, we generalize Theorem 4.2 that symmetric-key encryption for *random messages* are also subversion resistant. To see this, we analyze the stego-freeness algorithm by algorithm. The key generation is the same as the bit encryption. The encryption algorithm now takes input a uniform message, together with the key and security parameter. It means that the encryption algorithm (the deterministic part $\mathsf{dEnc}_{\text{SPEC}}$) now takes inputs from public input distributions, i.e., $\mathsf{KG}_{\text{IMPL}} \times \mathsf{RG}_{\text{IMPL}} \times \mathcal{U}_{\mathcal{M}}$, where $\mathcal{U}_{\mathcal{M}}$ is the uniform distribution over message space $\mathcal{M}$. Following Theorem 3.5, such encryption algorithm will be stego-free as long as the specification is designed as Figure 7. Now for the decryption algorithm, since the encryption is for uniform messages, thus the decryption algorithm now also takes a public input distribution. Next, we show the encryption specification defined in Figure 11 indeed takes uniform messages as input. (1.) Following Theorem 3.4, the uniform message sampler $\mathsf{MG}_{\text{SPEC}}$ is stego-free. (2.) With the trusted addition operation, when a that $\mathsf{Enc}_{\text{IMPL}}$ takes as input will be $m' = m \oplus u$, where u looks uniform even to the adversary, thus m' would look uniform to $\mathsf{Enc}_{\text{IMPL}}$ (actually the deterministic part $\mathsf{dEnc}_{\text{IMPL}}$). Similar to the analysis of Theorem 4.2, we can show a stronger result that handles the correctness and subversion resistance for symmetric-key encryption supporting long messages.

[9] We remark here that such addition operation can be considered to be from user's basic computing infrastructure, and furthermore, such extra assumption is still weaker than previous works, such as the reverse firewall.

Theorem 4.3. *For any IND-CPA secure symmetric-key encryption, the specification described as in Fig. 11 is subversion resistant and correct according to Def. B.1, assuming a trusted $\oplus$ operation.*

4.2 Cliptographically IND-CPA secure PKE

Now we turn to public-key encryption, which follows fairly directly from the previous construction. The major difference arises with key generation, as asymmetric key generation has to be treated with more care than simple randomness generation; see Figure 12, which indicates the construction. Specifically, the basic techniques used for symmetric key encryption above can be adapted for public key encryption. Key generation must be considered as a randomized algorithm producing structured output (with only security parameter as input). With these tools at hand, we resolve the major open problem to construct a IND-CPA secure PKE when facing subversions that we asked at the beginning of the paper. We remark that the assumption of "trusted $\oplus$" in the theorem below can be removed if the message space is small.

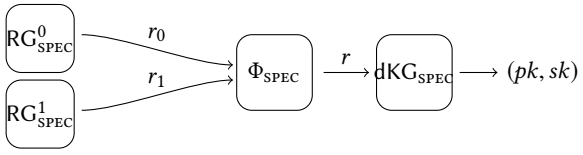


Figure 12: Stego-free asymmetric key generation.

Theorem 4.4. *For any IND-CPA secure public key bit encryption, there exists a specification such that it is subversion resistant and correct in the split-program model. Furthermore, for any IND-CPA public key encryption (supporting large input space), there exists a specification design such that it is subversion resistant and correct according to Def. B.2, if the user can do a trusted $\oplus$.*

5 CONCLUSIONS AND OPEN PROBLEMS

In this paper, we overcome the major obstacle in post-Snowden cryptography: We developed a general technique to eliminate subliminal channels in the subverted implementations, and we then

applied the technique to construct the first IND-CPA secure public key encryption scheme without relying on a trusted party. Furthermore, our symmetric key encryption scheme also advances the state of art in a way that we allow all algorithms to be subverted and we remove the unrealistic assumption of decryptability that was adopted in previous results.

Many interesting questions remain open, and we highlight a few below. Given that one of the major threats are due to the subverted randomness generation, the natural next question is whether we can generically *correct* a uniform distribution, for example, consider correcting a random oracle. The second one is whether we can apply our general technique to other scenarios where stenographic channels should be eliminated. One promising example might be to use our technique to construct collusion free protocols [15]. Third, our general technique can defend against the subliminal channel attack due to bad randomness, however the other simpler hidden-trigger attack is quite difficult to defend, e.g., in the setting of digital signature. Existing work either require an online watchdog [20] or has to require a trusted key generation with a verifiability assumption [1]. Giving a systematic study of defending mechanisms against a hidden-trigger would be important. Last but not least, we mostly consider stateless algorithms in this paper, in practice, some algorithms might be stateful, e.g., a counter mode encryption. Extending our research to the setting of stateful algorithms will be with both theoretical and practical importance.

REFERENCES

[1] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. 2015. Subversion-Resilient Signature Schemes. In *ACM CCS 15*, Indrajit Ray, Ninghui Li, and Christopher Kruegel: (Eds.). ACM Press, 364–375.

[2] Mihir Bellare and Viet Tung Hoang. 2015. Resisting Randomness Subversion: Fast Deterministic and Hedged Public-Key Encryption in the Standard Model. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 627–656. https://doi.org/10.1007/978-3-662-46803-6_21

[3] Mihir Bellare, Joseph Jaeger, and Daniel Kane. 2015. Mass-surveillance without the State: Strongly Undetectable Algorithm-Substitution Attacks. In *ACM CCS 15*, Indrajit Ray, Ninghui Li, and Christopher Kruegel: (Eds.). ACM Press, 1431–1440.

[4] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. 2014. Security of Symmetric Encryption against Mass Surveillance. In *CRYPTO 2014, Part I (LNCS)*, Juan A. Garay and Rosario Gennaro (Eds.), Vol. 8616. Springer, Heidelberg, 1–19. https://doi.org/10.1007/978-3-662-44371-2_1

[5] Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, Fuchun Guo, and Mingwu Zhang. 2016. Cryptographic Reverse Firewall via Malleable Smooth Projective Hash Functions. In *ASIACRYPT 2016, Part I (LNCS)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.), Vol. 10031. Springer, Heidelberg, 844–876. https://doi.org/10.1007/978-3-662-53887-6_31

[6] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. 2015. A More Cautious Approach to Security Against Mass Surveillance. In *FSE 2015 (LNCS)*, Gregor Leander (Ed.), Vol. 9054. Springer, Heidelberg, 579–598. https://doi.org/10.1007/978-3-662-48116-5_28

[7] Yvo Desmedt. 1988. Subliminal-Free Authentication and Signature (Extended Abstract). In *EUROCRYPT'88 (LNCS)*, C. G. Günther (Ed.), Vol. 330. Springer, Heidelberg, 23–33.

[8] Yvo Desmedt. 1990. Abuses in Cryptography and How to Fight Them. In *CRYPTO'88 (LNCS)*, Shafi Goldwasser (Ed.), Vol. 403. Springer, Heidelberg, 375–389.

[9] Docker.Inc. 2013. Docker. (2013). https://www.docker.com/.

[10] Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. 2015. A Formal Treatment of Backdoored Pseudorandom Generators. In *EUROCRYPT 2015, Part I (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9056. Springer, Heidelberg, 101–126. https://doi.org/10.1007/978-3-662-46800-5_5

[11] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. 2016. Message Transmission with Reverse Firewalls—Secure Communication on Corrupted Machines. In *CRYPTO 2016, Part I (LNCS)*, Matthew Robshaw and Jonathan Katz (Eds.), Vol. 9814. Springer, Heidelberg, 341–372. https://doi.org/10.1007/978-3-662-53018-4_13

[12] Carl M. Ellison, Roger A. Golliver, Howard C. Herbert, Derrick C. Lin, Francis X. McKeen, Gilbert Neiger, Ken Reneris, James A. Sutton, Shreekant S. Thakkar, and Millind Mittal. 2000. Controlling access to multiple isolated memories in an isolated execution environment Controlling access to multiple isolated memories in an isolated execution environment. (2000). https://www.google.com/patents/US6678825.

[13] Nicholas J. Hopper, John Langford, and Luis von Ahn. 2002. Provably Secure Steganography. In *CRYPTO 2002 (LNCS)*, Moti Yung (Ed.), Vol. 2442. Springer, Heidelberg, 77–92.

[14] Immunix. 1998. AppArmor. (1998). http://wiki.apparmor.net/index.php/Main_Page.

[15] Matt Lepinski, Silvio Micali, and abhi shelat. 2005. Collusion-free protocols. In *37th ACM STOC*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 543–552.

[16] Ramya Jayaram Masti, Devendra Rai, Claudio Marforio, and Srdjan Capkun. 2014. Isolated Execution on Many-core Architectures. Cryptology ePrint Archive, Report 2014/136. (2014). http://eprint.iacr.org/2014/136.

[17] Ilya Mironov and Noah Stephens-Davidowitz. 2015. Cryptographic Reverse Firewalls. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 657–686. https://doi.org/10.1007/978-3-662-46803-6_22

[18] Mendel Rosenblum. 2004. The Reincarnation of Virtual Machines The Reincarnation of Virtual Machines. *ACM Queue 2*, 15 (2004), 34–40.

[19] Ronitt A. Rubinfeld. 1991. *A Mathematical Theory of Self-checking, Self-testing and Self-correcting Programs.* Ph.D. Dissertation. University of California at Berkeley, Berkeley, CA, USA. UMI Order No. GAX91-26752.

[20] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. 2016. Cliptography: Clipping the Power of Kleptographic Attacks. In *ASIACRYPT 2016, Part II (LNCS)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.), Vol. 10032. Springer, Heidelberg, 34–64. https://doi.org/10.1007/978-3-662-53890-6_2

[21] Gustavus J. Simmons. 1983. The Prisoners' Problem and the Subliminal Channel. In *CRYPTO'83*, David Chaum (Ed.). Plenum Press, New York, USA, 51–67.

[22] Gustavus J. Simmons. 1986. A Secure Subliminal Channel (?). In *CRYPTO'85 (LNCS)*, Hugh C. Williams (Ed.), Vol. 218. Springer, Heidelberg, 33–41.

[23] NTT Data Corporation Tomoyo Linux. 2009. Tomoyo. (2009). http://tomoyo.osdn.jp/.

[24] Amit Vasudevan, Jonathan M. McCune, James Newsome, Adrian Perrig, and Leendert van Doorn. 2012. CARMA: a hardware tamper-resistant isolated execution environment on commodity x86 platforms. In *ASIACCS 12*, Heung Youl Youm and Yoojae Won (Eds.). ACM Press, 48–49.

[25] Wikipedia. 2016. Software Guard Extensions. (2016). https://en.wikipedia.org/wiki/Software_Guard_Extensions.

[26] Adam Young and Moti Yung. 1996. The Dark Side of "Black-Box" Cryptography, or: Should We Trust Capstone?. In *CRYPTO'96 (LNCS)*, Neal Koblitz (Ed.), Vol. 1109. Springer, Heidelberg, 89–103.

[27] Adam Young and Moti Yung. 1997. Kleptography: Using Cryptography Against Cryptography. In *EUROCRYPT'97 (LNCS)*, Walter Fumy (Ed.), Vol. 1233. Springer, Heidelberg, 62–74.

A THE CLIPTOGRAPHIC MODEL, IN BRIEF

We will mostly follow the definitional framework introduced in [20]. It is meant to capture a situation where an adversary has the opportunity to implement (and, indeed, "mis-implement" or subvert) our basic cryptographic tools. On the other hand, the adversary prefers to keep the subversion under the radar of any detection; this was reflected in the model via a checking component "watchdog" who will attempt to check, via black-box testing, that the cryptographic tools have been faithfully implemented by the adversary. The model, in brief:

Specification (see also ⋆ below.) The cryptographic primitive is *specified* as a tuple $\Pi_{\text{SPEC}} = (F^1_{\text{SPEC}}, \ldots, F^k_{\text{SPEC}})$ of "functionalities." Note that when specification contains multiple components, that means implementation of each component should be supplied. Each F^i_{SPEC} is either a function $F : \{0,1\}^* \to \{0,1\}^*$ (which specifies a deterministic algorithm) or a family of probability distributions, so that $F(x)$ is a probability distribution over $\{0,1\}^*$ for each x (which specifies a randomized algorithm).

Subversion. The adversary provides us with an "implementation" $\Pi_{\text{IMPL}} = (F^1_{\text{IMPL}}, \ldots, F^k_{\text{IMPL}})$ for **each** of the functionalities F^I_{IMPL}. Observe that the adversary may provide even the algorithms that generate randomness and random objects such as keys. Of course, in general the implementation for each component F^i_{IMPL} may disagree with the specification component F^i_{SPEC}, which can provide the adversary a novel avenue to attack the primitive. (The adversary, therefore, has a full subversion power over the functional cryptographic modules; Note that the basic non-cryptographic and simple firmware functions as well as the watchdog (below) are assumed to be correct since otherwise no computation or faithful checking may be possible).

Testing; the watchdog. The algorithmic and cryptographic building blocks are then sent to a trusted testing facility, the *watchdog*. The watchdog is aware of the official specification, and may query the adversary's implementations (treating each of the component as a block-box) in an attempt to detect disagreements between Π_{IMPL} and Π_{SPEC}.

The security game. After that the watchdog is satisfied, the implementations Π_{IMPL} are pressed into service, [10] then their security is modeled by a conventional security game, except that the challenger uses Π_{IMPL}.

★ **Remark: modular design.** We permit the designer of the cryptographic primitive (who determines its specification) an extra dimension of freedom which can assist the watchdog in his verification task: We permit the designer to functionally *decompose* the primitives into a fixed number of "pieces.", which is commonplace in software engineering where a big program with well defined pieces is simply designed to have a module for each piece and a simple program which just calls the pieces. We call this *split-program model*. For example, rather than specifying a function of interest $f : X \to Y$, the designer may instead specify two functions $h : X \to W$ and $g : W \to Y$ with the property that $f = g \circ h$. (Thus h and g together specify f.) An important example in our setting is specifying a randomized algorithm $\mathsf{G}(x)$ as a composition $\mathsf{dG}(x, \mathsf{RG}(1^\lambda))$, where dG is deterministic and RG, given the input 1^λ, produces λ uniformly random bits as output. In general, the decomposition may be arbitrary, but may only involve $O(1)$ pieces (in practice these composition is going to be natural as it follows the crypto system control flow). See right side of Fig. **??** and Sec. 2 for details.

There are two points we would like to emphasize. (i.) Once the decomposition is specified (simply long the control flow of the system), the implementation of an algorithm has to individually supply each component, otherwise the watchdog will trivially reject; (ii.) Such decomposition follows the various operations of the system naturally, and is no different from the modular design principle that is widely utilized in software engineering practice, explicitly or implicitly. For example, any encryption scheme, may, in fact, be defined arbitrarily, but is always described as containing three independent components: key generation, encryption, and decryption (the first two components may even have access to an external library for randomness generation in the actual system). See also Fig. 1 in Sec. 1.

[10] To ensure the implementations tested and used are the same, we may deploy simple methods such as code signing.

B OMITTED DEFINITIONS

B.1 Subversion resistant symmetric-key encryption with long messages

Definition B.1. For any randomized symmetric-key encryption scheme, consider a specification $\mathbf{E}_{\text{SPEC}} := (\mathsf{KG}_{\text{SPEC}}, \mathsf{Enc}_{\text{SPEC}}, \mathsf{Dec}_{\text{SPEC}})$. We say specification $\mathbf{E}_{\text{SPEC}}$ is *subversion resistant with a trusted addition and amalgamation in the offline watchdog model*, [11] if there exists an offline PPT watchdog $\mathcal{W}$, for any PPT adversary $\mathcal{A}$, playing the game defined in Figure 13, either,

$$\mathbf{Adv}_{\mathcal{A}} \text{ is negligible, or, } \mathbf{Det}_{\mathcal{W},\mathcal{A}} \text{ is non-negligible.}$$

where $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = |\Pr[b_C = 1] - \frac{1}{2}|$, and

$$\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda) = |\Pr[\mathcal{W}^{\mathbf{E}_{\text{IMPL}}}(1^\lambda) = 1] - \Pr[\mathcal{W}^{\mathbf{E}_{\text{SPEC}}}(1^\lambda) = 1]|,$$

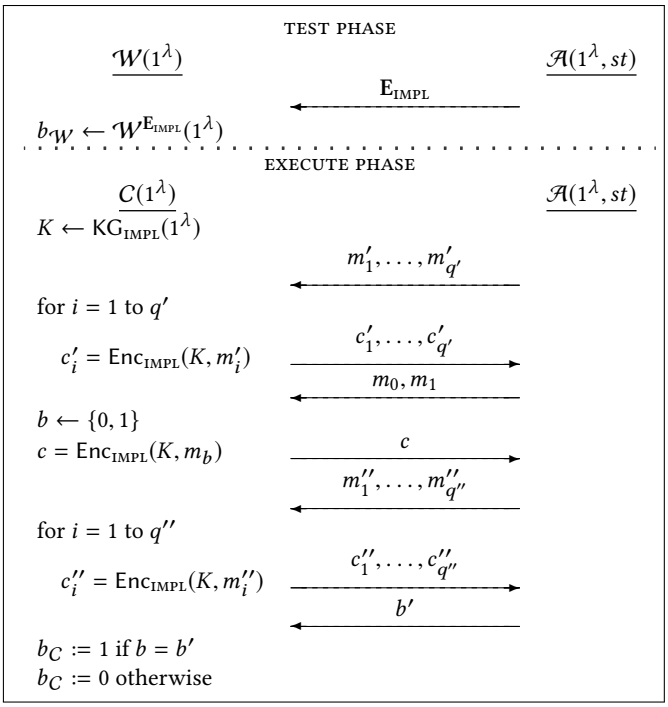


Figure 13: Subversion-resistant symmetric-key encryption with trusted addition, where $\mathbf{E}_{\text{IMPL}} := (\mathsf{KG}_{\text{IMPL}}, \mathsf{Enc}_{\text{IMPL}}, \mathsf{Dec}_{\text{IMPL}})$

Definition B.2. For any public key encryption scheme, consider a specification $\mathbf{E}_{\text{SPEC}} := (\mathsf{KG}_{\text{SPEC}}, \mathsf{Enc}_{\text{SPEC}}, \mathsf{Dec}_{\text{SPEC}})$. We say specification $\mathbf{E}_{\text{SPEC}}$ is *subversion resistant in the offline watchdog model* if there exists an offline PPT watchdog $\mathcal{W}$, for any PPT adversary $\mathcal{A}$, such that, at least one of the following conditions hold in the game defined in Figure 2:

[11] We remark here that for the succinctness of the definition, we put the trusted addition operation implicitly in the specifications, and it will be carried out by the challenger.

$\mathbf{Det}_{\mathcal{W},\mathcal{A}}$ is non-negligible, or, $\mathbf{Adv}_{\mathcal{A}}$ is negligible,
where the detection probability of the watchdog is defined as:

$$\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda) = |\Pr[\mathcal{W}^{\mathbf{E}_{\text{IMPL}}}(1^\lambda) = 1] - \Pr[\mathcal{W}^{\mathbf{E}_{\text{SPEC}}}(1^\lambda) = 1]|,$$

and, the adversary's advantage is defined as:

$$\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = |\Pr[b_C = 1] - \frac{1}{2}|.$$

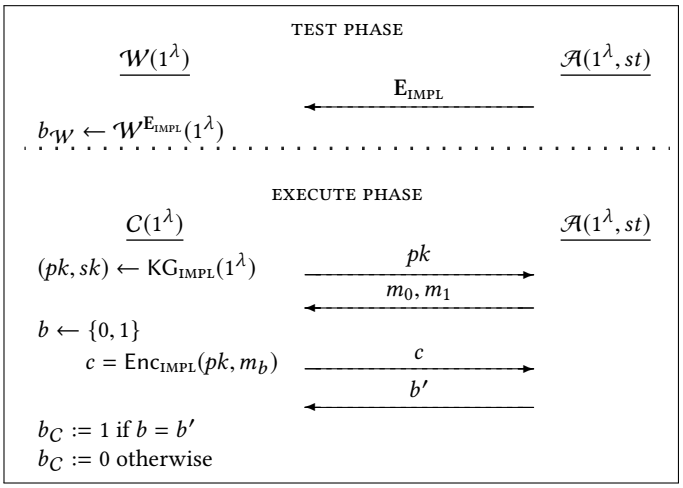


Figure 14: Subversion-resistant public key encryption supporting large messages, where $\mathbf{E}_{\text{IMPL}} := (\mathsf{KG}_{\text{IMPL}}, \mathsf{Enc}_{\text{IMPL}}, \mathsf{Dec}_{\text{IMPL}})$.

We say a public key encryption scheme is *correct* under subversion if the following holds: $\forall m$,

$$\Pr\left[\begin{array}{l} \mathsf{Dec}_{\text{IMPL}}(sk, c) \neq m : \\ C \leftarrow \mathsf{Enc}_{\text{IMPL}}(pk, m), (pk, sk) \leftarrow \mathsf{KG}_{\text{IMPL}}(1^\lambda) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the probability is over the coins used in $\mathsf{KG}_{\text{IMPL}}$ and $\mathsf{Enc}_{\text{IMPL}}$.

C AN ATTACK ON SINGLE-SOURCE RANDOMNESS PRIMITIVES

Subverted randomness generation attack: In the following attack on pubic key encryption, the adversary honestly implements the key generation and decryption, and only subverts the encryption algorithm. Suppose the specification of the (public-key) encryption algorithm is defined as $\mathsf{Enc}_{\text{SPEC}} := (\mathsf{RG}_{\text{SPEC}}, \mathsf{dEnc}_{\text{SPEC}}, \Phi_{\text{SPEC}})$. The meaning of each component is self-evident: $\mathsf{RG}_{\text{SPEC}}$ generates uniformly random bits r_0, the function Φ_{SPEC} "cleans" r_0 to produces the final random bits r, and, finally, $\mathsf{dEnc}_{\text{SPEC}}$ takes the random bits r, the encryption key pk, and the message bit as inputs and produces a ciphertext C.

The attack: The adversary $\mathcal{A}$ first randomly chooses a backdoor z, and prepares a subverted implementation $\mathsf{Enc}^z_{\text{IMPL}}$ which is composed of $(\mathsf{RG}_{\text{IMPL}}, \mathsf{dEnc}_{\text{IMPL}}, \Phi_{\text{IMPL}})$ with the backdoor embedded. In particular, $\mathsf{RG}_{\text{IMPL}} := \mathsf{RG}^z_{\text{IMPL}}$ carries out rejection sampling to ensure that the ciphertext encrypting 0 and the ciphertext encrypting 1 can be distinguished by applying a PRF (using z as the key); the algorithms $\mathsf{dEnc}_{\text{IMPL}}$ and Φ_{IMPL} are honestly implemented (that is, identical to the specifications). Later the adversary $\mathcal{A}$ can easily learn secret information (indeed, the plaintext) from the ciphertext generated by the subverted algorithms by applying the PRF (using her backdoor z as the key). See Figure 15 for detailed description.

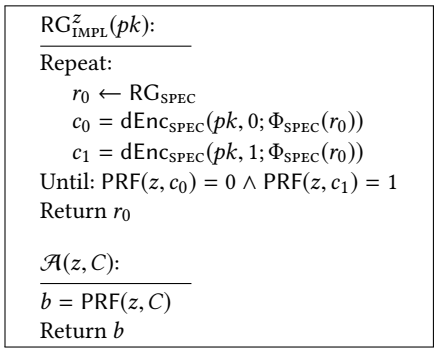


Figure 15: Subverted randomness generation and the message recovery algorithms

Security analysis. Due to the rejection sampling condition, it is easy to see that the adversary defined in Figure 15 can determine the plaintext bit perfectly from the ciphertext. As for the *detection probability*, the randomness output by $\mathsf{RG}_{\text{SPEC}}$ is a uniform λ-bit string; in contrast, the randomness output by $\mathsf{RG}_{\text{IMPL}}$ is a string selected uniformly from a (random) subset S of $\{0,1\}^\lambda$ (determined by the PRF). The subset S consists of all strings that carry 0 and 1 to ciphertexts satisfying a criterion given by the PRF. Let us think of the PRF as a random function, that means the rejection sampling condition will be satisfied with probability $1/4$ for each r_0 uniformly sampled. Essentially, we can consider S as a random subset of $\{0,1\}^\lambda$ with (expected) size $2^{\ell-2}$. If there is no collision when $\mathcal{W}$ asks q queries, then the q different bit strings observed by $\mathcal{W}$ can come from either of the whole space $\{0,1\}^\lambda$ or a subset S (with size larger than q). This means conditioned on no collision, no watchdog can tell apart $\mathsf{RG}_{\text{IMPL}}$ from $\mathsf{RG}_{\text{SPEC}}$. Using the total probability theorem, we can bound the distinguishing advantage by the probability that a collision appears in q queries when sample from S.

Proposition C.1. *For any CPA-secure (public-key) encryption scheme, for any public function Φ, the adversary $\mathcal{A}$ shown in Figure 15 can learn the plaintext with probability 1 given the ciphertext generated by Enc^z_{IMPL} even if the randomness generation is separated and immunized by a random oracle. Furthermore, suppose RG_{IMPL} outputs ℓ bits of randomness; then the detection advantage is $q^2/2^{\ell-4} + \mathsf{negl}(\lambda)$ for all PPT watchdogs making q queries during the interrogation, assuming* PRF *is a secure pseudorandom function.*

Proof. The effectiveness of $\mathcal{A}$ is straightforward, since the randomness generated by $\mathsf{RG}_{\text{IMPL}}$ makes the ciphertext to be distinguishable using the PRF, thus the adversary $\mathcal{A}$ who knows the backdoor recovers the message bit perfectly.

Next, we will argue that no offline watchdog can notice the subversion, particularly $\mathsf{RG}_{\text{IMPL}}$. (All other components are honestly implemented). We define the game sequence as follows:

Game-0 is the game that $\mathcal{W}$ is interacting with $\mathsf{RG}_{\text{IMPL}}$.

Game-1 is the same as Game-0 except that the PRF used in $\mathsf{RG}_{\text{IMPL}}$ is replaced with a random function R.

Game-2 is the same as Game-1 except that $\mathsf{RG}_{\mathrm{IMPL}}$ resamples, if $\mathcal{W}$ notices a collision in the responses to the q queries.

Game-3 is the same as Game-2 except that $\mathsf{RG}_{\mathrm{IMPL}}$ is replaced with $\mathsf{RG}_{\mathrm{SPEC}}$.

Game-4 is the same as Game-3 except removing the resampling condition.

Analyzing the gaps, we see: Game-0 and Game-1 are indistinguishable because of the PRF security; Game-1 and Game-2 are identical, if there is no collision among the q queries in Game-2. In Game 2, for any r_0, the probability that $R(c_0) = 0 \wedge R(c_1) = 1$ is $1/4$, where c_0, c_1 are ciphertexts encrypting $0, 1$ respectively using $\Phi(r_0)$ as the coin. Suppose S is the set that contains all the values that satisfies $R(c_0) = 0 \wedge R(c_1) = 1$, then the expected size of S, $E[|S|] = 2^\ell/4 = 2^{\ell-2}$. It follows that with negligible probability, $|S| \leq \frac{E[|S|]}{2} = 2^{\ell-3}$. Then the probability that there exists a collision among q samples from S would be bounded by $q^2/|S| \leq q^2/2^{\ell-3}$. Game-2 and Game-3 is identical, since the responses can either appear in a random subset or the whole space. Game-3 and Game-4 are identical except there is collision when sampling q uniform points. The probability of such collision exist is $q^2/2^\ell$. Combining them above, we have the proposition. □

Remark C.1. *We remark that for this particular attack we assume that the implementation has access to a public key—this yields an intuitively natural attack against a encryption scheme which permits full recovery of the message. However, the basic structure of the attack can be adapted to randomized algorithms in generality.*

D OMITTED PROOFS

Theorem 3.5. *For any randomized algorithm* G*, consider the specification* $\mathsf{G}_{SPEC} := (\mathsf{RG}_{SPEC}, \mathsf{dG}_{SPEC})$*, where* RG_{SPEC} *generates* $\lambda = \lambda(n)$ *bits of uniform randomness and* dG_{SPEC} *is deterministic. We define* RG_{SPEC} *to be* $(\mathsf{RG}^0_{SPEC}, \mathsf{RG}^1_{SPEC}, \Phi_{SPEC})$ *as above. If (1.)* $\mathsf{RG}^0_{SPEC}(1^\lambda)$ *and* $\mathsf{RG}^1_{SPEC}(1^\lambda)$ *output* λ *uniform bits; (2.)* Φ_{SPEC} *takes* $r_0 \circ r_1$ *as input, and outputs* r *(so it maps strings of length* 2λ *to strings of length* λ*) (see Fig. 7), then* G_{SPEC} *is stego-free with a trusted combining. Here* Φ_{SPEC} *is modeled as a random oracle, and* $\mathsf{RG}^0_{IMPL}, \mathsf{RG}^1_{IMPL}$ *are executed independently.*

Proof. The watchdogs will be a combination of the ones in Theorem 3.4 and Lemma 3.3 to guarantee unpredictability of $\mathsf{RG}^b_{\mathrm{IMPL}}$ and the overwhelming consistency for deterministic algorithms with a public input distribution. Here $\mathsf{dG}_{\mathrm{SPEC}}$ is a deterministic algorithm and the output of $\mathsf{RG}_{\mathrm{SPEC}} \times \mathsf{IG}$ would be the input distribution to $\mathsf{dG}_{\mathrm{SPEC}}$.

We here only describe briefly about the game changes.

Game-0 is the same as Figure 4, the adversary $\mathcal{A}$ prepares every piece of the implementation, and the challenger simply calls them and passes the inputs to the next implementation as defined (doing the basic amalgamation).

Game-1 is the same as Game-0 except that the randomness r is uniformly sampled by C.

Game-2 is the same as Game-1 except that $\mathsf{dG}_{\mathrm{IMPL}}$ is replaced with $\mathsf{dG}_{\mathrm{SPEC}}$.

Note that in Game-0, it corresponds to $b = 0$, while in Game-2, every implementation of the algorithm (except input generation) is now the specification, it corresponds to $b = 1$.

From Theorem 3.4, with a trusted amalgamation, the output from the implementation $\mathsf{RG}_{\mathrm{IMPL}} := (\mathsf{RG}^0_{\mathrm{IMPL}}, \mathsf{RG}^1_{\mathrm{IMPL}}, \Phi_{\mathrm{IMPL}})$ is pseudorandom to the adversary $\mathcal{A}$ who made $\mathsf{RG}_{\mathrm{IMPL}}$. Thus in the security game defined in Figure 4, we can let the challenger simply generate r uniformly to reach Game-1.

From Lemma 3.3, $\mathsf{dG}_{\mathrm{SPEC}}$ will be a deterministic algorithm with a public input distribution, thus $\mathsf{dG}_{\mathrm{IMPL}}$ would be consistent with $\mathsf{dG}_{\mathrm{SPEC}}$ with an overwhelming probability when inputs are sampled according to $\mathsf{RG}_{\mathrm{SPEC}} \times \mathsf{IG}$, thus Game-2 can be reached with only a negligible gap.

Once in Game-2, all components are specification. □