Subversion-Resilient Public Key Encryption with Practical Watchdogs

Pascal Bemmann[1], Rongmao Chen[2], and Tibor Jager[1]

[1] University of Wuppertal,
{bemmann, tibor.jager}@uni-wuppertal.de
[2] School of Computer, National University of Defense Technology, Changsha, China,
chromao@nudt.edu.cn

Abstract. Restoring the security of maliciously implemented cryptosystems has been widely considered challenging due to the fact that the subverted implementation could arbitrarily deviate from the official specification. Achieving security against adversaries that can arbitrarily subvert implementations seems to inherently require trusted component assumptions and/or architectural properties. At ASIACRYPT 2016, Russell *et al.* proposed an attractive model where a watchdog is used to test and approve individual components of an implementation before or during deployment. Such a detection-based strategy has been useful for designing various cryptographic schemes that are provably resilient to subversion.
We consider Russell *et al.*'s watchdog model from a practical perspective regarding watchdog efficiency. We find that the asymptotic definitional framework, while permitting strong positive theoretical results, does not yet guarantee practical watchdogs, due to the fact that the running time of a watchdog is only bounded by an abstract polynomial. Hence, in the worst case, the running time of the watchdog might exceed the running time of the adversary, which seems impractical for most applications. We adopt Russell *et al.*'s watchdog model to the concrete security setting and design the first subversion-resilient public-key encryption scheme which allows for extremely efficient watchdogs with only *linear* running time.
At the core of our construction is a new variant of a combiner for key encapsulation mechanisms (KEMs) by Giacon *et al.* (PKC'18). We combine this construction with a new subversion-resilient randomness generator that also can be checked by an efficient watchdog, even in *constant* time, which could be of independent interest for the design of other subversion-resilient cryptographic schemes. Our work thus shows how to apply Russell *et al.*'s watchdog model to design subversion-resilient cryptography with efficient watchdogs. We insist that this work does not intend to show that the watchdog model outperforms other defense approaches, but to demonstrate that practical watchdogs are practically achievable.

The first author was supported by the research training group "Human Centered Systems Security" (NERD.NRW) sponsored by the state of North-Rhine Westphalia. The second author was supported by the National Natural Science Foundation of China (Grant No.62032005, No.61702541, No.61872087, No.61872089) and the Young Elite Scientists Sponsorship Program by China Association for Science and Technology (No.YESS20170128). The third author was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement 802823.

Keywords: Subversion-Resilience · Watchdog · Randomness Generator · Public Key Encryption.

1 Introduction

Modern cryptography has been a tremendous success due to its remarkable functionalities and security guarantees. Nevertheless, it has been of increasing concern that the robustness of security guarantees provided by cryptographic tools in practice may be weaker than thought. In particular, cryptosystems in the real world could be stealthily weakened by attackers who have the capability of tampering with the actual algorithm implementation for exfiltrating secret information while keeping it indistinguishable—in black-box testing—from a truthful one. Such a powerful adversary was originally considered in the *kleptographic* setting by Young and Yung [31, 32] over two decades ago while the striking Snowden revelations (in 2013) of massive surveillance of encrypted communication attracted renewed attention worldwide. News reports [14] have recently confirmed that a spying agency subverted a widely-used implementation of the Dual EC pseudorandom number generator, which was then also exploited by another government. This provides a concrete real-world example showing the demand for subversion-resilient cryptosystems.

Recent research showed that such impactful subversion attacks could be mounted on a large class of cryptographic primitives, including encryption schemes [6, 5, 15, 11], digital signatures [2] and non-interactive zero-knowledge proofs [3, 20]. More generally, Berndt and Liśkiewicz [7] proved that there exist universal algorithm substitution attacks on any cryptographic algorithm with sufficiently large min-entropy. These negative results, in turn, significantly increased the awareness of the research community and consequently, extensive efforts have been made towards effective approaches to subversion-resilient cryptography.

Since subversion attacks consider a much stronger adversary than many classical security models, it turns out that further reasonable assumptions are generally needed, since otherwise there is no much hope to achieve meaningful security guarantees [6, 5, 2, 4, 15, 27–29, 13, 1, 25, 12, 17, 19]. All the different assumptions considered in these works be they trusted components [25, 2, 12, 17, 10] or architectural requirements [27–29, 13], have their own plausibility, and thus are generally incomparable, as they rely on different assumptions and may be useful in different application contexts. Nevertheless, it is still widely considered meaningful to weaken the specific assumptions adopted within each model for more practical and/or broadly realizable subversion-resilience. As we will explain below, this is the main motivation of our work.

Subversion-resilience in the watchdog model. In this work, we consider the watchdog model introduced by Russell *et al.* [27] at ASIACRYPT 2016. This model has been useful to design a broad class of cryptographic tools secure against subversion attacks. The watchdog model is based on an architectural assumption, namely the *split-program methodology*, where an algorithm is decomposed into several functional components. The adversary supplies implementations of all components to a so-called “watchdog”, which performs a black-box test to decide whether the (possibly subverted) implementation of each component is sufficiently compliant with the specification to provide the expected security properties. A cryptographic scheme is said to be subversion-resilient, if *either* there exists a watchdog that can detect the subversion of components with *non-negligible* probability; *or* the security of the cryptographic scheme is preserved even if the subverted implementation is used. It is shown that, under the aforementioned architectural assumption, such a watchdog-based approach is able to resist even complete subversion, where all algorithms are implemented by the adversary, including the key generation algorithm. Concretely, it is possible to construct various subversion-resilient primitives, including one-way-permutations [27], pseudorandom generators [27], randomness generators [28], semantically secure encryption schemes [28], random oracles [29] and signature schemes [13].

Weaker watchdogs imply stronger security. Generally, the stronger a watchdog is, the better it can prevent subversion attacks. As an extreme example, imagine a "super" watchdog with unlimited running time, which would be able to exhaustively test whether a given cryptographic algorithm perfectly meets the specification on all possible inputs, even if the input space is exponential (as common for the randomness space of a key generation algorithm, or the message and randomness space of an encryption algorithm, for instance). Such a watchdog would be able to detect and reject any subverted implementation that deviates from the specification, and thus trivially imply security in the subversion setting, since no subversion is permitted. However, such an ideal watchdog is too strong to be efficiently constructable in practice.

Russell *et al.* [27] considered various notions of watchdogs:

- An *offline watchdog* performs a one-time check of the supplied implementations prior to deployment;
- An *online watchdog* is executed in parallel to the deployment of the scheme, and able to access the full transcript of cryptographic protocols by monitoring public interactions between users;
- An *omniscient watchdog* is in addition also aware of the entire internal state (*e.g.*, secret keys) of the implementation.

Note that among all of the above, the offline watchdog is the weakest and thus the most desirable that one might want to use for achieving subversion resilience in the watchdog model. In particular, recent efforts have been made to construct subversion-resistant primitives in the offline watchdog model, including encryption schemes [28] and signature schemes [13]. We emphasize that an offline watchdog can not defend against *stateful* subversions, which are out of scope of this work (for more explanations see Section 3).

Our motivation: Subversion-resilience with efficient watchdogs. The offline watchdog is particularly attractive from a practical perspective, as only a "one-time" check is required. However, it turns out that such a one-time check can require very extensive testing in order to achieve some desirable security guarantee in practice. Concretely, the current approach [27] considers a watchdog already successful if it has a *non-negligible* probability of detecting a malicious subversion. To obtain some *concrete* acceptable detection probability, this could require the offline watchdog to do a *polynomial* amount of testing, where in the worst case the running time of the watchdog might even have to exceed the running time of the adversary, which seems not very practical. As pointed out by Russell *et al.* in [28], amplifying a non-negligible detection probability ϵ to an overwhelming detection probability $1 - \delta$ would require $\epsilon^{-1} \log(\delta^{-1})$ repetitions of testing. Also, due to the asymptotic definition, it is unclear how long the watchdog would need to be run *concretely* in practice before it could state with high confidence whether a given component is considered sufficiently secure or not.

In the theory of self-testing/correcting programs by Blum, Luby and Rubinfeld [8], limited testing by the tester/corrector is also considered as a central design goal. Therefore we are interested in constructing subversion-resilient cryptosystems in the watchdog model, such that we can construct extremely efficient watchdog algorithms, that are guaranteed to run significantly faster than an adversary, ideally in *linear* or even in *constant* time. We believe that this is a necessary step towards making Russell *et al.*'s [27] watchdog model applicable in real-world applications. In particular, we are interested in techniques to construct cryptographic schemes that allow for watchdogs with a concretely-bounded running time that only need to perform a very limited number of tests in order to achieve strong security guarantees.

Remark. We stress that we do not intend to show that the watchdog model outperforms other approaches in defending against subversion attacks, but mainly aim at exploring the possibility of achieving security against subversion attacks in this model using watchdogs with minimal running time. In fact, as mentioned above, all the existing models are generally incomparable, due to the adoption of different specific approaches and assumptions.

1.1 Our Results

Motivated by the aforementioned practical concerns, in this work, we design the first random number generator and public key encryption scheme with offline watchdogs that perform only limited testing, but still achieve strong standard security guarantees. In particular, we make the following contributions.

- The main theoretical contribution of our work is a parameterized refinement of Russell *et al.*'s watchdog model. More precisely, we present a generic model to capture the goal of subversion-resilience with a universal offline watchdog and trusted amalgamation for any cryptographic primitive. The model is defined with concrete security parameters so that specific bounds on the runtime of the watchdog are explicitly given. This requires a conceptual modification of the joint security experiment involving the watchdog and the adversary.
- As the first contribution, we then construct an extremely simple randomness generator that is guaranteed to output *uniformly* random bits even if the watchdog only tests the underlying component for a *constant* time. Note that this randomness generator could also be used to design other subversion-resilient cryptographic schemes with runtime-constrained watchdogs.
- Based on this randomness generator as well as an additional trusted XOR operation[3], we design a subversion-resilient key encapsulation mechanism (KEM) using watchdogs running in *linear* time. This KEM then implies a subversion-resilient

[3] Such a trusted operation is also required in the PKE construction by Russell *et al.*. in [28]

public key encryption scheme that has a watchdog with practical running time. The size of public keys and ciphertexts of this scheme are linear in the security parameter.

Below we elaborate on the proposed techniques in more detail.

1.2 Technical Overview

At the core of our proposed PKE scheme is the subversion-resilient KEM using offline watchdogs doing constant-time testing. As mentioned above, such a construction is challenging as we now only rely on watchdogs that do less testing. Below we first present the intuition why this is a non-trivial task and then the idea behind our construction.

The difficulty of recognizing a subverted KEM with a watchdog. Let $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be the specification of a legitimate (*i.e.*, not subverted) KEM. Let $R_{\mathsf{gen}}, R_{\mathsf{enc}}$ be the randomness spaces of Gen and Encaps. Let F be the deterministic function parameterized by a KEM which takes as input randomness $(r, s) \in R_{\mathsf{gen}} \times R_{\mathsf{enc}}$ for Gen and Encaps, respectively, and then computes

$$(pk, sk, C, K) = F_{\mathsf{KEM}}(r, s)$$

with

$$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda; r) \ \ , \ \ (C, K) \leftarrow \mathsf{Encaps}(pk; s).$$

Now let $\widetilde{\mathsf{KEM}} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Encaps}}, \widetilde{\mathsf{Decaps}})$ be a (possibly subverted) implementation of the algorithms of KEM. We observe that if

$$F_{\mathsf{KEM}}(r, s) = F_{\widetilde{\mathsf{KEM}}}(r, s) \tag{1}$$

on all $(r, s) \in R_{\mathsf{gen}} \times R_{\mathsf{enc}}$, then the security of the originally specified scheme KEM implies security of $\widetilde{\mathsf{KEM}}$, if (r, s) are indeed chosen randomly. If Equation (1) holds, then the implementations $\widetilde{\mathsf{Gen}}$ and $\widetilde{\mathsf{Encaps}}$ agree with the specification on all inputs, and these are the only algorithms of $\widetilde{\mathsf{KEM}}$ used in the security experiment. The same holds if Equation (1) holds for all but a *negligible* fraction of all r, s, since the probability that a security experiment chose (r, s) such that Equation (1) does *not* hold would be negligible.

Unfortunately, we are not able to test efficiently whether Equation (1) holds for all but a negligible fraction of all r, s. Let

$$\mathsf{Neq} := \left\{(r, s) : F_{\mathsf{KEM}}(r, s) \neq F_{\widetilde{\mathsf{KEM}}}(r, s)\right\} \tag{2}$$

and

$$\mathsf{Eq} := \left\{(r, s) : F_{\mathsf{KEM}}(r, s) = F_{\widetilde{\mathsf{KEM}}}(r, s)\right\}. \tag{3}$$

That is, Neq contains all "bad" randomness values such that Equation (1) does *not* hold with respect to $(\mathsf{KEM}, \widetilde{\mathsf{KEM}})$. Analogously, Eq contains all "good" randomness values for which Equation (1) does hold. Since Neq and Eq are disjoint sets, we have

$$\mathsf{Neq} \cup \mathsf{Eq} = R_{\mathsf{gen}} \times R_{\mathsf{enc}} \quad \text{and} \quad \mathsf{Neq} \cap \mathsf{Eq} = \emptyset.$$

Note that testing whether $|\mathsf{Neq}|/2^\lambda$ is negligible by repeatedly running F on different inputs takes exponential time. Even if we granted the watchdog a very large running time, by allowing it to evaluate F a polynomial $P(\lambda)$ of times, where $P(\lambda)$ is large, this watchdog could still fail to recognize that $|\mathsf{Neq}|$ is non-negligible with very high probability.

For instance, suppose that $\mathsf{Neq} \subset \{0,1\}^\lambda \times \{0,1\}^\lambda$ were a random subset of size $|\mathsf{Neq}| = |R_{\mathsf{gen}} \times R_{\mathsf{enc}}|/P^2(\lambda)$. Then a watchdog running in time P would fail to detect the subversion only with probability $1 - 1/P(\lambda)$. At the same time, the scheme would be insecure, since we have

$$\frac{|R_{\mathsf{gen}} \times R_{\mathsf{enc}}|/P^2(\lambda)}{|R_{\mathsf{gen}} \times R_{\mathsf{enc}}|} = \frac{1}{P^2(\lambda)}$$

and thus the security experiment would choose "bad" randomness $(r,s) \in \mathsf{Neq}$ with significant probability $\frac{1}{P^2(\lambda)}$.

Our approach to overcoming the technical difficulties. We build a subversion-resilient KEM $\mathsf{KEMSR} = (\mathsf{GenSR}, \mathsf{EncapsSR}, \mathsf{DecapsSR})$ based on a regular $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ in the following way. For ease of exposition, we describe a less efficient scheme here. Our actual scheme can be instantiated much more efficiently, by trading the size of keys and ciphertexts for reasonably increased running time of the watchdog. See Section 5.3.

A key pair $(pk, sk) = ((pk_i)_{i\in[\lambda]}, (sk_i)_{i\in[\lambda]})$ of KEMSR consists of λ many public keys

$$(pk_1, sk_1), \ldots, (pk_\lambda, sk_\lambda) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$$

of KEM. In order to generate an encapsulated key, we run $(C_i, K_i) \leftarrow \mathsf{Encaps}(pk_i)$ for all $i \in [\lambda]$, and return

$$(C, K) := ((C_1, \ldots, C_\lambda), K_1 \oplus \cdots \oplus K_\lambda).$$

Here the $\oplus$ is part of the trusted amalgamation function.

The security proof against subversion in the watchdog model of this construction is based on the following idea. Let $\widetilde{\mathsf{KEM}}$ be a subverted implementation of KEM[4] and let Neq be as in Equation (6). We construct a reduction to the security of the underlying KEM that goes through if the simulated security experiment generates *at least one* pair (pk_i, C_i) using "good" randomness, that is, choosing $(r,s) \xleftarrow{\$} R_{\mathsf{gen}} \times R_{\mathsf{enc}}$ such that $(r,s) \in \mathsf{Eq}$.

Note that even if the underlying KEM is heavily subverted, for instance such that half of all randomness tuples (r,s) are "bad" and we have

$$|\mathsf{Neq}| = \frac{|R_{\mathsf{gen}} \times R_{\mathsf{enc}}|}{2} \tag{4}$$

we would still have

$$\Pr\left[(r,s) \notin \mathsf{Neq} : (r,s) \xleftarrow{\$} R_{\mathsf{gen}} \times R_{\mathsf{enc}}\right] = 1/2.$$

[4] In our model the adversary provides an implementation of each building block instead of an implementation of KEMSR.

Therefore, the probability that the experiment generates at least one pair (pk_i, C_i) using "good" randomness is $1 - 2^{\lambda}$, which is almost certain, up to a negligibly small probability.

If the adversary produces a subverted implementation where $|\mathsf{Neq}|$ is larger than in Equation (4), then of course it becomes less likely that the experiment chooses "good" randomness. However, already for $|\mathsf{Neq}|$ as in Equation (4) we are able to detect the subversion almost certainly, and with a very efficient watchdog. Concretely, a watchdog running F λ times and comparing the output to the specification is able to detect the subversion with overwhelming probability $1-2^{-\lambda}$. This detection probability increases with larger $|\mathsf{Neq}|$.

In order to ease the notation and make our approach clear, the above sketch of our construction uses λ many executions of the KEM procedures as well as a watchdog which tests each algorithm λ many times. As we will see later, in our construction these parameters can be adjusted to allow for tradeoffs between ciphertext (and key) size and runtime of the watchdog.

On the KEM combiner. Note that a similar idea of using a KEM combiner was also used by Giacon *et al.* [21]. However, the work by Giacon *et al.* considers a different setting from our work. In their work, it is assumed that there are different KEMs with different security properties, in a situation where it is unclear which one is the most suitable candidate for establishing security (*e.g.*, since they are based on different hardness assumptions). Instead, we primarily address the problem of using a possibly subverted KEM to realize secure key encapsulation. Our main approach is to use the watchdog doing limited testing to ensure that the subverted KEM is consistent with its official specification at some points, based on which we are able to have desirable security guarantee via amplification.

One may wonder whether we could use combiners for public key encryption directly to obtain a subversion-resilient PKE scheme. The answer is negative. From a syntax level, most of the currently known combiners would be considered as part of the trusted amalgamation in our model. However, since we consider a runtime-constrained watchdog, we can not rule out the possibility that the PKE scheme would, for instance, simply output the message instead of a ciphertext for a specified message m . Such an attack is known as an "input trigger attack" [15] and prevents us from directly using most of the existing combiners/amplifiers in our setting.

Russell *et al.* [28] argued that subverted encryption algorithms can not have direct access to the input message in order to rule out input trigger attacks. Thus, they blind the message with a random coin which is output as part of the ciphertext. We remark that this approach does not work in our setting, since this still requires polynomial time testing by the watchdog in order to obtain an overwhelming detection probability. In our construction, we bypass this type of attacks by directly performing the XOR operation on the message with the output key of our designed subversion-resilient KEM.

Limitations of our approach. Note that while our construction allows for watchdogs with limited runtime, it comes with some limitations. Following Russel *et al.* [28], we proved our results under the assumption that trusted amalgamation is feasible. This means that all components of a cryptographic scheme can be split into multiple smaller parts, which can be tested individually by the watchdog. During the security experiment the amalgamation then assembles the building blocks without adversarial interference. Further, in our model we only account for *stateless* subversions, as we only rely on offline watchdogs with bounded runtime. If *stateful* subversions are allowed, a subverted algorithm could behave honestly until the watchdogs stop testing and then arbitrarily break security. For further discussion on alternative models, we refer to the following section. Furthermore, note that our proposed PKE scheme uses the key produced by our constructed subversion-resilient KEM to (in a trusted manner) XOR the message, thus limiting the message length to be encrypted.

2 Related Work

Since the Snowden revelations in 2013, extensive efforts have explored the feasibility results in the subversion setting, relying on different assumptions regarding trusted components and architectural requirements [6, 5, 2, 4, 15, 27–29, 13, 1, 25, 12, 17, 19].

2.1 Existing Constructions in the Watchdog Model

There have been various constructions in the typical watchdog model by [27]. Based on the split-program methodology [27], several cryptographic schemes were proposed and proved to be subversion-resilient in the complete subversion setting [27–29, 13]. Particularly, in [27], Russell *et al.* constructed subversion-resilient (trapdoor) one way permutations (TDOWP) in the offline watchdog model (assuming fixed public input distributions). The main idea is to use a standard hash function (modeled as a random oracle) to disable the ability of an adversary to embed any potential backdoors in the function. Further, based on this general sanitizing strategy, they built a subversion-resilient signature scheme with online watchdogs, and subversion-resilient pseudorandom generators (PRG) with offline watchdogs. To generically eliminate subliminal channels in randomized algorithms, Russell *et al.* [28] proposed a “double-splitting” strategy where the randomness generation is carried out by mixing the output of two independent components with an immunization function. Based on this they showed how to further immunize each algorithm of an encryption scheme, including symmetric-key encryption and public-key encryption, with offline watchdogs. In [11], Chen *et al.* also discussed how to construct subversion-resilient key encapsulation mechanisms by using this useful strategy. Russell *et al.* also considered how to correct subverted random oracles in [29] and Chow *et al.* [13] further extended their results to construct subversion-resilient signature schemes in the offline watchdog model. In [1], by relying on an additional independent (untamperable) source of public randomness, Ateniese *et al.* proposed a subversion-secure immunizer in the plain model for a broad class of deterministic primitives.

There are some other constructions which also implicitly rely on the (polynomial-testing) watchdog to achieve subversion-resilience. In [6], Bellare, Patterson and Rogaway showed that symmetric encryption producing unique ciphertexts could resist subversion attacks, assuming that all subverted ciphertext are decryptable. This decryptability condition was further relaxed by [15] for considering the possibility of input-trigger attacks. Note that both of them require an omniscient watchdog that needs to access the decryption key for verifying the ciphertext decryptability produced by the supplied implementation of encryption algorithm. Similarly, Ateniese, Magri and Venturi [2] showed that unique signatures are subversion-resilient on the condition that all subverted signatures are valid. They also proved the unforgeability of unique signatures still hold against random message attacks when the verifiability condition is relaxed in a way similar to what considered by [15]. Note that their constructions require an online watchdog as the signature scheme is publicly verifiable.

2.2 Combiner and Amplification

As mentioned earlier, Giacon *et al.* [21] also proposed the XOR combiner for KEMs but in a different setting from ours. Roughly, a combiner is an algorithm that takes as input several instantiations of the same primitive and then combines these into a single scheme, aiming at achieving desirable security on the condition that at least one of the underlying ”building blocks” is secure. There have been several works constructing combiners for different primitives such as KEMs [21], authenticated encryption with associated data (AEAD) [26] and functional encryption [24].

The watchdog model generally relies on trusted amplification to build fully functional implementation from individual “secure enough” components for strong security. In fact, amplification is strongly related to cryptographic combiners. Roughly speaking, given a cryptographic scheme with some “weak” security guarantee, the amplifier can construct a scheme with stronger security guarantees (for example simply by repeated executions). Amplification has been applied to different primitives like functional encryption [23], interactive cryptographic protocols [16] and CCA-secure public key encryption [22].

2.3 Cryptographic Reverse Firewalls

Note that the watchdog model essentially relies on testing, combined with the split-program methodology, to achieve feasibility results in the subversion setting. Alternatively, one may consider other models relying on different assumptions, such as trusted components. In particular, Mironov and Stephens-Davidowitz [25] introduced the notion of *cryptographic reverse firewalls*, which permits quite extensive results [2, 17, 12, 9, 10]. A reverse firewall is an independent trusted on-line party, which could be viewed as a proxy located between a (potentially) subverted machine and the outside world. It is assumed that the reverse firewall has access to a source of trusted randomness and could faithfully re-randomize all incoming/outgoing communication generated by the subverted algorithm, so that subversion-resilience is achieved even if the full implementation has been tampered with by the adversary. It is worth mentioning that the reverse firewall is not assumed to be a trusted party to achieve security in absence of subversion. In particular, it has no access to any secret information, such as secret keys. Note that similar to the online watchdog model, an (active) reverse firewall is able to defend against *stateful* subversion, which is inherently not captured by the offline watchdog model. Furthermore, many constructions using the reverse firewall model, such as [25, 2, 17, 12, 10], require some form of “re-randomizability” of the algorithm outputs provided to an adversary, which limits their applicability to non-rerandomizable primitives.

Two notable exceptions are due to Mironov and Stephens-Davidowitz [25], who proposed a generic approach to convert any protocol into a protocol that is compatible with reverse firewalls, and Bossuat *et al.* [9], where the reverse firewall is in possession of a public key which allows it to sanitize protocol messages without requiring rerandomizability of the underlying primitives.

2.4 Self-Guarding Mechanisms

Another alternative model is the so-called self-guarding mechanism, which was introduced by Fischlin and Mazaheri [19]. This model assumes that there exists an honest initialization phase where the algorithm is not subverted and thus produce a "clean" output. Thus, during this phase, one could gather a collection of samples by executing the honest implementation. After the implementation was tampered with, the output would be sanitized by the honest samples to resist any possible secret exfiltration. The main advantage of this model is that it does not require an active party (such as the reverse firewall or the watchdog) but rather constructs primitives that are "inherently" immune to subversion attacks. A limitation of this approach is the bounded security which depends on the number of samples collected during the good initial phase. Also, such a strategy only considers attacks that wake up at a later point in time, *e.g.*, due to the software update, while the watchdog model considers long-term subversion attacks that might be active during the whole life cycle of the cryptosystem.

3 A Definitional Framework for Subversion-Resilient Cryptography

In this section, we present a variant of the security definitions from [27] which is similar in spirit to theirs but captures our security goals for watchdogs with bounded running time better.

3.1 Notations

Before we present our model, we first introduce the notations used in this work. We will use $x \xleftarrow{\$} X$ to denote sampling x uniformly at random from the set X. Further, let $\mathcal{A}$ be a randomized algorithm. In order to explicitly reference the random coins used by $\mathcal{A}$, we will use $y \leftarrow \mathcal{A}(x; r)$ to denote assigning y to the output of $\mathcal{A}$ running on input x using the random coins r. $[1, n]$ will denote the set of natural numbers from 1 to n, *i.e.* $\{1, 2, \ldots, n-1, n\}$. To denote an implementation (provided by some adversary) of some algorithm $\mathcal{B}$, we will use $\widetilde{\mathcal{B}}$. Finally, we will use $\widehat{\Pi}$ to denote the specification of a scheme.

3.2 A General Security Definition for Cryptographic Schemes

We define a *cryptographic scheme* Π as a tuple of n algorithms

$$\Pi = (\Pi_1, \ldots, \Pi_n).$$

Note that for a specific primitive, n is a fixed number and is usually small. For example, for a public-key encryption scheme, which is typically defined as a tuple of algorithms $(\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$, we have $n = 3$ and

$$(\mathit{\Pi}_1, \mathit{\Pi}_2, \mathit{\Pi}_3) = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt}).$$

Security of $\mathit{\Pi}$ is defined via a *security experiment* $\mathsf{Exp}^{\mathit{\Pi}}_{\mathcal{A}}(1^\lambda)$ which involves $\mathit{\Pi}$ and an adversary $\mathcal{A}$ and outputs a bit $b \in \{0,1\}$. In the sequel, we will focus on security experiments based on the *indistinguishability* of two distributions.[5] Then we can generically define the *advantage function* of $\mathcal{A}$ with respect to $\mathit{\Pi}$ and experiment Exp as

$$\mathsf{Adv}^{\mathit{\Pi}}_{\mathcal{A}}(1^\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathit{\Pi}}_{\mathcal{A}}(1^\lambda) = 1\right] - 1/2\right|.$$

In the concrete security setting, we say the scheme $\mathit{\Pi}$ is (t, ϵ)*-secure*, if $\mathsf{Adv}^{\mathit{\Pi}}_{\mathcal{A}}(1^\lambda) \leq \epsilon$ for all $\mathcal{A}$ running in time at most t.

3.3 Subversion-resilience with an Offline Watchdog

To define security against subversion attacks (adversarial implementations), we follow Russell *et al.*'s approach [27] and consider a setting where *the adversary itself* provides the implementation used in the security experiment of the considered scheme. More precisely, we consider an adversary $\mathcal{A}$ that consists of two parts $(\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ produces a (possibly subverted) *implementation* and a state st

$$(\widetilde{\mathit{\Pi}}_1, \ldots, \widetilde{\mathit{\Pi}}_n, st) \xleftarrow{\$} \mathcal{A}_0(1^\lambda).$$

Then $\mathcal{A}_1(st)$ engages in the security experiment Exp, which uses $\widetilde{\mathit{\Pi}} = (\widetilde{\mathit{\Pi}}_1, \ldots, \widetilde{\mathit{\Pi}}_n)$. Note that in such a strong adversarial setting it is impossible to achieve meaningful security without further assumptions. Therefore, based on the fact that in the real world implementations of algorithms can be tested before deployment in an application, Russell *et al.* [27] introduced an additional party called the *watchdog*. The watchdog aims to detect a possible subversion in the implementation supplied by the adversary. The watchdog WD is aware of an "honest" (not subverted) *specification* of the scheme, denoted by

$$\widehat{\mathit{\Pi}} = (\mathit{\Pi}_1, \ldots, \mathit{\Pi}_n),$$

and has oracle access to the implementation $\widetilde{\mathit{\Pi}} = (\widetilde{\mathit{\Pi}}_1, \ldots, \widetilde{\mathit{\Pi}}_n)$ produced by $\mathcal{A}_0$. The adversary $\mathcal{A}$ is only considered successful if it breaks the security of the scheme and the subverted implementation *evades detection* by the watchdog. Hence, we consider the *subversion-resilience* security experiment $\overline{\mathsf{SR}}^{\mathcal{A},\mathsf{WD}}_{\mathsf{Exp},\mathit{\Pi}}(1^\lambda)$ from Figure 1 (a), which involves a subversion adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, a watchdog WD, a protocol specification $\widehat{\mathit{\Pi}}$ and an underlying standard (indistinguishability-based) experiment Exp .

[5] Even though it is straightforward to extend our description to the general case capturing both classical cases of "indistinguishability" and "search problems", we refrain from introducing additional notation to achieve this. We will only consider indistinguishability in this paper.

```
$\overline{\mathsf{SR}}^{\mathcal{A},\mathsf{WD}}_{\mathsf{Exp},\widehat{\Pi}}(1^\lambda)$
$(\widetilde{\Pi}, st) \leftarrow \mathcal{A}_0(1^\lambda)$
If $\mathsf{WD}^{\widetilde{\Pi}}(1^\lambda)$ then
    $b \xleftarrow{\$} \{0,1\}$
    Return $b$
Return $\mathsf{Exp}^{\mathcal{A}_1(st)}_{\widetilde{\Pi}}(1^\lambda)$
```

(a) Without trusted amalgamation and with $\widehat{\Pi} = (\Pi_1, \ldots, \Pi_n)$

```
$\mathsf{SR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{Exp},\widehat{\Pi}}(1^\lambda)$
$(\widetilde{\Pi}, st) \leftarrow \mathcal{A}_0(1^\lambda)$
If $\mathsf{WD}^{\widetilde{\Pi}}(1^\lambda)$ then
    $b \xleftarrow{\$} \{0,1\}$
    Return $b$
Return $\mathsf{Exp}^{\mathcal{A}_1(st)}_{\mathsf{Am}(\widetilde{\Pi})}(1^\lambda)$
```

(b) With trusted amalgamation and $\widehat{\Pi} = (\mathsf{Am}, \Pi_1, \ldots, \Pi_n)$

Fig. 1: Security experiments with watchdog and subverted implementation.

At the beginning of the experiment, adversary $\mathcal{A}_0$ produces a (subverted) implementation $\widetilde{\Pi}$ and a state st. The watchdog is provided oracle access to $\widetilde{\Pi}$. If WD outputs `true`, which means that the watchdog has detected a subverted implementation, the experiment outputs a random bit. This implies that $\mathcal{A}$ has zero advantage, if it outputs an implementation that WD recognizes as subverted. If $\mathsf{WD}^{\widetilde{\Pi}} = \texttt{false}$, then the security experiment Exp is executed, using the *adversarially-provided* implementation $\widetilde{\Pi}$ of $\widehat{\Pi}$ and with an adversary $\mathcal{A}_1(st)$ that may depend on the state produced by $\mathcal{A}_0$.

In order to avoid trivial watchdogs that always output `true`, such that any scheme would be provably secure, we require that the watchdog is "correct" in the sense that it *always* outputs `false` when provided with oracle access to the actual protocol. Formally:

Definition 1. *We say that* WD *is a* correct *watchdog for protocol specification $\widehat{\Pi}$, if*

$$\Pr\left[\mathsf{WD}^{\widehat{\Pi}} = \texttt{false}\right] = 1.$$

All watchdogs in this work will trivially fulfill this property. As one would intuitively expect, our watchdogs simply compare the output of the oracle with the expected results of the specification. Thus, our watchdogs will never reject the protocol specification.

Note that in the above security experiment (Figure 1), WD verifies $\widetilde{\Pi}$ prior to the experiment Exp. That is, our definition only considers offline watchdogs that simply check the supplied implementations by the adversary with no access to the full transcript of the experiment Exp. As discussed in [27, 28], such a watchdog is preferable over online watchdogs in the sense that it only carries out a one-time check on the implementation and does not require constant monitoring of all communication. We also remark that in this work we will only consider a *universal* watchdog, which means it is quantified before the adversary in the security definition. Thus, for a secure scheme there exists a *single* watchdog that defends against all considered adversaries.

Stateless Subversion. We also remark here that we only consider *stateless* subversion in this work. That is, the subverted implementation does not hold any state between different executions. Note that we are mainly interested in offline watchdogs which run in bounded time for testing. A trivial attack to evade such a detection is the so-called *time bomb* attack which only becomes active when the underlying algorithm is at some specific state. Specifically, the implementations would behave honestly when they are under testing by the offline watchdog, and at a later point in time the malicious behavior would be triggered to wake up. It is clear that such a tricky attack is impossible to be detected by our considered (*i.e.* bounded running time) watchdog. In fact, to prevent such an attack in hardware tokens, some previous work requires a semi-online watchdog to perform testing regularly [18]. Therefore, we insist that stateful subversion is not captured by our considered model. Another approach to consider stateful subversion are reverse firewalls, as discussed in Section 2.3.

3.4 The Split-Program Model and Trusted Amalgamation

The above offline watchdog model is meaningful and was used to establish security for some specific cryptographic primitives [27]. However, it turns out that it is still not generic enough to achieve subversion-resilience for many other primitives. Particularly, it is known that if the user makes only black-box use of the subverted implementation of randomized algorithms, it is hopeless to eliminate a steganographic channel built on the output of algorithms [28]. Therefore a non-black-box model is required for general feasibility results.

Motivated by the above, Russell *et al.* [28] proposed the *split-program model*, where the specification of each algorithm is split into a constant number of components. Precisely, a scheme $\mathit{\Pi}$ is still represented by a tuple of algorithms $(\mathit{\Pi}_1, \ldots, \mathit{\Pi}_n)$, but n may be larger than the actual number of algorithms of a protocol. The actual algorithms are then “amalgamated” by combining these underlying building blocks in a trusted way that cannot be influenced by the adversary. Using such a somewhat relaxed model, Russell *et al.* [28] showed how to generically design stego-free specifications for randomized algorithms which play a crucial role in constructing subversion-resilient encryption schemes (in their offline watchdog model). Basically, they further split the specification of any probabilistic algorithm into two parts: the randomized component and the deterministic component, that are tested individually by the watchdog. The randomized component generates random coins which (perhaps together with other inputs drawn from public distributions) are taken as input by the deterministic component to generate the output of the composed algorithm.

Note that for a meaningful construction of a subversion-resilient cryptosystem, the amalgamation should be *as-simple-as-possible*, such that most of the complexity of the involved algorithms is contained in the algorithms $\mathit{\Pi} = (\mathit{\Pi}_1, \ldots, \mathit{\Pi}_n)$ that may be subject to subversion. To make this approach more precise, we explicitly define an *amalgamation function* Am and include it in the specification of the scheme. For instance, for a public-key encryption scheme we would have the specification

$$\widehat{\mathit{\Pi}} = (\mathsf{Am}, \mathit{\Pi}) = (\mathsf{Am}, (\mathit{\Pi}_1, \ldots, \mathit{\Pi}_n))$$

for which

$$\mathsf{Am}(\mathit{\Pi}_1, \ldots, \mathit{\Pi}_n) = (\mathsf{Gen}, \mathsf{Encrypt}, \mathsf{Decrypt})$$

holds.

As depicted in Figure 1 (b), the subversion-resilience security experiment with trusted amalgamation proceeds exactly as the basic experiment described above, except that the watchdog has access to all procedures

$$(\widetilde{\Pi}_1, \ldots, \widetilde{\Pi}_n, st) \xleftarrow{\$} \mathcal{A}_0(1^\lambda)$$

produced by $\mathcal{A}_0$ *individually*. The security experiment is then executed with the amalgamated primitive

$$\mathsf{Am}(\widetilde{\Pi}_1, \ldots, \widetilde{\Pi}_n).$$

Following our example for public key encryption, this would correspond to

$$\mathsf{Am}(\widetilde{\Pi}_1, \ldots, \widetilde{\Pi}_n) = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Encrypt}}, \widetilde{\mathsf{Decrypt}}).$$

With our security model set up, we can now define the advantage of a subversion adversary in the split program model with offline watchdog as

$$\mathsf{AdvSR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{Exp},\widehat{\Pi}}(1^\lambda) := \left| \Pr\left[\mathsf{SR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{Exp},\widehat{\Pi}}(1^\lambda)\right] - 1/2 \right|.$$

This allows to present our formal definition of subversion-resilience.

Definition 2. *A specification of a cryptographic protocol* $\widehat{\Pi} = (\mathsf{Am}, \Pi)$ *is* $(t_{\mathsf{WD}}, t_{\mathcal{A}}, \varepsilon)$-subversion-resilient in the offline watchdog model with trusted amalgamation*, if one can efficiently construct a* correct *watchdog algorithm* WD *running in time at most* t_{WD} *such that for any adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *running in time at most* $t_{\mathcal{A}}$ *it holds that*

$$\mathsf{AdvSR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{Exp},\widehat{\Pi}}(1^\lambda) \leq \varepsilon$$

using the experiment shown in Figure 1 (b).

Here it might seem counterintuitive to define security with regards to the *specification*, if the underlying security experiment is executed with the *subverted* implementation. Our definition can be interpreted in the following way. While the security experiment is executed with the subverted implementation provided by the adversary, the watchdog (which is also executed in the experiment) tests the implementation with respect to that specification, and the adversary outputs subverted algorithms that syntactically follow the specification of the trusted amalgamation. Following Russell *et al.* [28], we split randomized algorithms into a probabilistic part (the randomness generation) and a deterministic part, where all parts can be tested individually. The trusted amalgamation then feeds the generated randomness into the deterministic algorithms. Since we are considering universal *offline* watchdogs, we have to make the assumption that a subverted implementation of a deterministic primitive also is deterministic. Otherwise, a subverted implementation could probabilistically deviate from the specification with some probability, where this probability could possibly be chosen depending on the watchdog and its bounded running time, so that the watchdog might fail to detect the subversion. (Note that is very closely connected to the reason why offline watchdogs cannot consider *stateful* subversion.)

3.5 Comparisons with Previous Watchdog Models

The security model presented in this section is a refinement of the security models from [28, 27]. We follow the “concrete security” approach instead of considering asymptotic definitions and assume the specification could be divided into an arbitrary number of components. Similarly to [28], we consider a single security experiment that can be separated in a ”detection phase” and a ”surveillance phase”. Note that in [28] two advantage functions are defined: one for the watchdog and another for the adversary. Security then holds if either the detection advantage of the watchdog is non-negligible or the adversaries’ advantage is negligible. We change the model in the regard that we enforce that the adversary “loses” the security experiment in case the watchdog detects subversion, by outputting a random bit instead of executing the experiment. In this way, we simplify the security definition by using a single advantage function. We remark that our refined model is inherently incomparable with previous models [28, 27] but has notational advantages for our goal of achieving subversion-resilience with efficient watchdogs.

4 Subversion-Resilient Randomness Generators

After presenting our security model we will now show how to generate randomness in a way that allows a watchdog to guarantee in *constant* time (also independent from an adversary’s runtime), that the outputs of our construction are *uniformly random* (*i.e.*, not just indistinguishable from random, but truly random). This randomness generator will then later be used to generate the random coins for our subversion-resilient KEM and PKE scheme. Additionally, this construction is of independent interest as it is a general and efficient tool to provide uniformly random coins to any cryptographic primitives in our model.

A randomness generator is a randomized algorithm which on input of a security parameter outputs some strings. We consider a randomness generator secure if its outputs are indistinguishable from uniformly random strings.

Definition 3. *We say that a randomness generator* RG *is* (t, ε)*-indistinguishable if for any adversary* $\mathcal{A}$ *running in time* t *it holds that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{RG}} = |\Pr[\mathsf{RGIND}_{\mathsf{RG}}^{\mathcal{A}}(1^\lambda) - 1/2]| \leq \varepsilon$$

with $\mathsf{RGIND}_{\mathsf{RG}}^{\mathcal{A}}(1^\lambda)$ *displayed in Figure 2 (a).*

Following Definition 2, we say that a randomness generator is *subversion-resilient under trusted amalgamation* if the randomness generator produces outputs that are indistinguishable from random, even in a security experiment which uses a trusted amalgamation of a subverted implementation.

Definition 4. *We say the specification of a randomness generator* $\widehat{\mathsf{RGSR}} = (\mathsf{Am}_{\mathsf{RG}}, \mathsf{RGSR})$ *is* $(t_{\mathsf{WD}}, t_{\mathcal{A}}, \varepsilon)$-subversion-resilient in the offline watchdog model with trusted amalgamation*, if one can efficiently construct a* correct *watchdog* WD *running in time*

```
$\mathsf{RGIND}^{\mathcal{A}}_{\mathsf{RG}}(1^\lambda)$
$b \overset{\$}{\leftarrow} \{0,1\}$
If $b == 0$ then
    return $\mathcal{A}^{\mathcal{O}}(1^\lambda) == b$
If $b == 1$ then
    return $\mathcal{A}^{\mathsf{RG}}(1^\lambda) == b$
```

(a) Experiment for a randomness generator RG. Oracle $\mathcal{O}$ returns uniformly random strings.

```
$\mathsf{SR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{RGIND},\widetilde{\mathsf{RGSR}}}(1^\lambda)$
$(\widetilde{\mathsf{RGSR}}, st) \leftarrow \mathcal{A}_0(1^\lambda)$
If $\mathsf{WD}^{\widetilde{\mathsf{RGSR}}}(1^\lambda)$ then
    $b \overset{\$}{\leftarrow} \{0,1\}$
    Return $b$
Return $\mathsf{RGIND}^{\mathcal{A}_1(st)}_{\mathsf{Am}(\widetilde{\mathsf{RGSR}})}(1^\lambda)$
```

(b) Subversion-resilience experiment for randomness generators.

Fig. 2: Security experiments with watchdog and subverted implementation.

at most t_{WD}, such that for any adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ running in time at most $t_{\mathcal{A}}$ it holds that:

$$\mathsf{AdvSR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{RGIND},\widetilde{\mathsf{RGSR}}}(1^\lambda) \leq \varepsilon$$

with the used experiments shown in Figure 2 (a) and (b).

Known impossibilities. Russell *et al.* [28] showed that it is impossible to immunize a *single* randomness generator against subversion with an immunizing function. Essentially, they adopt the approach of subverting algorithms from [6, 5] to randomness generators, showing that one can easily introduce a bias into a single source via rejection sampling. This bias can then be maintained by a subverted immunizing function. This bias may furthermore be "hidden", in the sense that detecting it requires knowledge of a secret key only known to the adversary to compute some predicate, such that a watchdog would not be able to efficiently detect it, while a subverting adversary may easily distinguish the subverted RG from random.

In order to overcome this general impossibility, Russel *et al.* [28] introduce the *"double splitting"* approach. Here, two RNGs are run independently in parallel. The outputs of these two RNGs are then fed into an immunization function, which may also be subverted. Russel *et al.* showed that if the immunization function is modeled as a random oracle, then this yields a randomness generator whose output is indistinguishable from outputs of a non-subverted randomness generator, even for the subverting adversary. They provide a standard-model construction of a randomness generator that outputs a single bit and a watchdog which tests whether one bit appears significantly more often than the other. Using the Chernoff bound, they argue that the watchdog will notice a bias after gathering enough samples. The randomness generator would then be run n times independently to obtain a n bit output.

We describe a new construction, which applies the "two independent RNG" approach of [28] in a different way. Our construction is extremely simple and efficient to test, yet provides *perfect* random bits and does not require a random oracle.

```
$\mathsf{Am}_{\mathsf{RG}}(1^\lambda, \mathsf{RG}, \mathsf{VN})$
$b := \perp$
**While** $b = \perp$ **do**
    $b_0 \leftarrow \mathsf{RG}(1^\lambda);\ b_1 \leftarrow \mathsf{RG}(1^\lambda);\ b := \mathsf{VN}(b_0, b_1)$
**Return** $b$
```

Fig. 3: The trusted amalgamation function $\mathsf{Am}_{\mathsf{RG}}$ for a randomness generator RG and a von Neumann extractor VN.

4.1 Construction

The specification $\widehat{\mathsf{RGSR}} = (\mathsf{Am}, \mathsf{RG}, \mathsf{VN})$ of our randomness generator consists of the following building blocks:

- A probabilistic algorithm RG that on input 1^λ outputs a bit $\mathsf{RG}(1^\lambda) \in \{0,1\}$.
- A simple binary and deterministic immunization function $\mathsf{VN} : \{0,1\} \times \{0,1\} \to \{0,1\}$, which is defined as follows:

$$\mathsf{VN}(b_0, b_1) := \begin{cases} 0 \text{ if } b_0 < b_1, \\ 1 \text{ if } b_0 > b_1, \\ \perp \text{ else.} \end{cases}$$

  Note that this function is the classical von Neumann extractor [30].

The von Neumann extractor takes as input two bits with some (arbitrary and unknown but fixed) bias and outputs a uniformly random bit, as long as the two input bits are distinct from each other.

Using these two building blocks, we construct an algorithm that on input 1^λ outputs a single bit using trusted amalgamation. The amalgamation is essentially a very simple while-loop, given in Figure 3. It can be easily generalized to output n bits by calling it n times.

The amalgamation function $\mathsf{Am}_{\mathsf{RG}}$ is extremely simple, it runs RG twice independently and applies VN to the output. This is repeated in a while-loop, until the output of VN is not the error symbol $\perp$, but a bit $b \in \{0,1\}$.

Correctness. Since we have $\mathsf{VN}(b_0, b_1) = \perp \iff b_0 = b_1$, the correctness of this algorithm depends on the probability that the two executions of RG computing b_0 and b_1 in the while-loop yield $b_0 \neq b_1$. Let $p := \Pr\left[\mathsf{RG}(1^\lambda) = 1\right]$, then we have

$$\Pr[b_0 \neq b_1] = 2p(1-p).$$

Hence, it takes an expected $(p(1-p))^{-1}$ executions of RG and $(2p(1-p))^{-1}$ executions of VN to generate an output bit. For instance, if RG is truly random, then we would have 4 expected executions of RG and 2 of VN. Even if RG is a rather bad random number generator, say with $p = 1/4$, then one would expect about $5 + 1/3$ executions of RG and $2 + 2/3$ of VN.

4.2 Security

The proof that $\widehat{\mathsf{RGSR}}$ is a subversion-resilient randomness generator uses the fact that the adversary provides a *single* implementation $\widetilde{\mathsf{RG}}$ which is then queried twice by the trusted amalgamation. Therefore, the two bits b_0, b_1 are computed *independently* in every while loop and are identically distributed. The watchdog only has to test the implementation of the von Neumann extractor on all four possible inputs. It is not necessary to test the implementation of the randomness generator at all to achieve security.

Theorem 1. *The specification* $\widehat{\mathsf{RGSR}}$ *as defined above is* $(\mathcal{O}(1), t_{\mathcal{A}}, 0)$*-subversion-resilient in the offline watchdog model with trusted amalgamation.*

Note that the theorem asserts that we can construct a *constant-time* watchdog. It will be independent of the runtime of the adversary or any bias possibly embedded in the subverted implementation of RG.

Proof. The only component that we test is the implementation of $\mathsf{VN} : \{0,1\} \times \{0,1\} \to \{0,1\}$, which is a very simple function with only four possible inputs. Therefore it can be checked on all possible inputs in constant time. The watchdog WD runs a given implementation $\widetilde{\mathsf{VN}}$ on all four possible inputs and checks the correctness of the output with the specification.

In case $\widetilde{\mathsf{VN}}$ deviates from the specification on any input, the watchdog will detect this with probability 1. Thus, this would immediately lead to an advantage of 0 for the adversary.

Provided that $\widetilde{\mathsf{VN}}$ implements VN correctly and using the fact that the two bits $b_0, b_1 \leftarrow \widetilde{\mathsf{RG}}(1^\lambda)$ are computed *independently* in every while loop, we obtain that when $\widetilde{\mathsf{RG}}$ outputs a bit b then we have

$$\begin{aligned}\Pr[b=0] = \Pr[b_0 = 0 \wedge b_1 = 1] &= \Pr[b_0 = 1 \wedge b_1 = 0] \\ &= \Pr[b = 1],\end{aligned}$$

and thus $\Pr[b=0] = \Pr[b=1] = 1/2$, again leading to an advantage of 0 for the adversary. □

4.3 Discussions

The main advantage of our proposed RG is that it achieves perfect security using a *constant*-time watchdog. Note that Russell *et al.* [28] also described several alternative approaches to purify randomness in the standard model. Below we provide more discussion about their approaches. It is worth mentioning that in [1], Ateniese *et al.* proposed a different approach to eliminating the requirement of random oracles by essentially relying on an additional independent and untamperable source of public randomness.

Simple multi-splitting. The first approach proposed in [28] is simple multi-splitting, which means that n copies of RG (each outputting a single bit) are run and all outputs are concatenated and output. The main problem with this approach is that RG has to be tested very many times, otherwise the watchdog is unable to notice a small, but non-negligible bias.

More efficient construction using randomness extractors. The second approach is to use randomness extractors, but in a totally different way. Precisely, it is observed that a watchdog making $\mathcal{O}(n^{2c})$ queries can verify that the output of each RG has at least $c \log n$ bits of entropy (for some constant c). Thus, Russell *et al.* [28] proposed to run RG for $\log n$ times to obtain a random string of length $\log n$, which is then used as a seed for a randomness extractor. This extractor can then be used to obtain more random bits from RG, which afterward can be expanded with a pseudorandom generator (PRG). Note that in this case a watchdog would not only have to check RG for entropy but also recompute all calculations of the extractor and the PRG to check if these components follow their specifications.

A disadvantage of our construction of RG is that we do not have a strict upper bound on its running time, but only expected bounds. We consider this a minor disadvantage, though, since lack of functionality will serve as an automated way to detect subversion that introduces a too heavy bias in RG. The efficiency can be improved if one is willing to do a little bit more testing. For instance, the watchdog could also test RG and check for a too heavy bias p that would significantly harm performance. Note however that even a particularly bad underlying randomness generator RG, which always outputs a constant 0, for instance, would only harm correctness, but not the security of RG.

5 Subversion-Resilient Key Encapsulation Mechanisms

In this chapter, we will construct a (t, ε)-indistinguishable key encapsulation mechanism that is subversion-resilient even if *all* the algorithms are subject to subversion, provided that we have a very simple trusted amalgamation function. Our construction achieves subversion-resilience with a watchdog whose running time is only *linear* in the security parameter, while also allowing for tradeoffs between ciphertext size and the watchdog's runtime.

5.1 Key Encapsulation Mechanisms and Security Definitions

We define *subversion-resilient* KEMs by adapting Definition 2 to KEMs.

Definition 5. *We say that the specification* $\widehat{\mathsf{KEMSR}} = (\mathsf{Am}_{\mathsf{KEM}}, \mathsf{KEMSR})$ *of a key encapsulation mechanism is* $(t_{\mathsf{WD}}, t_{\mathcal{A}}, \varepsilon)$ subversion-resilient in the offline watchdog model with trusted amalgamation*, if one can efficiently construct a correct watchdog* WD *running in time at most* t_{WD} *such that for any adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *running in time at most* $t_{\mathcal{A}}$ *it holds that:*

$$\mathsf{AdvSR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{KEMIND},\widehat{\mathsf{KEMSR}}}(1^\lambda) \leq \varepsilon$$

with the used experiments shown in Figure 4 (a) and (b).

Key encapsulation mechanisms are techniques to securely transport symmetric cryptographic key material using public key cryptography. We will use the following standard definitions for key encapsulation mechanisms and their security.

Definition 6. *A* key encapsulation mechanism *(or KEM) consists of three algorithms* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *with the following syntax.*

- $\mathsf{Gen}(1^\lambda)$*: The randomized key generation algorithm takes as input a security parameter* $\lambda \in \mathbb{N}$ *and outputs a key pair* (sk, pk).
- $\mathsf{Encaps}(pk)$*: The randomized encryption algorithm takes as input a public key* pk. *It outputs a key* $K \in \mathcal{KS}$*, where* $\mathcal{KS}$ *is called the* key space *defined by* pk *(either implicitly or explicitly), and a ciphertext* C.
- $\mathsf{Decaps}(sk, C)$ *: The deterministic decapsulation algorithm takes a secret key* sk *and a ciphertext* C. *It outputs a key* $K \in \mathcal{KS}$ *or a distinguished error symbol* $\perp$.

Definition 7. *We say that* $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ *is* $(t_\mathcal{A}, \varepsilon)$-indistinguishable *if for any adversary* $\mathcal{A}$ *running in time at most* $t_\mathcal{A}$ *it holds that*

$$\mathsf{Adv}_\mathcal{A}^{\mathsf{KEM}}(1^\lambda) := |\Pr[\mathsf{KEMIND}_{\mathsf{KEM}}^{\mathcal{A}}(1^\lambda)] - 1/2| \leq \varepsilon$$

with KEMIND *as defined in Figure 4 (a).*

```
$\mathsf{KEMIND}_{\mathsf{KEM}}^{\mathcal{A}}(1^\lambda)$
$(sk, pk) \leftarrow \mathsf{Gen}(1^\lambda)$
$K_0 \xleftarrow{\$} \mathcal{KS}$
$(C^*, K_1) \leftarrow \mathsf{Encaps}(pk)$
$b \xleftarrow{\$} \{0,1\}$
$b_\mathcal{A} \leftarrow \mathcal{A}(1^\lambda, pk, K_b, C^*)$
If $b_\mathcal{A} == b$ then return $1$
Else return $0$
```

(a) Indistinguishability experiment for key encapsulation mechanisms.

```
$\mathsf{SR}_{\mathsf{KEMIND},\widetilde{\mathsf{KEMSR}}}^{\mathcal{A},\mathsf{WD}}(1^\lambda)$
$(\widetilde{\mathsf{KEMSR}}, st) \leftarrow \mathcal{A}_0(1^\lambda)$
If $\mathsf{WD}^{\widetilde{\mathsf{KEMSR}}}(1^\lambda)$ then
    $b \xleftarrow{\$} \{0,1\}$
    Return $b$
Return $\mathsf{KEMIND}_{\mathsf{Am}(\widetilde{\mathsf{KEMSR}})}^{\mathcal{A}_1(st)}(1^\lambda)$
```

(b) Subversion-resilience experiment for key encapsulation mechanisms.

Fig. 4: Security experiments with watchdog and subverted implementation.

5.2 Our Proposed KEM

Idea overview. Before we present the technical details of our construction, let us illustrate our approach. In order to obtain random coins for our scheme, we will use the subversion-resilient randomness generator from the previous chapter. Overall, we will use n instantiations of a KEM in parallel, where n is a parameter that can be chosen appropriately depending on the application. This means, we run the key generation algorithm n times in parallel to obtain n key pairs. To encapsulate a key, we will also run the Encaps algorithm n times in parallel, each time using a public key that was previously generated. This gives us n ciphertext/key pairs. While all ciphertexts are just output as the final ciphertext, the amalgamation executes an XOR function on all keys. As we will see later in the security analysis, as long as one public key and the ciphertext under that public key were executed honestly, the resulting key pair will be indistinguishable from random.

Construction. With these definitions in place, we are now ready to describe our construction. Let $\widehat{\mathsf{RGSR}} = (\mathsf{Am}_{\mathsf{RG}}, \mathsf{RGSR})$ be the specification of a subversion-resilient randomness generator. Further, let $n > 0$ be an arbitrary constant which allows us to adjust the construction. Since we focus on the key encapsulation mechanism in this section, we will use

$$\mathsf{RG}(1^\lambda) := \mathsf{Am}_{\mathsf{RG}}(1^\lambda, \mathsf{RGSR})$$

to simplify notation. Let $(\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a key encapsulation mechanism. From these building blocks we define a specification of a subversion-resilient key encapsulation mechanism

$$\widehat{\mathsf{KEMSR}} = (\mathsf{Am}_{\mathsf{KEM}}, \mathsf{KEMSR}) = (\mathsf{Am}_{\mathsf{KEM}}, (\mathsf{RGSR}, \mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps}))$$

where the trusted amalgamation $\mathsf{Am}_{\mathsf{KEM}}$ defines algorithms $(\mathsf{GenSR}, \mathsf{EncapsSR}, \mathsf{DecapsSR})$ as follows.

- $\mathsf{GenSR}(1^\lambda)$: Compute $r_i \leftarrow \mathsf{RG}(1^\lambda)$ and $(sk_i, pk_i) \leftarrow \mathsf{Gen}(1^\lambda; r_i)$ for all $i \in [n]$ and output
$$pk := (pk_i)_{i\in[n]} \quad \text{and} \quad sk := (sk_i)_{i\in[n]}.$$
See Figure 5 for an illustration.
- $\mathsf{EncapsSR}(pk)$: On input $pk = (pk_1, \ldots, pk_n)$ compute $r_i \leftarrow \mathsf{RG}(1^\lambda)$ and $(C_i, K_i) \leftarrow \mathsf{Encaps}(pk_i; r_i)$ for all $i \in [n]$ and output
$$C := (C_1, \ldots, C_n) \quad \text{and} \quad K := K_1 \oplus \cdots \oplus K_n.$$
See Figure 6 for an illustration.
- $\mathsf{DecapsSR}(C, sk)$: On input $sk = (sk_1, \ldots, sk_n)$ and $C = (C_1, \ldots, C_n)$ compute $K_i = \mathsf{Decaps}(sk_i, C_i)$ for all $i \in [n]$. If there exists $i \in [n]$ such that $K_i = \perp$, then output $\perp$. Otherwise output
$$K = K_1 \oplus \cdots \oplus K_n.$$

The trusted amalgamation function $\mathsf{Am}_{\mathsf{KEM}}$ essentially consists of simple loops with n independent iterations of calls to the underlying RG and KEM procedures, plus a simple $\oplus$ function. Note that a trusted $\oplus$ was also used in [27] in order to handle large message spaces for public key encryption.

Security analysis.

Theorem 2. *Let* KEM *be a* $(t_{\mathcal{A}}, \varepsilon)$ *indistinguishable key encapsulation mechanism and* $\widehat{\mathsf{RGSR}}$ *be the specification of a* $(\mathcal{O}(1), t_{\mathcal{B}}, 0)$ *subversion-resilient randomness generator. Then* $\widehat{\mathsf{KEMSR}}$ *as defined above with parameters* $n, n_{\mathsf{WD}} > 0 \in \mathbb{N}$ *is* $(t_{\mathsf{WD}}, t'_{\mathcal{A}}, \varepsilon')$ *subversion-resilient in the offline watchdog model with trusted amalgamation with*

$$t_{\mathsf{WD}} \in \mathcal{O}(n_{\mathsf{WD}}), \qquad t'_{\mathcal{A}} \in \mathcal{O}(t_{\mathcal{A}} + t_{\mathcal{B}} + n),$$

$$\varepsilon' \leq \varepsilon + \left(\frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^{n_{\mathsf{WD}}} \cdot \left(1 - \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^{n}.$$

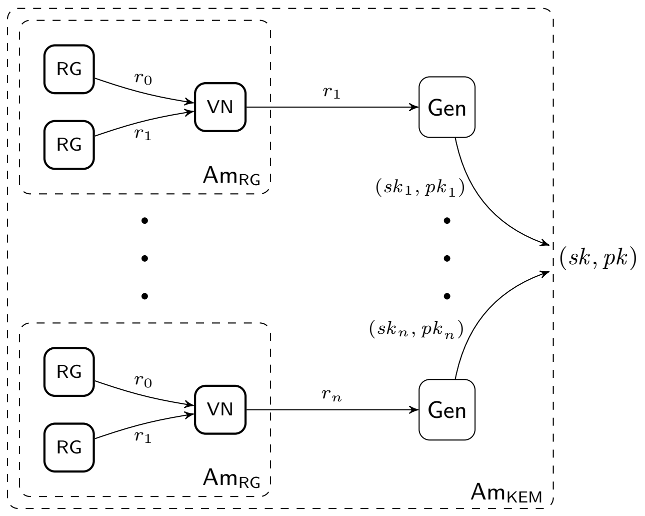

Fig. 5: Subversion-resilient KEM: Key generation algorithm.

Proof. The following notation and helper functions will be useful for the proof. Let $R_{\mathsf{gen}}, R_{\mathsf{enc}}$ denote the randomness space of the algorithms Gen and Encaps, respectively. Let F_{KEM} be the deterministic function parameterized by a key encapsulation mechanism $\mathsf{KEM} = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ which takes as input randomness $(r, s) \in R_{\mathsf{gen}} \times R_{\mathsf{enc}}$ for Gen and Encaps, respectively, and then computes

$$(pk, sk, C, K) = F_{\mathsf{KEM}}(r, s) \tag{5}$$

with

$$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda; r) \quad \text{and} \quad (C, K) \leftarrow \mathsf{Encaps}(pk; s).$$

For KEM (which is part of the specification $\widehat{\mathsf{KEMSR}}$) and a corresponding implementation $\widetilde{\mathsf{KEM}}$ we can now define sets Neq and Eq as

$$\mathsf{Neq} := \left\{(r, s) : F_{\mathsf{KEM}}(r, s) \neq F_{\widetilde{\mathsf{KEM}}}(r, s)\right\} \tag{6}$$

and

$$\mathsf{Eq} := \left\{(r, s) : F_{\mathsf{KEM}}(r, s) = F_{\widetilde{\mathsf{KEM}}}(r, s)\right\}. \tag{7}$$

Hence, set Neq contains all "bad" randomness values where the implementation deviates from the specification, and Eq contains all "good" randomness values where specification and implementation match. Since Neq and Eq are disjoint sets, we have

$$\mathsf{Neq} \cup \mathsf{Eq} = R_{\mathsf{gen}} \times R_{\mathsf{enc}} \quad \text{and} \quad \mathsf{Neq} \cap \mathsf{Eq} = \emptyset.$$

Watchdog construction. We construct a universal offline watchdog WD which proceeds as follows.

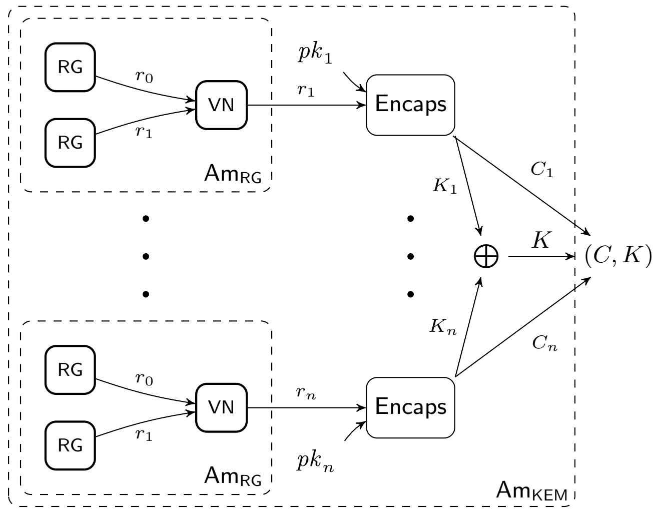

Fig. 6: Subversion-resilient KEM: Encapsulation algorithm.

1. First WD runs the watchdog for $\widehat{\mathsf{RGSR}}$ as a subroutine. If this algorithm outputs `true`, then WD outputs `true`. Otherwise, WD proceeds.
2. Then, for $i \in [1, n_{\mathsf{WD}}]$, WD picks $(r_i, s_i) \xleftarrow{\$} R_{\mathsf{gen}} \times R_{\mathsf{enc}}$ uniformly at random and checks whether

$$F_{\mathsf{KEM}}(r_i, s_i) = F_{\widetilde{\mathsf{KEM}}}(r_i, s_i)$$

holds where F is as defined in Equation (5). If any of these checks fails, then the watchdog outputs `true`. Otherwise, it outputs `false`.

Note that the above watchdog performs only a constant number of queries to check RG plus n_{WD} many evaluations of each Gen and Encaps.

Security analysis. In order to analyze the security of our scheme with respect to this watchdog, consider the following sequence of games.

Game 0. This Game is the $\mathsf{SR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{KEMIND},\widehat{\mathsf{KEMSR}}}(1^\lambda)$ experiment.

Game 1. This game is identical to Game 0, except that all invocations of $\widetilde{\mathsf{RG}}$ are replaced with uniformly random bits.

Since WD runs the watchdog for $\widehat{\mathsf{RGSR}}$ as a subroutine it outputs `true` only if the watchdog for $\widehat{\mathsf{RGSR}}$ does. By the $(\mathcal{O}(1), t_{\mathcal{B}}, 0)$-subversion-resilience of RG this game is therefore perfectly indistinguishable from Game 0.

Game 2. This game is identical to Game 1, except that the execution of game $\mathsf{KEMIND}^{\mathcal{A}}_{\mathsf{KEM}}$ is changed in the following way. After computing

$$(sk, pk) = ((pk_i)_{i\in[n]}, (sk_i)_{i\in[n]})$$

for $(sk_i, pk_i) = \widetilde{\mathsf{Gen}}(1^\lambda; r_i)$, and then

$$(C^*, K_1) = ((C_1^*, \ldots, C_n^*), (K_{11} \oplus \ldots \oplus K_{1n}))$$

with $(C_i^*, K_{1i}) = \widetilde{\mathsf{Encaps}}(pk_i; s_i)$ and uniform r_i, s_i, the experiment checks whether $\exists i \in [1, n]$ such that

$$F_{\mathsf{KEM}}(r_i, s_i) = (pk_i, sk_i, C_i, K_i) = F_{\widetilde{\mathsf{KEM}}}(r_i, s_i).$$

Thus, the experiment ensures that at least one ciphertext was computed according to the specification, with a public key that was also computed according to the specification. If such a ciphertext was not output, then the game simply aborts.

Note that Game 2 and Game 1 only differ if an abort occurs *after* the watchdog has approved the implementation. Therefore, the probability for this event is

$$\begin{aligned}\Pr[\text{Abort}] &= \Pr\left[\begin{array}{c}\mathsf{WD}^{\widetilde{\mathsf{KEM}}} = \text{ false} \wedge \\ \text{Challenger aborts } \mathsf{KEMIND}^{\mathcal{A}}_{\mathsf{KEM}}(1^\lambda)\end{array}\right] \\ &\overset{(*)}{=} \Pr\left[\mathsf{WD}^{\widetilde{\mathsf{KEM}}} = \text{ false}\right] \cdot \Pr\left[\text{ Challenger aborts } \mathsf{KEMIND}^{\mathcal{A}}_{\mathsf{KEM}}(1^\lambda)\right] \\ &\leq \left(\frac{|\mathsf{Eq}|}{|R_{\mathsf{gen}} \times R_{\mathsf{enc}}|}\right)^{n_{\mathsf{WD}}} \cdot \left(\frac{|\mathsf{Neq}|}{|R_{\mathsf{gen}} \times R_{\mathsf{enc}}|}\right)^{n} \\ &\overset{(**)}{\leq} \left(\frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^{n_{\mathsf{WD}}} \cdot \left(1 - \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^{n}\end{aligned}$$

with Neq and Eq as defined in Equation (6) and Equation (7), respectively. Note here that the equation marked with $(*)$ holds because the two events are independent, since they only depend on the used randomness and the watchdog samples its randomness independently from the experiment. The following inequality holds by definition of the watchdog and the abort condition. The bound marked with $(**)$ holds since the previous line can be written as $p^\lambda \cdot (1-p)^\lambda$ for some $p \in [0, 1]$. Calculating the derivative of this function and computing the root yields that the term is maximized for

$$p = \left(\frac{|\mathsf{Eq}|}{|R_{\mathsf{gen}} \times R_{\mathsf{enc}}|}\right) = \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}.$$

Thus, if we fix n_{WD} and n, the above term states the best bound any adversary can achieve.

Now we are ready to argue that security in Game 2 is implied by the security of the underlying KEM. To this end, consider a $(t_{\mathcal{A}}, \varepsilon_2)$ adversary $\mathcal{A}_2$ which breaks the security of Game 2. From this, we construct a $(t_{\mathcal{B}}, \varepsilon)$ adversary $\mathcal{B}$ breaking the security of the underlying KEM.

Construction of $\mathcal{B}$. Adversary $\mathcal{B}$ receives as input $(1^\lambda, pk, K_b, C^*)$ for a uniformly chosen $b \in \{0,1\}$ and then simulates Game 2 as follows.

First, it obtains an implementation $\widetilde{\mathsf{KEMSR}}$ and state st from $\mathcal{A}_0$. It runs the watchdog for $\widehat{\mathsf{KEMSR}}$ as specified above. In case that watchdog outputs false, $\mathcal{B}$ outputs a uniformly random bit, just like the original security experiment. Otherwise, $\mathcal{B}$ continues to simulate Game 2.

If this is the case, then $\mathcal{B}$ generates keys $(sk, pk) = ((sk_1, \ldots, sk_n), (pk_1, \ldots, pk_n))$ using the amalgamated algorithm $\widetilde{\mathsf{Gen}}$, based on the implementation provided by $\mathcal{A}_0$. In order to compute the challenge ciphertexts, $\mathcal{B}$ computes ciphertexts C_i and keys K_i for $i \in [1, n]$ by running $\widetilde{\mathsf{Encaps}}$ using uniformly random coins. As in Game 2, $\mathcal{B}$ checks whether there exist (sk_i, pk_i, C_i, K_i) for some $i \in [1, n]$ which were computed according to the specification. In case no such pair is found, $\mathcal{B}$ aborts.

Otherwise, let i denote the smallest index for which this condition is fulfilled. $\mathcal{B}$ then computes the challenge ciphertext for $\mathcal{A}$ by replacing (C_i, K_i) (which are guaranteed to be "honestly" generated, according to the specification) by its own challenge $(C^*, K_b,)$. More formally, $\mathcal{B}$ outputs $(1^\lambda, st, pk, K, C)$ with $K = (K_1 \oplus \ldots \oplus K_n)$ and $C = (C_1, \ldots, C_n)$ to $\mathcal{A}_1$, where $(C_i, K_i) = (C^*, K_b)$. Finally $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Now observe that if $b = 0$, then K_0 was chosen uniformly from the key space $\mathcal{KS}$ and therefore the key K is uniformly random in $\mathcal{KS}$. Otherwise, if $b = 1$, then K_1 is the key encapsulated in the ciphertext C^*.

It remains to analyze the advantage of $\mathcal{B}$. Since $\mathcal{B}$ simulates Game 2 perfectly, $\mathcal{B}$ "wins" if and only if $\mathcal{A}$ wins in Game 2, *i.e.* $\varepsilon = \varepsilon_2$. Since Game 2 and Game 1 only differ by the abort condition, we obtain that

$$\varepsilon_2 = \varepsilon_1 - \Pr[\text{ Abort}] \geq \varepsilon_1 - \left(\frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^{n_{\mathsf{WD}}} \cdot \left(1 - \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^n .$$

Finally, Game 1 and Game 0 are perfectly indistinguishable due to the $(\mathcal{O}(1), t_{\mathcal{B}}, 0)$-subversion-resilience of $\widehat{\mathsf{RG}}$. Since Game 0 is the original subversion-resilience Game $\mathsf{SR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{KEMIND},\widehat{\mathsf{KEMSR}}}(1^\lambda)$, we obtain that

$$\varepsilon_0 \leq \varepsilon + \left(\frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^{n_{\mathsf{WD}}} \cdot \left(1 - \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}} + n}\right)^n$$

which completes the proof. □

5.3 Efficient Instantiation of the Subversion-Resilient KEM and the Watchdog

The variable n determines the efficiency of the constructed scheme in terms of the number of parallel instances of the underlying KEM (note that this has direct impact on the size of keys and ciphertexts), while n_{WD} determines the number of tests performed by the watchdog. Both together determine the overall security guarantee inherited from the underlying KEM.

Security parameter λ	n	$\lceil \log_2(n_{\mathsf{WD}}) \rceil$
128	32	8
128	16	11
128	8	18
128	4	33
256	64	9
256	32	12
256	16	19
256	8	34

Table 1: Instantiating our construction and the watchdog with different values n and n_{WD}. Recall that n is the number of parallel KEM instances used in our construction and n_{WD} is the number of tests (on each Gen and Encaps) done by the watchdog.

Defining n_{WD} and n as variables yields interesting tradeoffs between the watchdog's runtime, the size of ciphertexts and keys, and the obtained security bounds.

In Table 1 we consider different choices of n and n_{WD} for $\lambda \in \{128, 256\}$, *i.e.*, "128-bit" and "256-bit" security. For different values of n, we compute the number n_{WD} of tests performed by the watchdog in order to achieve that

$$\left(\frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}}+n}\right)^{n_{\mathsf{WD}}} \cdot \left(1 - \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}}+n}\right)^{n} \leq 2^{-\lambda}$$

holds. Note that together with the assumption that the underlying KEM is instantiated such that it provides $\varepsilon \leq 2^{-\lambda}$, we thus obtain a security bound on the subversion-resilient KEM of

$$\varepsilon_0 \leq \varepsilon + \left(\frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}}+n}\right)^{n_{\mathsf{WD}}} \cdot \left(1 - \frac{n_{\mathsf{WD}}}{n_{\mathsf{WD}}+n}\right)^{n} \leq 2^{-\lambda+1}.$$

Table 1 shows how our subversion-resilient KEM can be instantiated. For instance, for $\lambda = 128$ and with $n = 8$, the watchdog only needs to test the Gen and Encaps algorithm only 2^{18} times, which can be practically accomplished for many underlying KEM constructions within a short time on moderate hardware. Even for $\lambda = 128$ and with n as small as $n = 4$ only 2^{33} tests are already sufficient, which also seems practically feasible, since it can be accomplished for most underlying KEMs within minutes or at most few hours on standard hardware such as a laptop computer.

6 Subversion-Resilient Public-Key Encryption

After successfully constructing a subversion-resilient KEM, we now proceed to construct a subversion-resilient public key encryption scheme. We will show that the standard way to construct public-key encryption from a KEM also preserves subversion-resilience, provided that a trusted XOR operation is given.

```
$\mathsf{PKEIND}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda)$
$(sk, pk) \leftarrow \mathsf{Gen}_{\mathsf{PKE}}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, pk)$
$b \xleftarrow{\$} \{0,1\}$
$C^* \leftarrow \mathsf{Encrypt}(pk, m_b)$
$b_{\mathcal{A}} \leftarrow \mathcal{A}(1^\lambda, C^*)$
If $b == b_{\mathcal{A}}$ then Return $1$
Else Return $0$
```

(a) Indistinguishability experiment for public key encryption schemes.

```
$\mathsf{SR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{PKEIND},\widehat{\mathsf{PKESR}}}(1^\lambda)$
$(\widetilde{\mathsf{PKESR}}, st) \leftarrow \mathcal{A}_0(1^\lambda)$
If $\mathsf{WD}^{\widetilde{\mathsf{PKESR}}}(1^\lambda)$ then
    $b \xleftarrow{\$} \{0,1\}$
    Return $b$
Return $\mathsf{PKEIND}^{\mathcal{A}_1(st)}_{\mathsf{Am}(\widetilde{\mathsf{PKESR}})}(1^\lambda)$
```

(b) Subversion-resilience experiment for public key encryption schemes.

Fig. 7: Security experiments for public key encryption schemes.

6.1 Definitions and Construction

We begin by recalling the standard definition for public key encryption and its standard IND-CPA-style security definition.

Definition 8. *Let* $\mathsf{PKE} = (\mathsf{Gen}_{\mathsf{PKE}}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be a public key encryption scheme with the following syntax:*

- $\mathsf{Gen}_{\mathsf{PKE}}(1^\lambda)$: *The randomized key-generation algorithm takes as input a security parameter* $\lambda \in \mathbb{N}$ *and outputs a key pair* (sk, pk).
- $\mathsf{Encrypt}(pk, m)$: *The randomized encrypt algorithm takes as input the public key* pk *and a message* m *and outputs the ciphertext* C.
- $\mathsf{Decrypt}(sk, C)$: *The deterministic decryption algorithm takes as input the secret key* sk *and the ciphertext* C*. It outputs a message* m *or the error symbol* $\perp$.

Definition 9. *We say that* $\mathsf{PKE} = (\mathsf{Gen}_{\mathsf{PKE}}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is* $(t_{\mathcal{A}}, \varepsilon)$-indistinguishable *if for any adversary* $\mathcal{A}$ *running in time at most* $t_{\mathcal{A}}$ *it holds that*

$$\mathsf{Adv}^{\mathsf{PKE}}_{\mathcal{A}}(1^\lambda) := |\Pr[\mathsf{PKEIND}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda)] - 1/2| \leq \varepsilon$$

with $\mathsf{PKEIND}^{\mathcal{A}}_{\mathsf{PKE}}(1^\lambda)$ *shown in Figure 7 (a).*

Definition 10. *We say that a specification of a public key encryption scheme* $\widehat{\mathsf{PKESR}} = (\mathsf{Am}_{\mathsf{PKE}}, \mathsf{PKESR})$ *is* $(t_{\mathsf{WD}}, t_{\mathcal{A}}, \varepsilon)$-subversion-resilient in the offline watchdog model with trusted amalgamation *if one can efficiently construct a* correct *watchdog* WD *running in time at most* t_{WD} *such that for any adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *running in time* $t_{\mathcal{A}}$ *it holds that*

$$\mathsf{AdvSR}^{\mathcal{A},\mathsf{WD}}_{\mathsf{PKEIND},\widehat{\mathsf{PKESR}}}(1^\lambda) \leq \varepsilon$$

with the used experiments shown in Figure 7 (a) and (b).

Description of the construction. Let $\widehat{\mathsf{KEMSR}} = (\mathsf{Am}_{\mathsf{KEM}}, \mathsf{KEMSR})$ be the specification of a subversion-resilient key encapsulation mechanism with

$$\mathsf{Am}_{\mathsf{KEM}}(\widehat{\mathsf{KEMSR}}) = (\mathsf{GenSR}_{\mathsf{KEM}}, \mathsf{EncapsSR}, \mathsf{DecapsSR}).$$

We then construct the specification of a public key encryption scheme $\widehat{\mathsf{PKESR}} = (\mathsf{Am}_{\mathsf{PKESR}}, \mathsf{KEMSR})$ with

$$\mathsf{Am}_{\mathsf{PKESR}}(\mathsf{KEMSR}) = (\mathsf{GenSR}_{\mathsf{PKE}}, \mathsf{EncryptSR}, \mathsf{DecryptSR})$$

where each algorithm is defined as follows:

- $\mathsf{GenSR}_{\mathsf{PKE}}(1^\lambda)$: Output $(sk, pk) = \mathsf{GenSR}_{\mathsf{KEM}}(1^\lambda)$.
- $\mathsf{EncryptSR}(pk, m)$: Compute $(C, K) \leftarrow \mathsf{EncapsSR}(pk)$ and output $(C, K \oplus m)$.
- $\mathsf{DecryptSR}(sk, C)$: Parse $C = (C_0, C_1)$. Compute $K \leftarrow \mathsf{DecapsSR}(sk, C_0)$. Output $m = C_1 \oplus K$.

Thus, the specification of our public key encryption scheme is basically the specification of the underlying subversion-resilient key encapsulation mechanism. Thus, the amalgamation $\mathsf{Am}_{\mathsf{PKESR}}$ is almost identical to $\mathsf{Am}_{\mathsf{KEM}}$. the only difference is that during encrypt, the message is additionally XOR'ed to the key K.

Security analysis. Subversion-resilience of the new public key encryption scheme follows directly from the security of the underlying KEM and the usage of a trusted $\oplus$.

Theorem 3. *Let* $\widehat{\mathsf{KEMSR}} = (\mathsf{Am}_{\mathsf{KEM}}, \mathsf{KEMSR})$ *be the specification of a* $(t_{\mathsf{WD}}, t_{\mathcal{A}}, \epsilon)$ *subversion-resilient KEM. Then* $\widehat{\mathsf{PKESR}}$ *as described above is* $(t_{\mathsf{WD}}, t_{\mathcal{A}}, \epsilon)$ *subversion-resilient under a trusted* $\oplus$ *operation.*

Proof sketch. The watchdog for PKE simply runs the watchdog for $\widehat{\mathsf{KEMSR}}$ as a subroutine. Thus, either the watchdog detects subversion or the ciphertext-key-pair output is ε-indistinguishable. Since the $\oplus$ operation is trusted, the resulting ciphertexts of PKE are also ε-indistinguishable. Therefore PKE is subversion-resilient iff $\widehat{\mathsf{KEMSR}}$ is subversion-resilient.

Note that while the adversary can freely choose the messages in the experiment, input trigger attacks are not possible. This is because no subverted algorithm has direct access to the message m_b, since the XOR operation used to encrypt the message with the KEM key is part of the trusted amalgamation.

Acknowledgments. We would like to thank Moti Yung and the anonymous reviewers of PKC 2021 for their helpful comments and suggestions, and in particular Cristina Onete for shepherding this paper and providing very detailed and valuable inputs.

References

1. Ateniese, G., Francati, D., Magri, B., Venturi, D.: Public immunization against complete subversion without random oracles. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 19. LNCS, vol. 11464, pp. 465–485. Springer, Heidelberg (Jun 2019)
2. Ateniese, G., Magri, B., Venturi, D.: Subversion-resilient signature schemes. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 364–375. ACM Press (Oct 2015)
3. Bellare, M., Fuchsbauer, G., Scafuro, A.: NIZKs with an untrusted CRS: Security in the face of parameter subversion. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 777–804. Springer, Heidelberg (Dec 2016)

4. Bellare, M., Hoang, V.T.: Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 627–656. Springer, Heidelberg (Apr 2015)
5. Bellare, M., Jaeger, J., Kane, D.: Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 1431–1440. ACM Press (Oct 2015)
6. Bellare, M., Paterson, K.G., Rogaway, P.: Security of symmetric encryption against mass surveillance. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 1–19. Springer, Heidelberg (Aug 2014)
7. Berndt, S., Liskiewicz, M.: Algorithm substitution attacks from a steganographic perspective. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1649–1660. ACM Press (Oct / Nov 2017)
8. Blum, M., Luby, M., Rubinfeld, R.: Self-testing/correcting with applications to numerical problems. In: 22nd ACM STOC. pp. 73–83. ACM Press (May 1990)
9. Bossuat, A., Bultel, X., Fouque, P.A., Onete, C., van der Merwe, T.: Designing reverse firewalls for the real world. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020, Part I. LNCS, vol. 12308, pp. 193–213. Springer, Heidelberg (Sep 2020)
10. Chakraborty, S., Dziembowski, S., Nielsen, J.B.: Reverse firewalls for actively secure MPCs. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 732–762. Springer, Heidelberg (Aug 2020)
11. Chen, R., Huang, X., Yung, M.: Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 98–128. Springer, Heidelberg (Dec 2020)
12. Chen, R., Mu, Y., Yang, G., Susilo, W., Guo, F., Zhang, M.: Cryptographic reverse firewall via malleable smooth projective hash functions. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part I. LNCS, vol. 10031, pp. 844–876. Springer, Heidelberg (Dec 2016)
13. Chow, S.S.M., Russell, A., Tang, Q., Yung, M., Zhao, Y., Zhou, H.S.: Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog. In: Lin, D., Sako, K. (eds.) PKC 2019, Part I. LNCS, vol. 11442, pp. 221–251. Springer, Heidelberg (Apr 2019)
14. Claburn, T.: NSA: We've learned our lesson after foreign spies used one of our crypto backdoors – but we can't say how exactly. The Register, `https://www.theregister.com/2020/10/28/nsa_backdoor_wyden/` (2020)
15. Degabriele, J.P., Farshim, P., Poettering, B.: A more cautious approach to security against mass surveillance. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 579–598. Springer, Heidelberg (Mar 2015)
16. Dodis, Y., Impagliazzo, R., Jaiswal, R., Kabanets, V.: Security amplification for interactive cryptographic primitives. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 128–145. Springer, Heidelberg (Mar 2009)
17. Dodis, Y., Mironov, I., Stephens-Davidowitz, N.: Message transmission with reverse firewalls—secure communication on corrupted machines. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 341–372. Springer, Heidelberg (Aug 2016)
18. Dziembowski, S., Faust, S., Standaert, F.X.: Private circuits III: Hardware trojan-resilience via testing amplification. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 142–153. ACM Press (Oct 2016)
19. Fischlin, M., Mazaheri, S.: Self-guarding cryptographic protocols against algorithm substitution attacks. In: CSF. pp. 76–90. IEEE Computer Society (2018)
20. Fuchsbauer, G.: Subversion-zero-knowledge SNARKs. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 315–347. Springer, Heidelberg (Mar 2018)
21. Giacon, F., Kiltz, E., Poettering, B.: Hybrid encryption in a multi-user setting, revisited. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 159–189. Springer, Heidelberg (Mar 2018)

22. Holenstein, T., Renner, R.: One-way secret-key agreement and applications to circuit polarization and immunization of public-key encryption. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 478–493. Springer, Heidelberg (Aug 2005)
23. Jain, A., Korb, A., Manohar, N., Sahai, A.: Amplifying the security of functional encryption, unconditionally. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 717–746. Springer, Heidelberg (Aug 2020)
24. Jain, A., Manohar, N., Sahai, A.: Combiners for functional encryption, unconditionally. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 141–168. Springer, Heidelberg (May 2020)
25. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 657–686. Springer, Heidelberg (Apr 2015)
26. Poettering, B., Rösler, P.: Combiners for AEAD. IACR Trans. Symmetric Cryptol. 2020(1), 121–143 (2020)
27. Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Cliptography: Clipping the power of kleptographic attacks. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016, Part II. LNCS, vol. 10032, pp. 34–64. Springer, Heidelberg (Dec 2016)
28. Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Generic semantic security against a kleptographic adversary. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 907–922. ACM Press (Oct / Nov 2017)
29. Russell, A., Tang, Q., Yung, M., Zhou, H.S.: Correcting subverted random oracles. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 241–271. Springer, Heidelberg (Aug 2018)
30. von Neumann, J.: Various techniques used in connection with random digits. In: Householder, A., Forsythe, G., Germond, H. (eds.) Monte Carlo Method, pp. 36–38. National Bureau of Standards Applied Mathematics Series, 12, Washington, D.C.: U.S. Government Printing Office (1951)
31. Young, A., Yung, M.: The dark side of "black-box" cryptography, or: Should we trust capstone? In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 89–103. Springer, Heidelberg (Aug 1996)
32. Young, A., Yung, M.: Kleptography: Using cryptography against cryptography. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 62–74. Springer, Heidelberg (May 1997)
33. Zhang, C., Cash, D., Wang, X., Yu, X., Chow, S.S.M.: Combiners for chosen-ciphertext security. In: COCOON. Lecture Notes in Computer Science, vol. 9797, pp. 257–268. Springer (2016)