IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 2, pp. 389–422. DOI:10.46586/tosc.v2021.i2.389-422

Algorithm Substitution Attacks: State Reset Detection and Asymmetric Modifications

Philip Hodges and Douglas Stebila

University of Waterloo, Waterloo, Canada
pwhodges@uwaterloo.ca, dstebila@uwaterloo.ca

Abstract. In this paper, we study algorithm substitution attacks (ASAs), where an algorithm in a cryptographic scheme is substituted for a subverted version. First, we formalize and study the use of state resets to detect ASAs, and show that many published stateful ASAs are detectable with simple practical methods relying on state resets. Second, we introduce two asymmetric ASAs on symmetric encryption, which are undetectable or unexploitable even by an adversary who knows the embedded subversion key. We also generalize this result, allowing for any symmetric ASA (on any cryptographic scheme) satisfying certain properties to be transformed into an asymmetric ASA. Our work demonstrates the broad application of the techniques first introduced by Bellare, Paterson, and Rogaway (Crypto 2014) and Bellare, Jaeger, and Kane (CCS 2015) and reinforces the need for precise definitions surrounding detectability of stateful ASAs.

Keywords: Algorithm Substitution Attack, Symmetric Encryption, State Reset, Kleptography

1 Introduction

In this paper, we study Algorithm Substitution Attacks (ASAs). ASAs were first presented by Bellare, Paterson, and Rogaway [BPR14], as a specific possibility within the research area of kleptography [YY96, YY97, YY98, YY04, YY03]. In an ASA on a symmetric encryption scheme, an attacker has the following goal: replace an encryption function with a subverted version, such that, upon observing ciphertexts, the attacker is able to recover the secret key while the user is unable to detect the difference between the subverted and original encryption algorithms [BPR14]. Such an attack is certainly outside of the traditional security definitions considered in cryptography. However, revelations of the activities of nation-state intelligence agencies since 2013 [BBG13] have shown that such attacks are aligned well with both the intentions and the capabilities of these potential adversaries. Under threat of mass surveillance, it is imperative for us to study the ways in which cryptographic security may be undermined so we can best mitigate the effects.

We contribute two main improvements to the existing literature on ASAs. First, we formalize a possible method for detecting ASAs involving the manipulation of the state maintained by the subverted algorithm. We describe a model that allows a detector to force re-use of the algorithm's maintained state, as could happen when the algorithm is running on a virtual machine. This changes the detectability analysis of several published ASAs, rendering them easily detectable. Second, we describe two modifications to existing ASAs that turn them into asymmetric ASAs, which are more resilient to exploitation by another party who reverse engineers the subverted implementation.

Background. The possibility of subverting a cryptographic algorithm by changing the implementation to differ from the specification is not new. Termed kleptography, research

Received: 2021-03-01 Accepted: 2021-05-01 Published: 2021-06-11

in this area was initiated by Young and Yung [YY96], who did several follow-up works [YY97, YY98, YY04, YY03]. Their work was inspired by subliminal channels in public-key cryptosystems [Sim85, Des90], which they then showed could create serious problems. Several authors have studied backdoors in symmetric encryption [RP97, Pat99]. Goh, Boneh, Pinkas, and Golle [GBPG03] studied implementations of TLS/SSL and SSH that provide key recovery capabilities. Schneier, Fredrikson, Kohno, and Ristenpart published a survey of cryptographic subversion in general [SFKR15].

Crucial in the development of this area of research were the revelations by Edward Snowden of the activities of the NSA in efforts to break the encryption used for internet traffic [BBG13]. This included some information on one case of successful cryptographic subversion in practice: the pseudorandom bit generator Dual_EC_DRBG [CNE+14]. These events renewed interest in this area of research, starting with the work of Bellare, Paterson, and Rogaway [BPR14] on ASAs.

Algorithm substitution attacks (ASAs). Suppose a user $\mathcal{U}$ is using a symmetric encryption scheme SE for encryption of their communications. An attacker (sometimes referred to as Big Brother) is able to exchange the encryption algorithm $\mathsf{SE.Enc}$ for an alternative algorithm $\mathsf{Sub.Enc}$, which we call the subverted algorithm. While using the algorithm $\mathsf{Sub.Enc}$, $\mathcal{U}$ will send a set of ciphertexts. From observation of these ciphertexts, the attacker wants to be able to recover the secret key k used for encryption. If this was the only requirement, an ASA would be trivial: the subverted encryption algorithm could simply output the secret key on any input. However, the attacker would like continuous exploitation of the system, and so wishes for $\mathcal{U}$ to be unaware of the subversion. For this we require an ASA to be undetectable, meaning that $\mathcal{U}$ should not be able to (with black box access) distinguish between $\mathsf{Sub.Enc}$ and $\mathsf{SE.Enc}$. A minimum requirement for undetectability would be that all (or all but a negligible fraction) of the possible ciphertexts correctly decrypt, but this alone would not be enough. The formal requirement is captured in a detectability game that the user $\mathcal{U}$ plays.

A variety of techniques can be used to create an ASA, but in general an ASA relies on the attacker sharing a key $\overline{k}$ with the subverted algorithm, and requires that $\mathsf{SE.Enc}$ is randomized. Bellare, Paterson, and Rogaway [BPR14], in their seminal paper, presented a technique involving acceptance-rejection on ciphertexts generated by the unsubverted encryption algorithm. A pseudorandom function is evaluated on $\overline{k}$ and a ciphertext, giving a single bit b. If the first bit of secret key is equal to b, then that ciphertext is returned; otherwise, a new one is computed. Since the attacker knows $\overline{k}$, he can recover the first bit of the secret key k from the ciphertext. Repeating this for each position within the secret key (which the ASA maintains as state) allows for recovery of the whole key. Since $\overline{k}$ is unknown to $\mathcal{U}$, the ciphertexts still appear randomly generated. Hence both key recovery and undetectability are achieved. [BPR14] called this the "biased-ciphertext attack."

Degabriele, Farshim, and Poettering [DFP15] gave refinements to the definitions of [BPR14]. In particular, they relaxed a requirement that all ciphertexts generated by the subverted encryption algorithm must decrypt correctly. They then introduced the notion of an "input-triggered" ASA, where a certain input would lead to leaking of the secret key k without regard for correct decryptability. This requires influence over the distribution of encrypted messages in order to enable key recovery. Bellare, Jaeger, and Kane [BJK15] improved on the results of [BPR14], notably introducing a stateless version of the biased-ciphertext attack. In this attack, instead of keeping an index as state, it is generated pseudorandomly along with the bit b, rendering the ASA stateless.

Aside from symmetric encryption, ASAs on other cryptographic primitives have also been studied. Several authors have published ASAs on signature schemes [AMV15, BSKC19, LCWW18]. Armour and Poettering explored ASAs on MAC schemes and the decryption side of authenticated encryption [AP19a, AP19b]. Chen, Huang, and Yung

presented an ASA against KEMs [CHY20]. More recently, Berndt et al. studied the implementation of ASAs on the TLS, WireGuard, and Signal protocols [BWP+20].

1.1 State resets for detection of ASAs

In the literature, there has been a tendency towards ensuring that new ASAs are stateless. [BPR14], who coined the term ASA, noted that the biased-ciphertext attack they introduced was stateful. They said that, since the attack is stateful, "a reset of the state will lead to increased detection ability for an observer, but ... this increase does not appear to be enough to lead to actual detection." Improving on the results of [BPR14], [BJK15] presented their stateless ASA. They seem to interpret some of the conclusions from [BPR14] differently, saying, with reference to the previous work, "a state reset, as can happen with a reboot or cloning to create a virtual machine, leads, in their attack, to detection." They then define a notion of undetectability that necessitates statelessness, and call this *strong undetectability*. As a result of the interpretations and emphasis of [BJK15], as well as the fact that stateless subversions have proven more difficult to develop, later work has often acknowledged that stateless schemes are surely preferable. Authors detailing stateful subversions have spent time justifying that the amount of state that they are maintaining is small, and so more palatable for the adversary to include [BSKC19, CHY20].

This begs a question: how does an adversary against the undetectability of a subversion use state resets to detect the subversion? To our knowledge, this question has not been addressed fully in the literature. The closest example is due to Baek, Susilo, Kim, and Chow [BSKC19], who included a simple state reset oracle in their undetectability game, which resets the state to the initial null value. However, as noted by [BJK15], we also wish to consider what happens when the algorithm is running on a virtual machine, where the machine state can be cloned and re-run from the same point, potentially many times. The simple state reset oracle is therefore insufficient.

Contributions. We present a stronger state reset oracle than that used by [BSKC19], which is able to reset the state of the ASA to *any* state previously used in the detection game. Under this new definition, we show that ASAs given by Ateniese, Magri, and Venturi [AMV15] (on signatures), Baek et al. [BSKC19] (on DSA signatures), and Chen, Huang, and Yung [CHY20] (on key exchange) are all easily detectable. On the other hand, we show that original biased-ciphertext ASA by [BPR14] is actually just as undetectable as the "upgraded", stateless version given by [BJK15]. Our analysis of the ASA from [BPR14] also uses the same game-playing proof framework as used in [BJK15], avoiding the "coin-injective" assumption on the encryption scheme that was necessary in [BPR14]. We present these results in Section 3.

1.2 Asymmetric ASAs

When introducing ASAs, [BPR14] also considered the possibility of an asymmetric ASA on symmetric encryption. An *asymmetric* ASA is one where the subverted algorithm uses an *embedded* key that is different from the *extraction* key used for key recovery; for example, the two keys could be a public-private key pair. The motivation for this comes from noticing that the embedded subversion key is not particularly protected. Instead, it is embedded in, for example, malware, distributed to the target. While we assume in this model that the target themselves will not scrutinize the code they are using, some third party might find out about the subversion, and reverse engineer the software to learn the embedded key. The subverter would have a strong incentive to prevent a third party from obtaining the same key recovery capabilities as the subverter. If the embedded key is presumed to be public knowledge, and the ASA remains undetectable, then the subverter is assured that they are the only one capable of exploiting the ASA. In an

appendix, [BPR14] give the necessary definitional extensions for asymmetric ASAs, and leave the development of an asymmetric ASA as an open problem. Later works considered asymmetric ASAs in certain specific contexts, like on signature schemes and KEMs that satisfy certain conditions [CHY20, BSKC19].

Contributions. In this paper, we will consider two different kinds of asymmetric ASAs. In a *type 1* asymmetric ASA, the subversion is required to be undetectable to an adversary in possession of the embedded subversion key; we call this augmented undetectability. This is the simplest way of thinking about an asymmetric ASA, and the definition that has been used in other literature. In a *type 2* asymmetric ASA, we will instead require that the subversion is only undetectable to an adversary who does not know the embedded subversion key, as in the case of a symmetric ASA, but we also require that a type 2 asymmetric ASA is secure against exploitation (in the sense that the attacker exploits the ASA) by an adversary in possession of the embedded subversion key. This less restrictive requirement is a reflection of the fact that the main goals for an asymmetric ASA (besides recovering the targeted information) are as follows: to ensure that the user of the cryptographic scheme (or some entity with the decision-making authority to halt usage of the cryptographic scheme) being attacked is unaware of the attack, and to ensure that no other entity is able to exploit the ASA to recover the targeted information. While this is accomplished by a type 1 asymmetric ASA (indeed, a type 1 ASA is also a type 2 ASA), our stipulated requirements for a type 2 asymmetric ASA will also suffice. The relaxed requirements allow for more flexibility when designing an ASA, and allows us to create an ASA whose executions take less time.

In Section 4, we modify the ASA of [BPR14] to obtain a type 1 asymmetric ASA on symmetric encryption, that is, an ASA undetectable by an adversary who is in possession of the embedded subversion key and is able to use state resets on the encryption scheme. This provides an answer to their open problem explicitly in the case of symmetric encryption. In Section 5 we modify the ASA of [BJK15] (which is itself a modification of the ASA from [BPR14]) to obtain a type 2 asymmetric ASA on symmetric encryption. To show the advantages of this ASA, we do a thorough analysis of the parameters and techniques the attacker can use in practice to recover the key. We show that our type 2 asymmetric ASA can enable key recovery in practice with a subverted encryption function which runs in less time, making it, in theory, less susceptible to detection by timing.

In order to give a better idea of how these new ASAs compare to other published ASAs, we give a comparison of some basic properties in Table 1.

Finally, in Section 6 we give a generalization of the modifications we made to the above ASAs, in order to apply our results to other cryptographic primitives and security notions. Our results allow for a large class of ASAs to be modified to create type 1 and type 2 asymmetric ASAs. These results apply to any cryptographic primitive, and in the case of the type 2 modification, any game-based notion of security. These results will make it easier for future researchers to evaluate whether their symmetric ASAs can be modified to create asymmetric ASAs.

2 Preliminaries and Definitions

2.1 Games and algorithms

Proofs in this paper will use the cryptographic game-playing framework [BR06]. In these games, assignment is denoted by $\leftarrow$, while random sampling is denoted by $\xleftarrow{\$}$. We write $y \xleftarrow{\$} \mathsf{A}(x)$ to denote running the probabilistic algorithm A on input x, and assigning the result to y. If we wish to specify the random coins r used in a randomized algorithm, we will write $y \leftarrow \mathsf{A}(x; r)$. We will use min-entropy as a measure of the randomness of an

Table 1: Comparison of properties of various ASAs. An augmented adversary refers to an adversary in possession of the embedded subversion key. If applicable, $|k| = 128$.

	[BPR14]	[BJK15]	[BSKC19]	[CHY20]	Our type 1	Our type 2
Asymmetric	○	○	●	●	●	●
No state reset						
Undetectable vs. regular adversary	●	●	●	●	●	●
Undetectable vs. augmented adversary	○	○	●	●	●	○
Secure vs. augmented adversary	○	○	●	●	●	●
State reset						
Undetectable vs. regular adversary (SRDET)	●	●	○	○	●	●
Undetectable vs. augmented adversary (ASRDET)	○	○	○	○	●	○
Secure vs. augmented adversary	○	○	●	●	●	●
Intercepted transmissions needed	128	≈ 700	3	2	400	≈ 2600
Runtime multiplier	≥ 7	≥ 2	≈ 1	≈ 1	≥ 9	≥ 2

algorithm. Define η_{A} according to $2^{-\eta_{\mathsf{A}}} = \max_{x,y} (\Pr[y \leftarrow \mathsf{A}(x;r)])$, where the probability is taken over the choice of coins r. The min-entropy of A is η_{A}. $\perp$ is used to denote a null value. If G is a game, then $\Pr[G]$ indicates the probability that G returns true.

We will denote game adversaries by script letters (e.g. $\mathcal{A}$). An adversary $\mathcal{A}$ is simply an algorithm. The notation $\mathcal{A}^{\mathcal{O}}$ indicates that the adversary $\mathcal{A}$ has access to the oracle $\mathcal{O}$ for use as a subroutine. The running time of $\mathcal{A}$ is the worst-case execution time of $\mathcal{A}$ including the time it takes to execute any subroutines.

2.2 Cryptographic schemes

A cryptographic scheme Λ is a set of algorithms $\Lambda.\mathsf{Alg}_1, ..., \Lambda.\mathsf{Alg}_u$. We will be using several cryptographic schemes in this paper, including symmetric encryption and public-key encryption. We introduce these here, and other schemes as needed.

A symmetric encryption scheme SE is composed of three algorithms: $\mathsf{SE.KeyGen}$, $\mathsf{SE.Enc}$, and $\mathsf{SE.Dec}$. $\mathsf{SE.KeyGen}$ randomly selects a single secret key k of length $\mathsf{SE.klen}$ from $\{0,1\}^{\mathsf{SE.klen}}$. $\mathsf{SE.Enc}$ is a randomized algorithm with coins $r \in \{0,1\}^{\mathsf{SE.rlen}}$ and takes a key and a plaintext $m \in \{0,1\}^{\mathsf{SE.mlen}}$ and produces a ciphertext $c \in \{0,1\}^{\mathsf{SE.clen}}$. $\mathsf{SE.Dec}$ is a deterministic algorithm, takes a key and a ciphertext, and returns a plaintext or $\perp$, indicating an error.

A public-key (or asymmetric) encryption scheme PKE is similarly composed of three algorithms: $\mathsf{PKE.KeyGen}$, $\mathsf{PKE.Enc}$, and $\mathsf{PKE.Dec}$. $\mathsf{PKE.KeyGen}$ randomly generates a secret key sk and a public key pk. $\mathsf{PKE.Enc}$ is a randomized algorithm with coins $r \in \{0,1\}^{\mathsf{PKE.rlen}}$ and takes a public key and a plaintext $m \in \{0,1\}^{\mathsf{PKE.mlen}}$ and produces a ciphertext $c \in \{0,1\}^{\mathsf{PKE.clen}}$ (note that we will consider only fixed length ciphertexts for public-key schemes). $\mathsf{PKE.Dec}$ is a deterministic algorithm, takes a secret key and a ciphertext, and returns a plaintext or $\perp$, indicating an error.

We say that a public-key encryption scheme is δ-correct if, for all sk, pk generated from $\mathsf{PKE.KeyGen}$ and m, $\Pr[\mathsf{PKE.Dec}(sk, \mathsf{PKE.Enc}(pk, m)) = m] \geq \delta$, where the probability is taken over the choice of coins r for the encryption function. We could define an analogous property for symmetric encryption, but we will mostly assume that such a δ value will always be 1 unless otherwise stated.

$\mathrm{PRF}_{\mathsf{F}}(\mathcal{F})$

1. $k \leftarrow\$ K_{\mathsf{F}}$
2. $X \leftarrow \emptyset$
3. $b \leftarrow\$ \{0,1\}$
4. $b' \leftarrow\$ \mathcal{F}^{\mathcal{O}_{\mathsf{F}}}$
5. **return** $b = b'$

$\mathcal{O}_{\mathsf{F}}(x)$

1. **if** $b = 0$ **then** $w \leftarrow \mathsf{F}(k, x)$
2. **if** $b = 1$ **then**
3. **if** $x \notin X$ **then**
4. $w_x \leftarrow\$ W$
5. $X \leftarrow X \cup \{x\}$
6. **return** w_x

Figure 1: The PRF security game.

2.3 Pseudo-random functions and random oracles

In this paper, we will make use of two standard ways of talking about functions whose output is hard to predict on new inputs. The first is the notion of a pseudo-random function (PRF). Let $\mathsf{F} : K_{\mathsf{F}} \times \{0,1\}^* \to W$ be a function, for some output set W and key space K_{F}. Let the PRF game for F be as defined in Figure 1. For an adversary $\mathcal{F}$ in the PRF game for F, we define the advantage of $\mathcal{F}$ as $\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F}) = \left|\Pr[\mathrm{PRF}_{\mathsf{F}}(\mathcal{F})] - \frac{1}{2}\right|$.

The second way to talk about functions with unpredictable output is by using the random oracle model. This model is useful for situations in which there is no secret input to the function F, so the game in Figure 1 is no longer relevant. In this model, we replace F by a lazily-sampled random function, and provide oracle access to this function to all game adversaries. A lazily-sampled random function will return random outputs on previously-unseen queries, and outputs consistent with previous outputs for any previously-seen queries. (In the case of $b = 1$ in the PRF game, the oracle $\mathcal{O}_{\mathsf{F}}$ behaves as a lazily-sampled random function.)

2.4 Algorithm substitution attacks

The main focus of this paper is on different kinds of Algorithm Substitution Attacks (ASAs), and so we define this notion here. Let Λ be a cryptographic scheme composed of algorithms $\Lambda.\mathsf{Alg}_1, ..., \Lambda.\mathsf{Alg}_u$. An ASA on Λ, denoted Sub (for subversion), specifies the following:

- a subversion-key generation function $\mathsf{Sub.KeyGen}$,
- an index λ for the component algorithm of Λ to be subverted, and
- a subverted algorithm $\mathsf{Sub.Alg}_\lambda$ to replace the chosen algorithm $\Lambda.\mathsf{Alg}_\lambda$.

The key generation algorithm $\mathsf{Sub.KeyGen}$ takes no arguments and returns a pair of keys ek and xk (for embedded key and extraction key). The subverted algorithm $\mathsf{Sub.Alg}_\lambda$ has the same inputs as $\Lambda.\mathsf{Alg}_\lambda$, represented by a tuple x, plus an embedded key ek, and a state variable τ (potentially $\perp$, for stateless ASAs); $\mathsf{Sub.Alg}_\lambda$ has the same outputs as $\Lambda.\mathsf{Alg}_\lambda$ plus the updated state variable τ'. For example, if Λ is a symmetric encryption scheme SE and $\Lambda.\mathsf{Alg}_\lambda$ is $\mathsf{SE.Enc}$, then we have $c, \tau' \leftarrow\$ \mathsf{Sub.Enc}(k, m, ek, \tau)$.[1]

The idea here is that the algorithm $\mathsf{Sub.Alg}_\lambda$ is chosen by an adversary $\mathcal{A}$ who is trying to subvert the security of scheme Λ. A user $\mathcal{U}$ of Λ will unknowingly use $\mathsf{Sub.Alg}_\lambda$, which has the key ek embedded, in place of $\Lambda.\mathsf{Alg}_\lambda$. The adversary $\mathcal{A}$ will observe $\mathcal{U}$'s communication with other users of the scheme Λ, and violate the security of Λ by making use of the extraction key xk. Depending on the instantiation, $\mathcal{A}$'s specific attack goal can vary, although a common one is recovery of whatever secret key is used by $\mathcal{U}$ for $\mathsf{Sub.Alg}_\lambda$.

Note that we consider here only the subversion of a single algorithm of the scheme Λ. Other works have considered the case of total subversion (any or all of the algorithms are substituted), mostly in the context of countermeasures [AFMV19, RTYZ17, RTYZ16].

[1] For clarity we make a slight abuse of notation here, writing $\mathsf{Sub.Enc}(k, m, ek, \tau)$ instead of $\mathsf{Sub.Enc}(x, ek, \tau)$ for $x = (k, m)$.

The two keys used by $\mathcal{A}$ can be the same, $xk = ek$. In this case, we may denote them simply by $\overline{k}$, and we refer to this type of ASA as a symmetric ASA. In other cases, (xk, ek) will be a private key-public key pair, reflecting the fact that embedding the key ek into an $\mathsf{Sub.Alg}_\lambda$ may lead to its recovery by some other party. We call such an ASA an asymmetric ASA. Note that an *asymmetric ASA* can be used to attack a *symmetric-key primitive* Λ (e.g., symmetric encryption), and vice versa. We will discuss the advantages and disadvantages of an asymmetric ASA in Section 4.

The attacker $\mathcal{A}$ wants to complete their attack in a way that is not detectable by $\mathcal{U}$. In order to measure detectability, we allow $\mathcal{U}$ blackbox access to the algorithm $\mathsf{Sub.Alg}_\lambda$ (with the embedded key and state implicitly provided), and calculate how well $\mathcal{U}$ can differentiate $\mathsf{Sub.Alg}_\lambda$ from $\Lambda.\mathsf{Alg}_\lambda$. Formalization of the notion of detectability will be covered in the next section.

There are a few different ways that previous works have considered the subverter's influence on $\mathcal{U}$ after the relevant algorithm has been substituted. Bellare, Paterson, and Rogaway [BPR14] implicitly considered the attacker passive (after the substitution), only observing ciphertexts of messages that the user decided to encrypt. Degabriele, Farshim, and Poettering [DFP15] presented an "input-triggered" subversion, which relied on the subverter's ability to influence the user to encrypt some particular value, making the subverter active. Bellare, Jaeger, and Kane [BJK15], addressed this issue by requiring key recovery to occur for messages sampled from any distribution $\mathcal{M}$, ruling out the attack from [DFP15] and again making the subverter passive. We follow [BJK15] and consider a passive subverter who should be able to complete their attack no matter how inputs are chosen for $\mathsf{Sub.Alg}_\lambda$.

3 Using State Reset to Detect ASAs

State reset detection techniques against the undetectability of stateful ASAs have been acknowledged since the work of Bellare, Paterson, and Rogaway [BPR14], but apart from Baek, Susilo, Kim, and Chow [BSKC19], the formalization of the state reset capabilities of a user $\mathcal{U}$ has been ignored. In their paper, [BSKC19] capture the idea of state reset with the ability of $\mathcal{U}$ to reset state to a null value. This is akin to wiping the memory of the program running a cryptographic algorithm, or rebooting the machine. We wish, however, to capture a stronger notion of state reset. For example, when running on a virtual machine, instead of being wiped, memory could be cloned during imaging, allowing a user to force a program to run from the same point of execution multiple times. This would allow a user to reset the state of a subverted algorithm to any previously used state.

We will define two similar games: an augmented and a non-augmented (or regular) state reset detection game. These two games differ only in that, in the augmented game, $\mathcal{U}$ is given the embedded key ek. Hence, the augmented game is intended primarily for asymmetric ASAs, and the non-augmented game is intended for symmetric ASAs (this is not an absolute requirement, and we will consider regular detectability of asymmetric ASAs in Section 5). This captures the difference between, for example, a nation state reverse-engineering one implementation to recover the embedded key, and a casual end-user doing black-box detection. The state reset detection games ($\mathrm{ASRDET}_{\mathrm{Sub}}(\mathcal{U})$ and $\mathrm{SRDET}_{\mathrm{Sub}}(\mathcal{U})$ respectively) are given in Figure 2, for some cryptographic scheme Λ and ASA Sub. These are distinguishability games, where $\mathcal{U}$ is asked to determine whether the oracle it is using is returning values according to the subverted algorithm $\mathsf{Sub.Alg}_\lambda$ or the original algorithm $\Lambda.\mathsf{Alg}_\lambda$. All state variables used by the subverted algorithm are saved between oracle calls, and $\mathcal{U}$ has access to an oracle Reset which allows it to reset state to any previously saved state (but does not give $\mathcal{U}$ the contents of the state).

For an adversary $\mathcal{U}$ in the state reset detectability games, we define the advantages of $\mathcal{U}$ as $\mathrm{Adv}^{\mathrm{SRDET}}_{\mathsf{Sub}}(\mathcal{U}) = |\Pr[\mathrm{SRDET}_{\mathsf{Sub}}(\mathcal{U})] - \frac{1}{2}|$ and $\mathrm{Adv}^{\mathrm{ASRDET}}_{\mathsf{Sub}}(\mathcal{U}) = |\Pr[\mathrm{ASRDET}_{\mathsf{Sub}}(\mathcal{U})] -$

$\boxed{\text{A}}\text{SRDET}_{\mathsf{Sub}}(\mathcal{U})$

1. $(xk, ek) \leftarrow\$ \mathsf{Sub.KeyGen}()$
2. $i \leftarrow 1$
3. $\tau_0 \leftarrow \perp$
4. $b \leftarrow\$ \{0,1\}$
5. $b' \leftarrow\$ \mathcal{U}^{\mathcal{O}_{\mathsf{Alg}_\lambda}, \boxed{\mathsf{Reset}}}(ek)$
6. **return** $b = b'$

$\mathsf{Reset}(j),\ 0 \le j < i$

1. **if** $b = 1$ **then**
2. $\quad \tau_i \leftarrow \tau_j$
3. $\quad i \leftarrow i + 1$

$\mathcal{O}_{\mathsf{Alg}_\lambda}(x)$

1. **if** $b = 0$ **then**
2. $\quad y \leftarrow\$ \Lambda.\mathsf{Alg}_\lambda(x)$
3. **if** $b = 1$ **then**
4. $\quad (y, \tau_i) \leftarrow\$ \mathsf{Sub.Alg}_\lambda(x, ek, \tau_{i-1})$
5. $\quad i \leftarrow i + 1$
6. **return** y

Figure 2: The augmented and non-augmented state reset detection games. The augmented game ASRDET includes the code in the box; the non-augmented one SRDET does not.

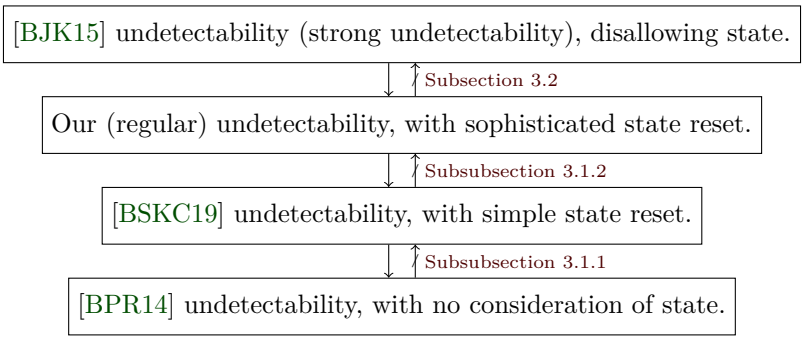

Figure 3: Relationships between detectability games in other works and ours. Arrows indicate that if an ASA is undetectable in the game at the tail of the arrow, then it is also undetectable in the game at the head of the arrow. Crossed arrows indicate that there exists an ASA undetectable in the game at the tail of the arrow but detectable at the head of the arrow.

$\frac{1}{2}|$. Informally, we say that Sub is undetectable if the corresponding advantage is small for any efficient adversary $\mathcal{U}$. Otherwise, it is detectable, and an adversary $\mathcal{U}$ with large advantage represents a strategy for detection.

It is worth taking some time to compare our new detectability game with those in previous works. The detectability game in [BPR14] did not include state resets, and fully allowed stateful ASAs. [BJK15] considered all stateful ASAs detectable, and formalized this by providing the state directly to $\mathcal{U}$ (the adversary in the detectability game), hence any non-$\perp$ state would lead to detection. They called this "strong undetectability". We also include the definition from [BSKC19] in this comparison, since, to our knowledge, they are the only other authors to include a state reset oracle in their detection analysis. Their state reset oracle only resets the state variables to their initial values, and not to any previously used values. A hierarchy here is clear, and we illustrate this in Figure 3. The implications given can be seen by simply noting that with each game higher in the hierarchy, the adversary $\mathcal{U}$ in the detectability game is given more capabilities with respect to manipulation and knowledge of the state of the ASA.

In fact, in the rest of this section, we will see that for each implication in Figure 3, there is a separation, meaning no two definitions are equivalent. First, in Subsubsection 3.1.1, we will use a simple state reset to detect an ASA that is undetectable in the [BPR14] model. In Subsubsection 3.1.2, we will use our more sophisticated state reset to detect an ASA that is undetectable even with simple state resets. In Subsection 3.2, we will show

that the original ASA of [BPR14], while detectable in the game of [BJK15] due to the use of state, remains undetectable in our game. Note that these are not artificial constructions, but rather existing ASAs, which are illustrating the significant differences between the definitions in Figure 3.

3.1 Detection of ASAs using state reset

In this section we will look at published ASAs against different cryptographic primitives and see that they are detectable using our notion of state reset detectability. This will demonstrate that the addition of our state reset oracle does indeed make our notion of detectability stronger than the basic definition used by [BPR14]. When we examine the result from [BSKC19], we will see that our state reset oracle also places further restrictions on ASAs wishing to achieve undetectability than their simple state reset oracle, which only resets state to a null value.

3.1.1 Detecting Ateniese, Magri, and Venturi's ASA using simple state reset

Ateniese, Magri, and Venturi [AMV15] describe two different symmetric algorithm substitution attacks on signature schemes. The first is virtually identical to the attack described by [BJK15]. The second is an attack on *coin-extractable schemes* (schemes for which the randomness used to generate the signature can be efficiently extracted from the signature itself). It works on any such scheme that makes use of at least a single bit of randomness.

Their attack on coin-extractable schemes works by having the subverted algorithm maintain the state of a stateful pseudorandom generator. Under our definition of state reset (or in fact even the simpler kind, resetting state to null values), their attack becomes detectable: in their ASA, re-use of state of the pseudorandom generator leads to re-use of the signature.

We first define some notation for signature schemes. A signature scheme SIG is composed of three algorithms: $\mathsf{SIG.KeyGen}$, $\mathsf{SIG.Sign}$, and $\mathsf{SIG.Ver}$. $\mathsf{SIG.KeyGen}$ randomly selects a secret private key sk and a public verification key pk as a pair from the key space $\mathcal{K}_{\mathsf{SIG}}$. $\mathsf{SIG.Sign}$ is a randomized algorithm with coins $r \in \{0,1\}^{\mathsf{SIG.rlen}}$. It takes a private key and a message $m \in \mathcal{M}_{\mathsf{SIG}}$ and produces a signature $s \in \mathcal{S}_{\mathsf{SIG}}$. $\mathsf{SIG.Ver}$ is a deterministic algorithm, taking a public key, a signature, and a message, and returning a boolean value indicating whether the signature passes verification.

Let G be a stateful pseudo-random generator with output length $d = \mathsf{SIG.rlen}$, i.e. it has input of some state t and outputs a pseudorandom output v of length d and new state t'. Assume for simplicity that d divides $|sk|$. The subversion of [AMV15] is shown in Figure 4a. On each execution of $\mathsf{Sub.Sign}$, this ASA encrypts the next d bits of the signing key, denoted by $sk[\ell+1, \ell+d]$, using G as a stream cipher. This encryption is then used in place of the coins for the signature. Since the subverter can get the coins from the signature and knows the embedded key, they can recover the signing key by decrypting the recovered coins. We use the regular detectability game from Figure 2 with SIG as Λ and $\mathsf{SIG.Sign}$ as $\Lambda.\mathsf{Alg}_\lambda$ to reason about the detectability of this ASA.

Detectability under SRDET can be seen as follows: The detector first calls the signing oracle once with some message m and signing key sk, and the state is then set to $\tau_1 = (\mathsf{G}(\bar{k}), d)$. The detector then calls the reset oracle with $j = 0$, and τ_2 is set to $\tau_0 = \bot$. On a second oracle call with the message m and signing key sk, τ_3 is set to $(\mathsf{G}(\bar{k}), d)$ as before. Let s_1 and s_2 be the two signatures received. Note that the same v and ℓ values were used, and hence the same $\tilde{r}$ value was used, to generate both signatures. Hence the detector will observe that $s_1 = s_2$ with certainty, where this would only be the case some small fraction of the time for an unsubverted scheme, yielding large detection advantage.

$\mathsf{Sub.Sign}(sk, m, \bar{k}, \tau)$

1. **if** $\tau = \bot$ **then**
2. $\quad \tau \leftarrow (\bar{k}, 0)$
3. $(t, \ell) \leftarrow \tau$
4. **if** $\ell \geq |sk|$ **then**
5. $\quad \ell \leftarrow 0$
6. $(v, t) \leftarrow \mathsf{G}(t)$
7. $\tilde{r} \leftarrow v \oplus sk[\ell + 1, \ell + d]$
8. $\tau \leftarrow (t, \ell + d)$
9. $s \leftarrow \mathsf{SIG.Sign}(sk, m; \tilde{r})$
10. **return** (s, τ)

(a) ASA on signature schemes, by Ateniese et al. [AMV15]

$\mathsf{Sub.Sign}(x, m, \bar{k}, \tau)$

1. **if** $\tau = \bot$ **then**
2. $\quad \tau \leftarrow (0, \bot)$
3. $(j, \sigma) \leftarrow \tau$
4. **if** $j = 0 \bmod 2$ **then**
5. $\quad \kappa \leftarrow\$ \mathbb{Z}_q$
6. $\quad r \leftarrow \mathsf{H}_1(g^{\kappa})$
7. $\quad s \leftarrow \kappa^{-1}(\mathsf{H}_2(m) + xr) \bmod q$
8. **else**
9. $\quad \tilde{\kappa} \leftarrow \mathsf{H}_3(\bar{k}, \sigma)$
10. $\quad r \leftarrow \mathsf{H}_1(g^{\tilde{\kappa}})$
11. $\quad s \leftarrow \tilde{\kappa}^{-1}(\mathsf{H}_2(m) + xr) \bmod q$
12. $\tau \leftarrow (j + 1, r)$
13. **return** $((s, r), \tau)$

(b) ASA on DSA, by Baek et al. [BSKC19]

$\mathsf{Sub.Encaps}(ek, pk, \tau)$

1. **if** $\tau = \bot$ **then**
2. $\quad \tau \leftarrow\$ \mathcal{R}_{\mathsf{KEM}}$
3. **else**
4. $\quad t \leftarrow \mathsf{KEM.kgen}(ek, \tau)$
5. $\quad \tau \leftarrow \mathsf{F}(ek, t)$
6. $k \leftarrow \mathsf{KEM.kgen}(pk, \tau)$
7. $c \leftarrow \mathsf{KEM.cgen}(\tau)$
8. **return** (c, τ)

(c) ASA on KEMs, by Chen et al. [CHY20]

Figure 4: ASAs for analysis in Section 3

3.1.2 Detecting Baek, Susilo, Kim, and Chow's ASA using sophisticated state reset

Baek, Susilo, Kim, and Chow [BSKC19] describe a symmetric ASA against the Digital Signature Algorithm (DSA). Using what they call a small amount of state, they are able to recover the signing key from only 3 subverted signatures. In their paper, they even consider state resets in their formalism of undetectability (and appear to be the first to do so). However, their state resets only set the state back to a null value, and not any previously used state. We show that under our stronger definition allowing resets to any previous state, their subversion is easily detectable despite the small amount of state kept.

We again use the regular detectability game SRDET given in Figure 2, with SIG as Λ and $\mathsf{SIG.Sign}$ as $\Lambda.\mathsf{Alg}_\lambda$, but specify a few more details about the signature scheme SIG, since the ASA from [BSKC19] is specific to DSA signatures. Let H_1 and H_2 be cryptographic hash functions. Let $\mathbb{G}$ be a cyclic group of prime order q, and let g be a generator of that group. We define x to be the signing key and $y = g^x$ be the verification key. The algorithm $\mathsf{SIG.Sign}$ will first sample $\kappa \leftarrow\$ \mathbb{Z}_q$, and then return $(r, s) \leftarrow (\mathsf{H}_1(g^{\kappa}), \kappa^{-1}(\mathsf{H}_2(m) + xr) \bmod q)$.

Let H_3 be a PRF. The ASA from [BSKC19] is shown in Figure 4b. The key idea here is that the signing algorithm will only subvert one out of every two signatures. The signatures are subverted by controlling the way the per-signature randomness κ is generated. A signature is subverted by making the randomness used in a signature dependent on the randomness used in the previous signature, in a way that can be reverse-engineered by the subverter.

Under a state reset where the state is set to initial values, j is reset to 0 and the value of σ is cleared. Then the next signature generated is always an unsubverted one. This proper sampling of randomness leads to undetectability in this case, and indeed [BSKC19] show this. However, if a detector is able to reset state to any previous value, this no longer holds, since all later signatures after the first are deterministically generated based on previous state. Observe the following attack on undetectability. The detector first calls the signing oracle twice with some message m and signing key x, and the state is set to $\tau_1 = (1, \mathsf{H}_1(g^{\kappa}))$ on the first call, for some randomly chosen κ. The detector then calls the reset oracle with $j = 1$ so that the next state τ_3 is set to prior state $\tau_1 = (1, \mathsf{H}_1(g^{\kappa}))$. The detector then makes a third signing oracle call with the same message m and signing key x. Let s_2 and s_3 be the s-values of the second and third signatures received, respectively. The same value of $\tilde{\kappa}$ was used to generate both these signatures, so the detector will observe that $s_2 = s_3$ with certainty, whereas this would be very unlikely in an unsubverted scheme. Therefore after observing only 3 signatures, and using one state reset to a prior state, the

detector's detection advantage is extremely close to 1.

3.1.3 Detecting Chen, Huang, and Yung's ASA using state resets

Chen, Huang, and Yung [CHY20] describe an asymmetric ASA against a key encapsulation mechanism (KEM) which is stateful and recovers the encapsulated key using only two consecutive encapsulation ciphertexts. Their subversion works on KEMs that can be decomposed into specific sub-functions, most notably requiring that generation of the ciphertext does not require the public encapsulation key, only the coins used to generate the shared secret key. Furthermore, their attack is asymmetric, meaning it is undetectable (under their definition) even if the key embedded into the subversion is known to the detector. The subverter makes use of a corresponding private extraction key in order to exploit the subversion.

A key encapsulation scheme KEM is composed of three algorithms: $\mathsf{KEM.KeyGen}$, $\mathsf{KEM.Encaps}$, and $\mathsf{KEM.Decaps}$. $\mathsf{KEM.KeyGen}$ randomly generates a secret decapsulation key sk and a public encapsulation key pk. $\mathsf{KEM.Encaps}$ is a randomized algorithm with coins $r \in \mathcal{R}_{\mathsf{KEM}}$. It takes a public key and produces a ciphertext $c \in \mathcal{C}_{\mathsf{KEM}}$, and a session key $k \in \mathcal{K}_{\mathsf{KEM}}$. For the ASA from [CHY20], we require that it decomposes into three components:

1. $r \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{KEM}}$.
2. $\mathsf{KEM.kgen}$, which takes as input public key pk and randomness r, is used to generate key $k \in \mathcal{K}_{\mathsf{KEM}}$.
3. $\mathsf{KEM.cgen}$, which takes only the randomness r, outputs ciphertext $c \in \mathcal{C}_{\mathsf{KEM}}$.

Finally, $\mathsf{KEM.Decaps}$ is a deterministic algorithm, takes a private key and a ciphertext, and returns a session key k or an error.

Let F be a PRF which takes the embedded key ek and a value in $\mathcal{R}_{\mathsf{KEM}}$ and returns a value in $\mathcal{R}_{\mathsf{KEM}}$. The ASA from [CHY20] is given in Figure 4c. This ASA can be used to recover the established shared key k_i, $i > 1$, using the two consecutive ciphertexts c_{i-1} and c_i: since c_i was not the first ciphertext sent, it was generated using subverted randomness τ_i. Note that because ciphertext generation does not depend on the public encapsulation key used, the same ciphertext c_{i-1} is generated for k_{i-1} with the legitimate encapsulation key and for t_{i-1} with the subverter's embedded key ek. Hence t_{i-1} can be obtained by decapsulating c_{i-1} using xk: $t_{i-1} \leftarrow \mathsf{KEM.Decaps}(xk, c_{i-1})$. Then $\tau_i \leftarrow \mathsf{F}(ek, t_{i-1})$. This allows one to compute $k_i \leftarrow \mathsf{KEM.kgen}(pk, \tau_i)$, the shared key corresponding to the ciphertext c_i.

The detection game we use is ASRDET from Figure 2 with KEM as Λ and $\mathsf{KEM.Encaps}$ as $\Lambda.\mathsf{Alg}_\lambda$. This subversion is detectable under our definition. Observe the following attack on undetectability. The detector first calls the encapsulation oracle twice with some encapsulation key pk, and the state is set to $\tau_1 = \tau$ on the first call, for some randomly chosen τ. The detector then calls the reset oracle with $j = 1$, and τ_3 is set to $\tau_3 = \tau_1 = \tau$. The detector then makes a third encapsulation oracle call with the encapsulation key pk. Let c_2 and c_3 be the ciphertexts received from the second and third encapsulation oracle calls respectively. Note that the same value of τ was used to generate both ciphertexts. Hence the detector will observe that $c_2 = c_3$ with certainty, whereas this would be very unlikely in an unsubverted scheme. Thus, after observing only 3 ciphertexts, and using one state reset to a prior state, the detector's detection advantage is extremely close to 1.

Note that the detection methods for the ASA from [BSKC19] and the ASA from [CHY20] are very similar. Both of these papers purported to have "small" state, which should not be considered unreasonable in practical contexts. However, very simple state resets, as could happen even accidentally with virtual machine images, will result in guaranteed or very likely repetition of output, which is catastrophic for detection.

3.2 Undetectability of Bellare, Paterson, and Rogaway's ASA

Contrary to the results we've seen so far in this section, the original "biased ciphertext attack" ASA on symmetric encryption by [BPR14] is still undetectable in our new framework with state resets (specifically, SRDET). In fact, the majority of the proof provided by [BJK15] of the undetectability of their ASA applies directly to the ASA of [BPR14], even in the presence of the state reset oracle.

The subverted encryption algorithm used by [BPR14] is equivalent to the one given in Figure 5a, where F is a PRF which takes a key and a ciphertext of $\mathsf{SE.Enc}$ and returns a single bit, and s is a predetermined parameter of the subversion which bounds the number of loops the ASA will execute before returning a value. This description looks very similar to the ASA given in [BJK15]; indeed, we have essentially taken their subversion, and re-included the index-as-state technique that they removed when presenting their improvement over the ASA from [BPR14].

Theorem 1. *Let $\mathcal{U}$ be an adversary in the regular state reset detectability game in Figure 2, $\mathrm{SRDET}_{\mathsf{Sub}}(\mathcal{U})$, with symmetric encryption scheme SE as Λ and $\mathsf{SE.Enc}$ as $\Lambda.\mathsf{Alg}_\lambda$, where $\mathsf{Sub.Enc}$ is the algorithm given in Figure 5a. If n is the number of queries that $\mathcal{U}$ makes to its encryption oracle and η is the min-entropy of $\mathsf{SE.Enc}$, then there is an adversary $\mathcal{F}$ in the $\mathrm{PRF}_{\mathsf{F}}(\mathcal{F})$ game such that $\mathrm{Adv}^{\mathrm{SRDET}}_{\mathsf{Sub}}(\mathcal{U}) \leq 2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F}) + n^2 s^2 \cdot 2^{-\eta-1}$. The running time of $\mathcal{F}$ is about that of $\mathcal{U}$, and $\mathcal{F}$ makes at most ns oracle queries.*

Proof. As much of this proof is the same as that given by [BJK15], we will only include some details. We proceed by a sequence of games. Let H_0 be the regular state reset detectability game of Figure 2 with the appropriate substitutions mentioned in the theorem statement. Let H_1 be the same as H_0 but with F replaced by a lazily-sampled random function of c. Let H_2 be the same as H_1 but with the lazily-sampled random function replaced by fully random sampling of w. Let H_3 be the regular state reset detectability game where the encryption oracle simply returns $c \leftarrow\!\!\$\ \mathsf{Enc}(k, m)$, and the Reset oracle has no effect.

The first game change is standard, and proceeds exactly as described by [BJK15], with $|\Pr[H_1] - \Pr[H_0]| = 2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F})$ for an adversary $\mathcal{F}$. The second game change also proceeds identically to the description given by [BJK15]: in order to replace the lazily sampled random function with true random sampling, we must bound the probability that some value of c is repeated during the game. Call this event P. Since the number of loops for each oracle call is bounded by s, the probability of P occurring is therefore bounded by $\binom{ns}{2} \cdot 2^{-\eta} \leq n^2 s^2 \cdot 2^{-\eta-1}$, where n is the number of queries to the oracle and η is the min-entropy of $\mathsf{SE.Enc}$. Thus we have $|\Pr[H_2] - \Pr[H_1]| \leq n^2 s^2 \cdot 2^{-\eta-1}$.

Now we have game H_2, shown in Figure 5b. We argue that H_2 is equivalent to H_3. In particular, we argue that the implementation of $\mathsf{Sub.Enc}$ in the encryption oracle of H_2 is identical to $\mathsf{SE.Enc}$. To see this, note that, despite any runs of the Reset oracle, the value of $k[\tau]$ is fixed at the start of an oracle call. Since w is sampled randomly, the decision of which c to return is independent of the state τ, and is in fact the same as simply sampling coins r and returning the resulting ciphertext c. This is precisely $\mathsf{SE.Enc}$. Therefore $\Pr[H_3] = \Pr[H_2]$.

In H_3, since the oracle behaviour is independent of b, we have that $\Pr[H_3] = \frac{1}{2}$. Putting together all these results, we have

$$\mathrm{Adv}^{\mathrm{SRDET}}_{\mathsf{Sub}}(\mathcal{U}) \leq 2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F}) + n^2 s^2 \cdot 2^{-\eta-1},$$

as desired. □

The number of queries n is polynomially bounded, and $2^{-\eta}$ is negligible for most randomized schemes. The value of s can be set to a small constant without a strong effect on the success of the ASA, and we assume that $2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F})$ is small for a good PRF F.

$\mathsf{Sub.Enc}(k, m, \bar{k}, \tau)$

1. **if** $\tau = \perp$ **then** $\tau \leftarrow 0$
2. **else** $\tau \leftarrow \tau + 1 \bmod |k|$
3. $j \leftarrow 0$
4. **do**
5. $\quad j \leftarrow j + 1$
6. $\quad r \leftarrow\$ \{0,1\}^{\mathsf{SE.rlen}}$
7. $\quad c \leftarrow \mathsf{Enc}(k, m; r)$
8. $\quad w \leftarrow \mathsf{F}(\bar{k}, c)$
9. **until** $k[\tau] = w$ **or** $j = s$
10. **return** (c, τ)

(a) ASA of [BPR14], where s is a predetermined parameter.

H_2

1. $\bar{k} \leftarrow\$ \mathsf{Sub.KeyGen}()$
2. $i \leftarrow 1$
3. $\tau_0 \leftarrow \perp$
4. $b \leftarrow\$ \{0,1\}$
5. $b' \leftarrow\$ \mathcal{U}^{\mathcal{O}_{\mathsf{Enc}}, \mathsf{Reset}}$
6. **return** $b = b'$

$\mathsf{Reset}(j),\ 0 \le j < i$

1. **if** $b = 1$ **then**
2. $\quad \tau_i \leftarrow \tau_j$
3. $\quad i \leftarrow i + 1$

$\mathcal{O}_{\mathsf{Enc}}(k, m)$

1. **if** $b = 0$ **then** $c \leftarrow\$ \mathsf{Enc}(k, m)$
2. **if** $b = 1$ **then**
3. $\quad$ **if** $\tau = \perp$ **then** $\tau = 0$
4. $\quad$ **else** $\tau \leftarrow \tau + 1 \bmod |k|$
5. $\quad j \leftarrow 0$
6. $\quad$ **do**
7. $\quad\quad j \leftarrow j + 1$
8. $\quad\quad r \leftarrow\$ \{0,1\}^{\mathsf{SE.rlen}}$
9. $\quad\quad c \leftarrow \mathsf{SE.Enc}(k, m; r)$
10. $\quad\quad w \leftarrow\$ \{0,1\}$
11. $\quad$ **until** $k[\tau] = w$ **or** $j = s$
12. $\quad \tau_i \leftarrow \tau; i \leftarrow i + 1$
13. **return** c

(b) The game H_2 for the proof of Theorem 1.

Figure 5: ASA on symmetric encryption by Bellare, Paterson, and Rogaway [BPR14] and game for proof of its undetectability in Theorem 1.

Hence we can conclude from Theorem 1 that the ASA defined in Figure 5a is undetectable under state resets, even to any prior state.

3.3 Discussion

The reader may find the results in this section to not be technically deep, and indeed they would be correct. We included significant details nonetheless in order to demonstrate precisely the implications of our model. Firstly, the simplicity of the state reset detection attacks in Subsection 3.1 raise the question of why these attacks were not considered in a formal manner previously in the literature, despite being pointed out as early as [BJK15]. Secondly, the similarity of the proof in Subsection 3.2 to the proofs of [BJK15] raises the question of why the norm of stateful schemes being considered less desirable was adopted so readily.

Perhaps there is reason not to consider such a strong notion of state reset in certain circumstances. However, the above results do show a couple things conclusively. Firstly, for researchers who avoid or discount stateful schemes, it should be made clear what detection threat model they are working in. Secondly, for researchers who develop stateful schemes, undetectability should be proven in a formal model including some version of state reset, or detection methods in such a framework should be acknowledged. As we have previously mentioned, we believe our notion of state reset is a good choice for analysis, as it formalizes the kind of resets that can occur during virtual machine cloning and rebooting, but weaker models might be justified depending on the threat model.

4 A Type 1 Asymmetric ASA on Symmetric Encryption

Now that we have established a good framework from which to evaluate the undetectability of stateful ASAs, we will present a simple modification to the subversion from [BPR14] to get a type 1 asymmetric ASA on a symmetric encryption scheme. Recall that a type 1 asymmetric ASA must be undetectable in the augmented state reset detectability game ASRDET of Figure 3.

In order to construct an asymmetric ASA which is undetectable against an adversary with the embedded key, we will use an additional building block: public-key encryption with ciphertexts that are indistinguishable from random. We recall the notion of ciphertext indistinguishability from random bits for public-key encryption schemes: let PKE be a public-key

$\text{IND\$}_{\mathsf{PKE}}(\mathcal{B})$

1. $(pk, sk) \leftarrow\$ \mathsf{PKE.KeyGen}()$
2. $b \leftarrow\$ \{0,1\}$
3. $b' \leftarrow\$ \mathcal{B}^{\mathcal{O}_{\mathsf{PKE.Enc}}}(pk)$
4. **return** $b = b'$

$\mathcal{O}_{\mathsf{PKE.Enc}}(m)$

1. **if** $b = 0$ **then**
2. $\quad c \leftarrow\$ \mathsf{PKE.Enc}(pk, m)$
3. **if** $b = 1$ **then**
4. $\quad c \leftarrow\$ \{0,1\}^{\mathsf{PKE.clen}}$
5. **return** c

Figure 6: The ciphertext indistinguishability-from-random game for a public-key encryption scheme PKE.

encryption scheme, and consider the game in Figure 6. In this game, adversary $\mathcal{B}$ is tasked with deciding whether the oracle provided to it is returning encryptions under PKE.Enc or random bits. The advantage of $\mathcal{B}$ is defined as $\text{Adv}^{\text{IND\$}}_{\mathsf{PKE}}(\mathcal{B}) = |\Pr[\text{IND\$}_{\mathsf{PKE}}(\mathcal{B})] - \frac{1}{2}|$. Informally, we say that the scheme PKE is IND\$-secure if the advantage of any efficient adversary $\mathcal{B}$ is small.

We present our asymmetric ASA, ASub, against a symmetric encryption scheme SE, in Figure 7. This ASA uses an IND\$-secure public-key encryption scheme PKE, and a parameter s to bound the number of loops the ASA will execute before returning a value. The essence of the subversion is that the secret key to be exfiltrated is encrypted using the public-key encryption scheme, then a technique similar to that used by [BPR14] is used to leak the resulting ciphertext. The subverter can recover the key by decrypting the extracted ciphertext.

$\mathsf{ASub.Enc}(k, m, ek, \tau)$

1. **if** $\tau = \perp$ **then**
2. $\quad \sigma \leftarrow 0; \kappa \leftarrow\$ \mathsf{PKE.Enc}(ek, k)$
3. **else** $(\sigma, \kappa) \leftarrow \tau$
4. **if** $\sigma = \mathsf{PKE.clen}$ **then**
5. $\quad \sigma \leftarrow 1; \kappa \leftarrow\$ \mathsf{PKE.Enc}(ek, k)$
6. **else** $\sigma \leftarrow \sigma + 1$
7. $j \leftarrow 0$
8. **do**
9. $\quad j \leftarrow j + 1$
10. $\quad r \leftarrow\$ \{0,1\}^{\mathsf{SE.rlen}}$
11. $\quad c \leftarrow \mathsf{SE.Enc}(k, m; r)$
12. $\quad w \leftarrow \mathsf{F}(ek, c)$
13. **until** $\kappa[\sigma] = w$ **or** $j = s$
14. $\tau \leftarrow (\sigma, \kappa)$
15. **return** (c, τ)

Figure 7: Our type 1 asymmetric ASA on symmetric encryption.

In practice, the main advantage to a type 1 asymmetric ASA lies in the fact that the subverter maintains possession of all information required to complete their attack. This differs from the symmetric case, where the same key used to extract values from the target user is also embedded in the algorithm that the target user is using. There are practical situations in which it may be relevant to consider the possibility of a detector who knows ek. For example, suppose the user is aware of other subverted implementations, for which the corresponding ek is known, and wishes to test if the implementation they are using has also been subverted with the same key. In this section, we will consider the user $\mathcal{U}$ as being the detector with knowledge of the key ek. Indeed, if an ASA is undetectable in our augmented detection game ASRDET, then neither the user nor anyone else without the key xk is able to detect the ASA, and hence no third party is able to exploit the ASA either. In Section 5, we will consider a type 2 asymmetric ASA, which is undetectable to a user without ek, and detectable to, but nonetheless still secure against, a third party with knowledge of ek. This will more fully explore the nuance associated with the benefits of an asymmetric ASA with respect to all parties who may possibly be involved.

The main drawback of an asymmetric ASA, especially when it comes to attacking symmetric schemes, is speed. An asymmetric ASA that makes use of asymmetric cryptography will inevitably be slower than the symmetric algorithms being subverted. This exacerbates an existing issue with many ASAs that rely on coin rejection sampling, including ours: since the algorithm being subverted must be run multiple times, a detector could time the execution of the algorithm and conclude that a slower algorithm is subverted (this side-channel attack is not captured in our framework). We do note, however, that our ASA uses far fewer executions of asymmetric algorithms than symmetric ones, and moreover

the asymmetric executions can be done ahead of time (but must be after the algorithm substitution has occurred). One could imagine a clever implementation of our ASA where the evaluation of PKE.Enc is spread out over many calls to ASub.Enc, amortizing the time penalty added by the use of a public-key encryption scheme.

4.1 Undetectability of our type 1 asymmetric ASA

The following theorem shows that ASub of Figure 7 is undetectable in the augmented state reset detectability game ASRDET, when modeling F as a random oracle.

Theorem 2. *Let $\mathcal{U}$ be an adversary in the augmented state reset detectability game in Figure 2,* $\text{ASRDET}_{\mathsf{ASub}}(\mathcal{U})$*, with symmetric encryption scheme* SE *as* Λ *and* SE.Enc *as* $\Lambda.\mathsf{Alg}_\lambda$*, where* ASub.Enc *is the algorithm given in Figure 7. Assume that* F *is an ideal hash function, which we model as a random oracle* H*. If n, q are the number of queries that $\mathcal{U}$ makes to its encryption oracle and the random oracle respectively, and η is the min-entropy of* SE.Enc*, then there is an adversary $\mathcal{B}$ against the* IND$*-security of* PKE *such that* $\text{Adv}^{\text{ASRDET}}_{\mathsf{ASub}}(\mathcal{U}) \leq 2\text{Adv}^{\text{IND\$}}_{\mathsf{PKE}}(\mathcal{B}) + ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1}$*. The running time of $\mathcal{B}$ is about that of $\mathcal{U}$ and $\mathcal{B}$ makes n queries to its own encryption oracle.*

Proof. Consider the augmented state reset detectability game of Figure 2, with all the substitutions in the theorem statement, and F modeled as the random oracle H provided to the adversary $\mathcal{U}$. This is shown in Figure 8, as G_0. The oracle H only takes input c, since ek is fixed throughout the game.

We proceed by a sequence of games $G_0, ..., G_4$, as shown in Figure 8. Let G_0 be the undetectability game with the random oracle implementation. G_1 is the same as G_0 but with κ sampled randomly instead of computed as an encryption of k. G_2 and G_3 are shown in Figure 8. G_4 is the augmented state reset detectability game where the encryption oracle is replaced by an oracle that simply returns $\mathsf{SE.Enc}(k, m)$.

Let $\mathcal{B}$ be an adversary to the IND$ game for PKE. The adversary $\mathcal{B}$ will act as a challenger in the game G_0 with a few small changes: it will not generate a key pair itself, instead using the public key pk provided to it in the IND$ game; it will return 1 if $\mathcal{U}$ correctly guesses the bit b and 0 otherwise; and it will use its provided $\mathcal{O}_{\mathsf{PKE.Enc}}$ oracle in the place of PKE.Enc. This can all be done efficiently. In this way, $\mathcal{B}$ interpolates between games G_0 and G_1. In particular, let $b_\$$ denote the bit from the $\text{IND\$}_{\mathsf{PKE}}(\mathcal{B})$ game. Note that if $b_\$ = 1$, then the Enc oracle simulated by $\mathcal{B}$ proceeds exactly as in game G_1, and if $b_\$ = 0$, then it proceeds exactly as in game G_0. Thus we have $\Pr[\mathcal{B} \Rightarrow 1 \mid b_\$ = 1] = \Pr[G_1]$, $\Pr[\mathcal{B} \Rightarrow 1 \mid b_\$ = 0] = \Pr[G_0]$, and hence $|\Pr[G_1] - \Pr[G_0]| = 2\text{Adv}^{\text{IND\$}}_{\mathsf{PKE}}(\mathcal{B})$.

Next, consider G_2 shown in Figure 8. We claim that $\Pr[G_2] = \Pr[G_1]$. To see this, note that lines 1–10 of the Helper_1 function simply act as bookkeeping in order to use a single bit of κ for each encryption, and to generate a new κ once we've iterated through its bits. The index σ is used to iterate through κ and κ is re-sampled when all the bits are used. This procedure is identical to sampling a single random bit for each call, hence our equality. Notice also that this removes any dependence on the state, and so the state reset oracle no longer has any function. The only other change is the addition of a bad variable, used in the next game transition.

In game G_3, shown in Figure 8, we replace the selection of w in the encryption oracle with true random sampling of w, regardless of whether c was input to the random oracle before. Let Col be the event where bad is set to true in game G_2. This happens when some c previously generated by the encryption oracle or previously queried in the random oracle is obtained again during an encryption oracle query. We can bound $\Pr[\mathsf{Col}]$ from above by considering the case where all the random oracle queries happen before the encryption oracle queries. Let η be the min-entropy of SE.Enc. Then we have that $\Pr[\mathsf{Col}] \leq \left(\binom{ns+q}{2} - \binom{q}{2}\right) \cdot 2^{-\eta} = ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1}$.

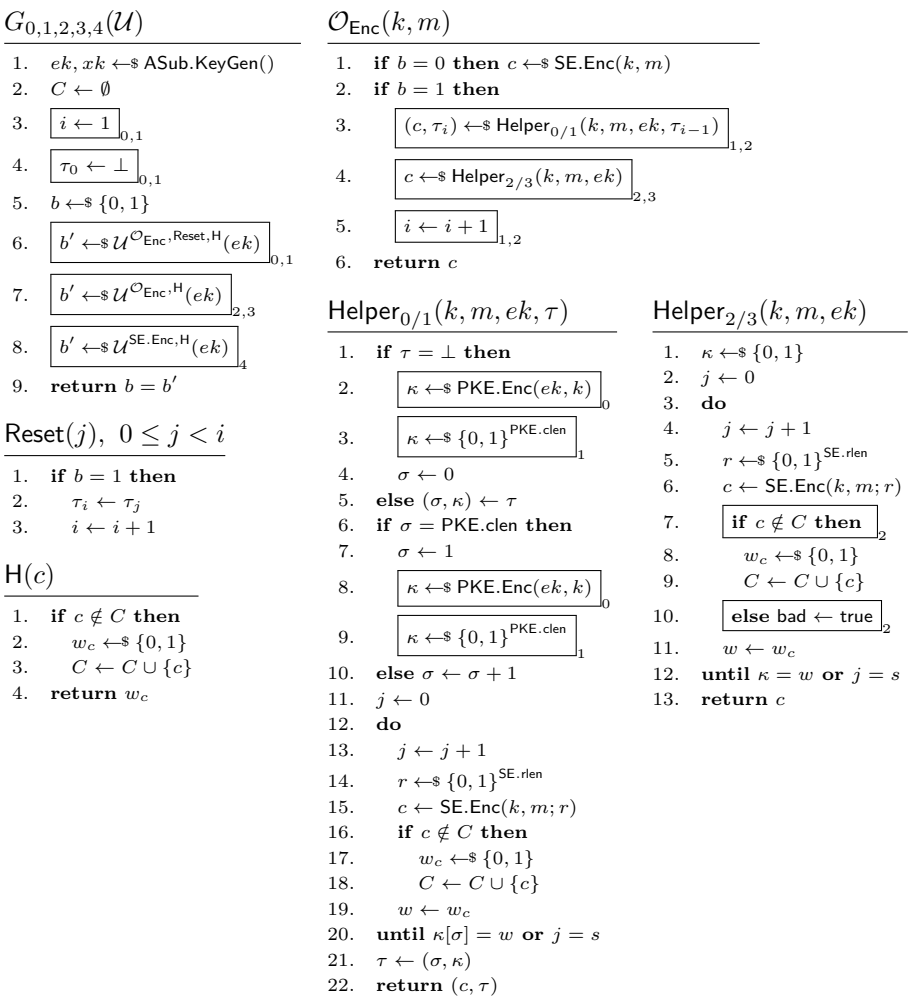

Figure 8: Games G_0 through G_4 for the proof of Theorem 2. For boxed code, only the games indicated in the subscripts contain that code.

By the Fundamental Lemma of Game-Playing [BR06], we have $|\Pr[G_3] - \Pr[G_2]| \le \Pr[\mathsf{Col}] \le ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1}$.

Finally, G_4 is the detectability game where the encryption oracle is replaced by an oracle that simply returns $\mathsf{SE.Enc}(k, m)$. In game G_3, since the loop condition is no longer dependent on the selection of c, the Helper_3 is identical to $\mathsf{SE.Enc}(k, m)$. Hence $\Pr[G_3] = \Pr[G_4]$. Further, note that $\Pr[G_4] = \frac{1}{2}$, since the encryption oracle is not dependent on b.

Putting all these results together, we have

$$\mathrm{Adv}^{\mathrm{ASRDET}}_{\mathsf{ASub.Enc}}(\mathcal{U}) \le 2\mathrm{Adv}^{\mathrm{IND\$}}_{\mathsf{PKE}}(\mathcal{B}) + ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1},$$

as desired. □

Assuming that PKE is IND$-secure and that n and q are small in comparison to 2^η, which is the case for a sufficiently randomized encryption scheme, Theorem 2 shows that the ASA given by the subversion in Figure 7 is undetectable in the augmented state reset detection game ASRDET, proving our claim that ASub is a type 1 asymmetric ASA.

4.2 Key recovery of our type 1 asymmetric ASA

The goal of the ASA presented in Figure 7 is to recover the secret key k used in the encryption algorithm $\mathsf{SE.Enc}$. Some authors, such as [BJK15], have treated key recovery formally with a key recovery game. Their approach largely carries over here, and one could readily develop a theorem bounding from below the probability that the subverter can recover the secret key for our ASA. We will only outline the ideas involved to give the reader a good sense of how key recovery works.

Once the subverted algorithm is being used by a user, the subverter will attempt to recover the secret key k by observing the generated ciphertexts. These ciphertexts are generated according to a message distribution that the subverter has no control over, and hence the recovery strategy will be independent of the messages used. For our ASA, the subverter can use the following method to recover the secret key:

1. Collect ciphertexts $c_1, ..., c_n$.
2. Group ciphertexts together into consecutive groups of size $\mathsf{PKE.clen}$.
3. For each ciphertext c_σ in each group, compute $w_\sigma = \mathsf{F}(ek, c_\sigma)$.
4. For each resulting group of bits, concatenate all the w_σ to obtain κ and compute $k' = \mathsf{PKE.Dec}(xk, \kappa)$.
5. Each k' obtained in step 4 is a candidate for k.

We can estimate the probability of success of key recovery. We make a few simplifying assumptions; namely, that F is a good PRF, and that there is sufficient randomness in $\mathsf{SE.Enc}$ so that no two of the ciphertexts $c_1, ..., c_n$ are the same. The effect of these assumptions can be quantified[2], but we will omit those details here, and simply note that such quantification will result in $2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F}) + n^2 s^2 \cdot 2^{-\eta-1}$, where η is the min-entropy of $\mathsf{SE.Enc}$ and $\mathcal{F}$ is a PRF adversary, being subtracted from the probability of key recovery. These assumptions allow us to calculate the success probability when all the w values are randomly generated, which is much simpler.

Suppose that PKE is a δ-correct public-key encryption scheme. Let n be the total number of ciphertexts intercepted by the subverter. For each group of $\mathsf{PKE.clen}$ ciphertexts (of which there are $\lfloor n/\mathsf{PKE.clen} \rfloor$), the probability that every ciphertext c in the group was chosen to successfully leak a bit ($\kappa[\sigma] = \mathsf{F}(ek, c)$) is $(1 - 2^{-s})^{\mathsf{PKE.clen}}$ (this is where we assume each w is random), and the probability that the group decrypts correctly is δ. Hence the probability that one of the keys k' obtained in step 5 is the key k is $P_{kr} = 1 - \left(1 - \delta(1 - 2^{-s})^{\mathsf{PKE.clen}}\right)^{\lfloor \frac{n}{\mathsf{PKE.clen}} \rfloor}$. As an example, suppose $\mathsf{PKE.clen} = 400$ (an IND$-secure public-key scheme with ciphertext lengths around this is given by Möller [Möl04]), $n = 1600$, $s = 7$, and $\delta = 1$. Then $P_{kr} \geq 0.6$. The value of s can easily be increased to improve this probability, for example, if δ is lower, $\mathsf{PKE.clen}$ is higher, or the number of ciphertexts n that the subverter can intercept is small.

4.2.1 Key recovery in the presence of state resets

We note here that if the the user of the encryption scheme performs regular state resets of the type used in the detection scenario, then key recovery becomes very unlikely for this ASA. Since the entire PKE ciphertext κ must be exfiltrated before the subverter is able to decrypt it to recover k, reset of state would restart the process of key recovery. Since κ is required to be indistinguishable from random, we do not have the option of reconstructing the same κ after a state reset. With $\mathsf{PKE.clen} = 400$, a state reset after every 399 encryptions would successfully thwart this ASA.

[2] There are several examples of this in the literature, as well as in our proof of Theorem 2 in the previous subsection.

Such a defense may or may not be practical, depending on the specific implementation of the encryption scheme and the execution environment. For example, a user may not have sufficient access to an encryption scheme provided to them as a black-box to perform such state resets, and would rely on the provider of the encryption service to include such a mitigation. Furthermore, the performance impacts may be significant. Clearing sections of memory between encryptions may be a quick operation, but the ASA may decide to place state in storage instead (at increased risk of being detected from storage monitoring). Resetting to a previous virtual machine image would thwart this as well, but at increased performance cost. Such performance penalties may be mitigated by containerized implementations of cryptographic schemes. We encourage future work on the feasibility of this approach, and more generally on the use of state resets as a countermeasure to ASAs.

5 A Type 2 Asymmetric ASA on Symmetric Encryption

In the last section, we presented a type 1 asymmetric ASA, where no other parties besides the subverter, even if they have the embedded key ek, would be able to detect the subversion. In this section we will explore a more nuanced requirement of undetectability. In fact, we will present an asymmetric ASA that is *detectable* in the augmented detectability game in Figure 2, but undetectable in the regular detectability game. This is a type 2 asymmetric ASA.

To see why such an ASA may still be relevant, consider again the context of an ASA. There are three relevant parties: the subverter, the user $\mathcal{U}$, and some third party $\mathcal{V}$. The subverter is executing an algorithm substitution attack on $\mathcal{U}$. It is critically important that $\mathcal{U}$ is not able to detect the ASA, since then $\mathcal{U}$ will stop using the subverted scheme. The subverter relies on the fact the $\mathcal{U}$ is not able to examine the code being used. Hence the situations in which $\mathcal{U}$ knows the embedded key ek, but does not already know of the subversion, are limited. With this reasoning, we can restrict ourselves to evaluating the detectability of an ASA with respect to a user $\mathcal{U}$ only in the regular detectability game, regardless of whether or not the ASA is symmetric or asymmetric.

On the other hand, the requirements on the third party $\mathcal{V}$ are different, and for a type 2 asymmetric ASA we will allow for the possibility that a sophisticated third party $\mathcal{V}$ is able to reverse-engineer the cryptographic scheme and obtain the key ek or outright detect the scheme in this way. Such a $\mathcal{V}$ may not have decision-making authority to change the scheme being used, so it may not be catastrophic for $\mathcal{V}$ to be able to detect the ASA. Hence requiring augmented undetectability with respect to $\mathcal{V}$ is not necessary. The real requirement for a type 2 asymmetric ASA is that the subverter is the only one able to take advantage of the subversion and break the security of the subverted scheme. To reflect this, we can simply require that the subverted algorithm $\mathsf{ASub.Alg}_\lambda$ preserve security properties of the original algorithm $\Lambda.\mathsf{Alg}_\lambda$.[3]

The situation described above admittedly has stronger behavioural assumptions on the involved parties than the one we covered in Section 4, where we proved that even $\mathcal{V}$ would not be able to detect the subversion. The reason that we wish to consider this more restricted context is that we will be able to construct a type 2 asymmetric ASA (one that satisfies the above notion of undetectability) that is less susceptible to detection via timing side channels. We will still use a repetition parameter s in the same way as in the ASA from Section 4, but key recovery will be possible with a smaller s.

Before continuing, we will define the notions of IND-CPA security for symmetric and

[3] A version of security preservation appears in Appendix A of the eprint version of [DFP15]. The authors obtained an elegant result relating the security of a subverted scheme to the security of the original scheme and the detectability of the ASA by an adversary who is in possession of the key $\overline{k}$. However, they only considered symmetric ASAs; the asymmetric case is certainly different.

$\text{IND-CPA}_{\mathsf{SE}}(\mathcal{B})$

1. $k \leftarrow\$ \mathsf{SE.KeyGen}()$
2. $b \leftarrow\$ \{0,1\}$
3. $b' \leftarrow\$ \mathcal{B}^{\mathcal{O}_{\mathsf{SE.Enc}}}$
4. **return** $b = b'$

$\mathcal{O}_{\mathsf{SE.Enc}}(m)$

1. **if** $b = 0$ **then**
2. $\quad c \leftarrow\$ \mathsf{SE.Enc}(k, m)$
3. **if** $b = 1$ **then**
4. $\quad c \leftarrow\$ \mathsf{SE.Enc}(k, \mathbf{0})$
5. **return** c

$\text{IND-CPA}_{\mathsf{PKE}}(\mathcal{B})$

1. $(pk, sk) \leftarrow\$ \mathsf{PKE.KeyGen}()$
2. $b \leftarrow\$ \{0,1\}$
3. $b' \leftarrow\$ \mathcal{B}^{\mathcal{O}_{\mathsf{PKE.Enc}}(pk)}$
4. **return** $b = b'$

$\mathcal{O}_{\mathsf{PKE.Enc}}(m)$

1. **if** $b = 0$ **then**
2. $\quad c \leftarrow\$ \mathsf{PKE.Enc}(pk, m)$
3. **if** $b = 1$ **then**
4. $\quad c \leftarrow\$ \mathsf{PKE.Enc}(pk, \mathbf{0})$
5. **return** c

Figure 9: The IND-CPA games for (left) a symmetric encryption scheme SE and (right) a public-key encryption scheme PKE.

asymmetric encryption. This is a weaker form of security than IND$ we used in Section 4 (which implicitly was in chosen-plaintext form), but it is all we will require in this section.

The IND-CPA games for symmetric encryption scheme SE and public-key encryption scheme PKE are given in Figure 9.

The advantage of $\mathcal{B}$ at each of these games is $\text{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{B}) = |\Pr[\text{IND-CPA}_{\mathsf{PKE}}(\mathcal{B})] - \frac{1}{2}|$ and $\text{Adv}^{\text{IND-CPA}}_{\mathsf{SE}}(\mathcal{B}) = |\Pr[\text{IND-CPA}_{\mathsf{SE}}(\mathcal{B})] - \frac{1}{2}|$. Informally, we say that the schemes PKE and SE are IND-CPA-secure if the advantage of any efficient adversary $\mathcal{B}$ in the corresponding game is small.

We use a "multi-challenge" IND-CPA game, where $\mathcal{B}$ is not limited to a single challenge ciphertext from which it has to guess. This multi-challenge game has been used, for example, by Rogaway [Rog04]. We choose to use this particular notion, as it more closely resembles our detectability games, leading to proofs that are easier to follow. Security of multi-challenge IND-CPA follows from single-challenge security using a hybrid argument.

We now present our type 2 asymmetric ASA ASub2. Let SE be a symmetric encryption scheme. Let PKE be an IND-CPA-secure public-key encryption scheme, and let F be a PRF with output space $\{1, ..., \mathsf{PKE.clen}\} \times \{0,1\}$. Then ASub2, an ASA on SE is shown in Figure 10, where s is a parameter of the subversion to bound the loops as before.

The function ASub2.Enc bears many similarities to ASub.Enc from Section 4 and to the ASA of [BJK15]. In fact, the ASA ASub essentially consists of encrypting k to κ using IND$-secure public-key encryption, and then leaking κ using the same techniques as [BPR14]. The ASA ASub2, on the other hand, consists of encrypting k to κ using IND-CPA-secure public-key encryption, and then leaking κ using the same techniques as [BJK15]. We will explore some of these parallels in Section 6.

$\mathsf{ASub2.Enc}(k, m, ek, \tau)$

1. **if** $\tau = \perp$ **then**
2. $\quad \kappa \leftarrow\$ \mathsf{PKE.Enc}(ek, k); \tau \leftarrow \kappa$
3. **else** $\kappa \leftarrow \tau$
4. $j \leftarrow 0$
5. **do**
6. $\quad j \leftarrow j + 1$
7. $\quad r \leftarrow\$ \{0,1\}^{\mathsf{SE.rlen}}$
8. $\quad c \leftarrow \mathsf{SE.Enc}(k, m; r)$
9. $\quad (\sigma, w) \leftarrow \mathsf{F}(ek, c)$
10. **until** $\kappa[\sigma] = w$ **or** $j = s$
11. **return** (c, τ)

Figure 10: Our type 2 asymmetric ASA on symmetric encryption.

The fundamental difference between the two ASAs ASub and ASub2, as well as the reason for no longer requiring PKE to be IND$, becomes evident when we consider how an adversary $\mathcal{V}$ in possession of the key ek is able to interact with the scheme. Informally, the subverter is able to obtain k because anyone with ek can obtain κ, and the subverter can obtain k from κ using xk. In the case of ASub, we required PKE to be IND$ so that the adversary $\mathcal{V}$ would not be able to distinguish the κ they obtain by this process from random bits. Making sure to never leak the same bit of κ twice, we proved that this makes the scheme undetectable. For the ASA ASub2, even an IND$ scheme would not provide this guarantee. Re-use of the same κ for more than $|\kappa|$ encryptions is required: randomization of the leaked bit of κ on each execution means that $\mathcal{V}$ cannot be sure all have been leaked. Therefore all (index, bit)-pairs that $\mathcal{V}$ receives using this decoding process that have the same index will also have the same bit with certainty, and this is unlikely in an unsubverted scheme. While not undetectable, we can still show that the subverted scheme is secure against $\mathcal{V}$, implying that $\mathcal{V}$ is not able to recover the secret

$I_{1/2}(\mathcal{U})$

1. $(ek, xk) \leftarrow\$ \mathsf{ASub2.KeyGen}()$
2. $C \leftarrow \emptyset$
3. $i \leftarrow 1$
4. $\tau_0 \leftarrow \perp$
5. $b \leftarrow\$ \{0,1\}$
6. $b' \leftarrow\$ \mathcal{U}^{\mathcal{O}_{\mathsf{Enc}}, \mathsf{Reset}}$
7. **return** $b = b'$

$\mathsf{Reset}(j),\ 0 \le j < i$

1. **if** $b = 1$ **then**
2. $\quad \tau_i \leftarrow \tau_j$
3. $\quad i \leftarrow i + 1$

$\mathcal{O}_{\mathsf{Enc}}(k, m)$

1. **if** $b = 0$ **then** $c \leftarrow\$ \mathsf{Enc}(k, m)$
2. **if** $b = 1$ **then**
3. $\quad (c, \tau_i) \leftarrow\$ \mathsf{Helper}(k, m, ek, \tau_{i-1})$
4. $\quad i \leftarrow i + 1$
5. **return** c

$\mathsf{Helper}(k, m, ek, \tau)$

1. **if** $\tau = \perp$ **then**
2. $\quad \kappa \leftarrow\$ \mathsf{PKE.Enc}(ek, k); \tau \leftarrow \kappa$
3. **else** $\kappa \leftarrow \tau$
4. $j \leftarrow 0$
5. **do**
6. $\quad j \leftarrow j + 1$
7. $\quad r \leftarrow\$ \{0,1\}^{\mathsf{SE.rlen}}$
8. $\quad c \leftarrow \mathsf{SE.Enc}(k, m; r)$
9. $\quad$ **if** $c \notin C$ **then**
10. $\quad\quad (\sigma_c, w_c) \leftarrow\$ \{0, \ldots, \mathsf{PKE.clen}\} \times \{0,1\}$
11. $\quad\quad C \leftarrow C \cup \{c\}$
12. $\quad$ **else** $\mathsf{bad} \leftarrow \mathsf{true}$
13. $\quad \sigma, w \leftarrow \sigma_c, w_c$
14. **until** $\kappa[\sigma] = w$ **or** $j = s$
15. **return** (c, τ)

Figure 11: Games I_1 and I_2 for the proof of Theorem 3. Game I_1 contains the boxed code while game I_2 does not (including the appropriate indentation corrections).

key in the same way as the subverter.

In the rest of this section, we will prove that ASub2 is a type 2 asymmetric ASA. That is, we will prove that the scheme is (1) undetectable to the user $\mathcal{U}$ in the regular detection game, (2) secure against the third party $\mathcal{V}$, and (3) that the subverter can recover the secret key k.

5.1 Undetectability of our type 2 asymmetric ASA

We first prove undetectability of ASub2 against an adversary $\mathcal{U}$ in the regular state reset detectability game SRDET.

Theorem 3. *Let $\mathcal{U}$ be an adversary in the regular state reset detectability game in Figure 2,* $\mathrm{SRDET}_{\mathsf{ASub2}}(\mathcal{U})$*, with symmetric encryption scheme* SE *as* Λ *and* SE.Enc *as* $\Lambda.\mathsf{Alg}_\lambda$*, where* ASub2.Enc *is the algorithm given in Figure 10. If n is the number of queries that $\mathcal{U}$ makes to its encryption oracle, and η is the min-entropy of* SE.Enc*, then there is an adversary $\mathcal{F}$ against the* PRF*-security of* F *such that* $\mathrm{Adv}^{\mathrm{SRDET}}_{\mathsf{ASub}}(\mathcal{U}) \le 2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{F}}(\mathcal{F}) + (ns)^2 \cdot 2^{-\eta-1}$*. The running time of $\mathcal{F}$ is about that of $\mathcal{U}$, and $\mathcal{F}$ makes at most ns oracle queries.*

Proof. We proceed by a sequence of games. Let I_0 the regular state reset detectability game of Figure 2, with all the substitutions in the theorem statement. Let I_1 be the same as I_0 but with F replaced by lazy random sampling; this is given in Figure 11. Let I_2 be the same as I_1 but with w and σ sampled randomly instead; this is also given in Figure 11. Let I_3 be the regular state reset detectability game where the encryption oracle is replaced by an oracle that simply returns $\mathsf{SE.Enc}(k, m)$, and the Reset oracle removed.

Consider first games I_0 and I_1. Similarly to the proof of Theorem 3, this is a straightforward gamehop based on indistinguishability of the PRF F, so we omit the detailed reduction. Instead, note that an adversary $\mathcal{F}$ in the $\mathrm{PRF}_{\mathsf{F}}(\mathcal{F})$ game is able to completely simulate the games I_0 and I_1 for an adversary $\mathcal{U}$ using the oracle provided to it in place of F. Let b_{PRF} be the challenge bit in the PRF game. Then if $b_{\mathrm{PRF}} = 1$, the Enc oracle simulated by $\mathcal{F}$ proceeds exactly as in game I_1, and if $b_{\mathrm{PRF}} = 0$, then it proceeds exactly as in game I_0. Thus we have $\Pr[\mathcal{F} \Rightarrow 1 \mid b_{\mathrm{PRF}} = 1] = \Pr[I_1]$, $\Pr[\mathcal{F} \Rightarrow 1 \mid b_{\mathrm{PRF}} = 0] = \Pr[I_0]$, and hence $|\Pr[I_1] - \Pr[I_0]| = 2\mathrm{Adv}^{\mathrm{PRF}}_{\mathsf{PKE}}(\mathcal{F})$.

$\text{IND-CPA}'_{\mathsf{ASub2}}(\mathcal{V})$

1. $(ek, xk) \leftarrow\$ \mathsf{ASub2.KeyGen}()$
2. $k \leftarrow\$ \mathsf{SE.KeyGen}()$
3. $\tau \leftarrow \perp$
4. $b \leftarrow\$ \{0,1\}$
5. $b' \leftarrow\$ \mathcal{V}^{\mathcal{O}_{\mathsf{ASub2.Enc}}}(ek)$
6. **return** $b = b'$

$\mathcal{O}_{\mathsf{ASub2.Enc}}(m)$

1. **if** $b = 0$ **then**
2. $\quad (c, \tau) \leftarrow\$ \mathsf{ASub2.Enc}(k, m, ek, \tau)$
3. **if** $b = 1$ **then**
4. $\quad (c, \tau) \leftarrow\$ \mathsf{ASub2.Enc}(k, \mathbf{0}, ek, \tau)$
5. **return** c

Figure 12: The IND-CPA′ game for asymmetric ASA ASub2.

Now consider games I_1 and I_2. In game I_2, we replace the selection of w and σ in the encryption oracle with true random sampling of w and σ, regardless of whether c was input to the random oracle before. Let Col be the event where bad is set to true in game I_2. This happens when some c previously generated by the encryption oracle is obtained again during an encryption oracle query. We upper bound $\Pr[\mathsf{Col}]$: let η be the min-entropy of SE.Enc. Then we have that $\Pr[\mathsf{Col}] \leq \binom{ns}{2} \cdot 2^{-\eta} \leq (ns)^2 \cdot 2^{-\eta-1}$.

By the Fundamental Lemma of Game-Playing [BR06], we have $|\Pr[I_2] - \Pr[I_1]| \leq \Pr[\mathsf{Col}] \leq (ns)^2 \cdot 2^{-\eta-1}$.

Finally, I_3 was defined as the detectability game where the encryption oracle is replaced by an oracle that simply returns $\mathsf{SE.Enc}(k, m)$. In game I_2, since the loop condition is no longer dependent on the selection of c, the implementation $\mathsf{ASub2.Enc}_2$ is identical to $\mathsf{SE.Enc}(k, m)$. Hence $\Pr[I_2] = \Pr[I_3]$. Further, note that $\Pr[I_3] = \frac{1}{2}$, since the encryption oracle in I_3 is not dependent on b.

Putting all these results together, we have

$$\text{Adv}^{\text{SRDET}}_{\mathsf{ASub2}}(\mathcal{U}) \leq 2\text{Adv}^{\text{PRF}}_{\mathsf{F}}(\mathcal{F}) + (ns)^2 \cdot 2^{-\eta-1},$$

as desired.

□

5.2 Security of our type 2 asymmetric ASA

We prove here that the ASA ASub2 is secure against an adversary $\mathcal{V}$ who has knowledge of ek. "Secure", here, should mean in the same sense as the original symmetric encryption scheme: IND-CPA. However, we cannot apply this notion directly to ASub2, since the IND-CPA game does not provide the necessary parameters. Instead, we introduce the modified game IND-CPA′ on ASub2 with adversary $\mathcal{V}$, shown in Figure 12. This game contains the required modifications to Figure 9 in order to include the function ASub2.Enc, as well as providing the embedded key ek to the adversary $\mathcal{V}$. Assuming that SE.Enc is IND-CPA secure, we say that ASub2.Enc is secure if it is IND-CPA′ secure against $\mathcal{V}$.

Theorem 4. *Let* SE *be a symmetric encryption scheme, and let* ASub2 *be the ASA on* SE *given by the description in Figure 10. Let* $\mathcal{V}$ *be an adversary in the game* $\text{IND-CPA}'_{\mathsf{ASub2.Enc}}(\mathcal{V})$. *Then there is an adversary* $\mathcal{B}_1$ *against the* IND-CPA *security of* PKE *and an adversary* $\mathcal{B}_2$ *against the* IND-CPA *security of* SE *such that* $\text{Adv}^{\text{IND-CPA}'}_{\mathsf{ASub2}}(\mathcal{V}) \leq 2\text{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{B}_1) + 2\text{Adv}^{\text{IND-CPA}}_{\mathsf{SE}}(\mathcal{B}_2)$. *The running time of* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *are both about that of* $\mathcal{V}$. *If* n *is the number of oracle queries that* $\mathcal{V}$ *makes, then* $\mathcal{B}_1$ *makes* n *queries to its own oracle, and* $\mathcal{B}_2$ *makes at most* ns *queries to its own oracle.*

Proof. We proceed by a sequence of games. Let J_0 be the $\text{IND-CPA}'_{\mathsf{ASub2.Enc}}(\mathcal{V})$ game. Let J_1 be the same as J_0 but with κ computed as an encryption of $\mathbf{0}$ instead of as an encryption of k. Let J_2 be the same as J_1 but with c computed as an encryption of $\mathbf{0}$ instead of as an encryption of m. All of these games are shown in Figure 13.

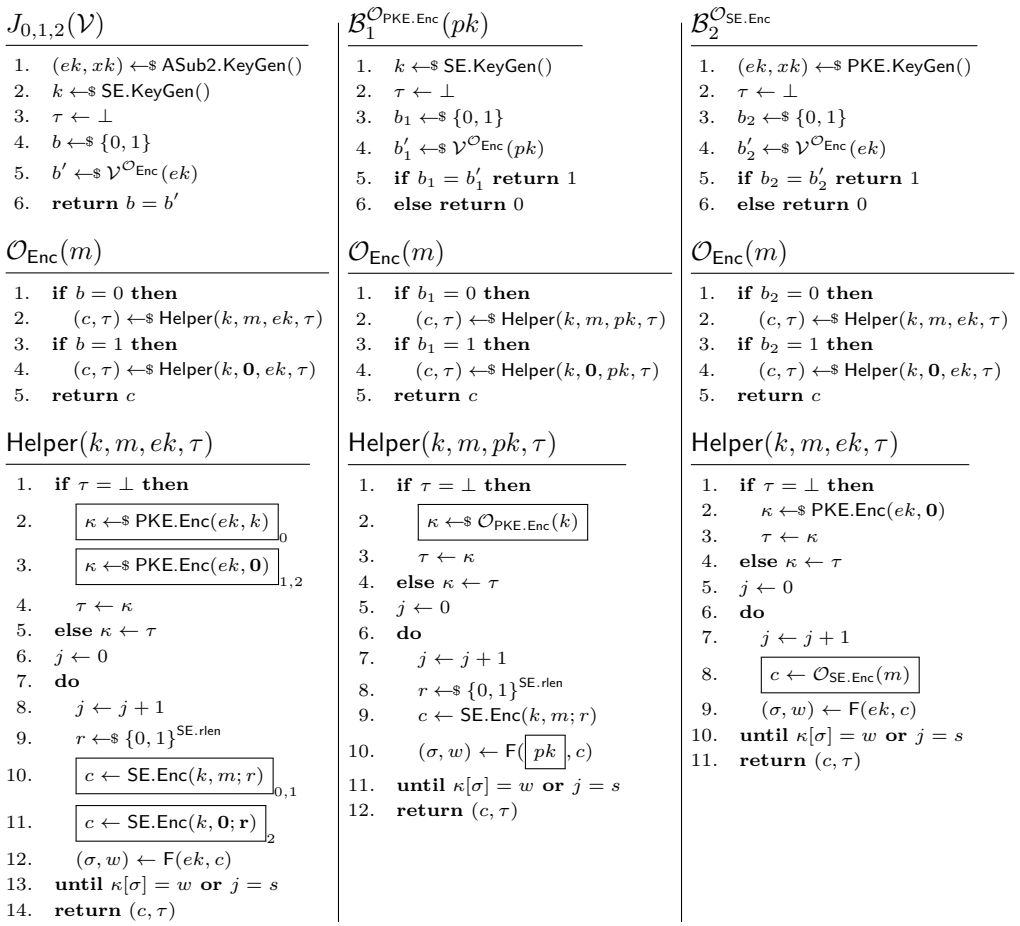

Figure 13: Games J_0, J_1, and J_2 and adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ for proof of Theorem 4. In the J-games, each game only include the boxed code if it has matching subscript. In $\mathcal{B}_1$ and $\mathcal{B}_2$, the boxes serve to highlight changes.

Let $\mathcal{B}_1$, defined in Figure 13, be an adversary to the IND-CPA game on PKE. Acting as a challenger, $\mathcal{B}_1$ simulates the IND-CPA game on ASub2 for $\mathcal{V}$, in particular using the PKE.Enc oracle and public key given to it to simulate ASub2.Enc.

Let b_{pke} denote the bit from the IND-CPA$_{\mathsf{PKE}}(\mathcal{B}_1)$ game. Note that if $b_{pke} = 1$, then the Enc oracle simulated by $\mathcal{B}_1$ proceeds exactly as in game J_1, and if $b_{pke} = 0$, then it proceeds exactly as in game J_0. Thus we have $\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{pke} = 1] = \Pr[J_1]$, $\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{pke} = 0] = \Pr[J_0]$, and hence $|\Pr[J_1] - \Pr[J_0]| = 2\text{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{B}_1)$.

Let $\mathcal{B}_2$, defined in Figure 13, be an adversary to the IND-CPA game on SE. Acting as a challenger, $\mathcal{B}_2$ simulates the game J_1 for $\mathcal{V}$, in particular using the SE.Enc oracle given to it to simulate ASub2.Enc$_1$.

Game J_2 is given in Figure 13. The only change from J_1 is that the oracle will now always use $\mathbf{0}$ in the place of m.

Let b_{se} denote the bit from the IND-CPA$_{\mathsf{SE}}(\mathcal{B}_2)$ game. Note that if $b_{se} = 1$, then the Enc oracle simulated by $\mathcal{B}_2$ proceeds exactly as in game J_2, and if $b_{se} = 0$, then it proceeds exactly as in game J_1. Thus we have $\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{se} = 1] = \Pr[J_2]$, $\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{se} = 0] = \Pr[J_1]$,, and hence $|\Pr[J_2] - \Pr[J_1]| = 2\text{Adv}^{\text{IND-CPA}}_{\mathsf{SE}}(\mathcal{B}_2)$.

Finally, we note that in J_2, there is no difference between the cases $b = 0$ and $b = 1$, since m is never used. Hence $\Pr[J_2] = \frac{1}{2}$.

Putting all these results together, we have

$$\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{ASub2}}(\mathcal{V}) \leq 2\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{B}_1) + 2\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{SE}}(\mathcal{B}_2),$$

as desired.

□

Assuming that PKE and SE are IND-CPA-secure, this result shows that ASub2 is IND-CPA secure against an adversary $\mathcal{V}$ in possession of the embedded key ek.

5.3 Key recovery of our type 2 asymmetric ASA

As in Subsection 4.2, we will treat key recovery somewhat less formally than detectability, and focus on the recovery probability in the case where there is sufficient randomness in the encryption scheme SE.Enc and sufficient unpredictability in the function F to assume that all the σ and w values generated are random.

The following method for key recovery is identical to that of [BJK15], with the extra step of public-key decryption at the end:

1. Collect ciphertexts $c_1, ..., c_n$. Initialize κ to a string of length PKE.clen of null values.
2. For each ciphertext c, compute $(\sigma, w) = \mathsf{F}(ek, c)$. Set $\kappa[\sigma] = w$.
3. Compute $k = \mathsf{PKE.Dec}(xk, \kappa)$

The probability of success here can be calculated by making use of a coupon-collector problem analysis. [4] Suppose PKE is δ-correct. The probability that every ciphertext was chosen to successfully leak a bit ($\mathsf{F}(ek, c) = \kappa[\sigma]$) is $(1 - 2^{-s})^n$. Let ℓ be the length of κ. Then the probability that a given index of κ was never selected is $(1 - 1/\ell)^n$ (assuming F is random). Hence the probability that at least one of the indices was never selected is at most $\ell(1 - 1/\ell)^n$, and the probability that the full κ can be recovered is at least $(1 - 2^{-s})^n \cdot (1 - \ell(1 - 1/\ell)^n)$. Considering any decryption failures in PKE, we have that the probability of recovery of the key k is $P_{kr} = \delta \cdot (1 - 2^{-s})^n \cdot (1 - \ell(1 - 1/\ell)^n)$.

This is sufficient for establishing key recovery in a theoretical setting. [BJK15] give example values of $s = 13$, $\ell = 128$, and $n = 896$, with key recovery probability close to $1/2$. In our setting, we have PKE.clen ≈ 400, and hence we require more ciphertexts. With $n = 2900, s = 13, \delta = 1, \ell = 400$, we get a key recovery probability above $1/2$.

But this analysis does not account for other ways that key recovery could be performed. In fact, it should be clear that even in the presence of erroneous encodings (i.e. some ciphertext c is returned such that $\mathsf{F}(ek, c) \neq \kappa[\sigma]$), the subverter may still be able to recover the correct key if there are enough ciphertexts to encode each index several times. To illustrate this better strategy, consider modifying the above strategy so that in step 2, the subverter takes the most probable value of w for each ciphertext, based on the number of times 0 or 1 was observed. This strategy will tolerate far more errors than our above analysis, while still recovering the key. It may require many ciphertexts to ensure there are large samples to draw from, but simultaneously it enables a smaller s value, since as n grows, the error rate is primarily determined by s.

We evaluated this strategy with a simulation, assuming that the probability of correctly encoding a key bit in a ciphertext is exactly $1 - 2^{-s}$. In our case, the length of the value to leak, κ, is around 400, and with $s = 2$ and $n = 14000$, this majority-voting strategy enables key recovery in over 50% of cases.

This method of key recovery could be even further improved with the observation that the subverter could brute-force a small number of key bits. Given that the incorrectly decoded key bits are more likely to have fewer samples or closer tallies between correct

[4] https://en.wikipedia.org/wiki/Coupon_collector%27s_problem

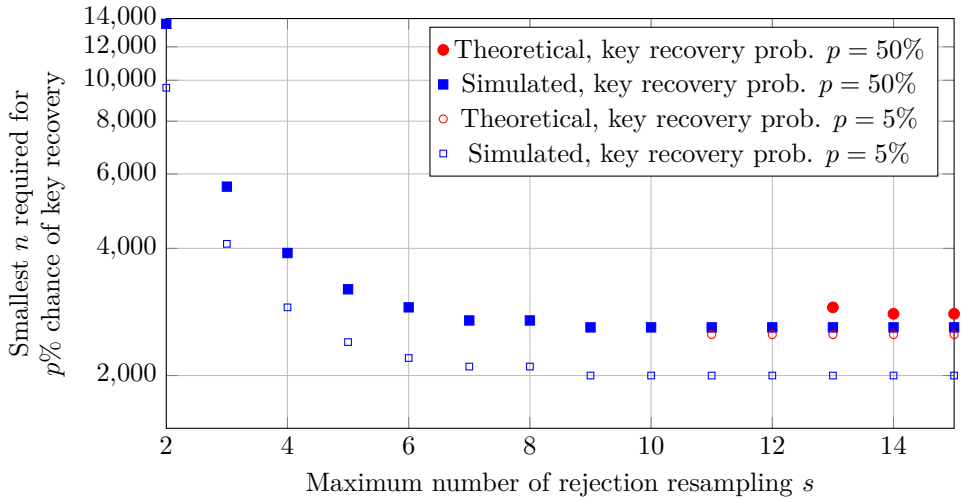

Figure 14: Key recovery parameter tradeoff, showing the smallest number of ciphertexts n (rounded up to the nearest 100) that result in 50% and 5% probability of key recovery, for κ of length 400, and the value of s on the x-axis.

and incorrect encodings, errors may be easy to identify. We will not include this possibility in our analysis.

To further illustrate the tradeoff between n and s, we plotted our simulated results on the graph in Figure 14. The graph shows the smallest n value (rounded up to the nearest 100) that results in 50% and 5% key recovery probability (with $|\kappa| = 400$) for each value of s. We could use a smaller value of $|\kappa|$ here, since we do not require PKE to be IND$ for our type 2 asymmetric ASA, but using $|\kappa| = 400$ certainly provides an upper bound. The values given by the theoretical key recovery probability above are included in red for comparison; values of s lower than 13 and 11 were not able to provide key recovery probability above 50% and 5% respectively for any value of n. Of particular interest are lower values of s: experimental results show it is possible to use values $s = 6$ or smaller, while the theoretical probability would imply that these values of s are unusable.

The main difference between the simulation values and the theoretical approximation comes from the requirement, in the theoretical case, that all attempts to encode key bits in ciphertexts must succeed. With small s, $(1 - 2^{-s})^n$ actually gets smaller as n gets larger and dominates the key recovery estimation, whereas the key recovery probability should increase when more samples are collected. Directly calculating the complicated probability expression of key recovery without this assumption is precisely the difficulty that we are avoiding with the simulation.

Earlier, we noted that one of the advantages of this ASA over the ASA in Section 4 was better resilience to timing detection. Recall that the parameter s determines the maximum number of regular encryptions that the subverted encryption algorithm will do to return one ciphertext. Effectively, the runtime of the subverted scheme is up to s times the runtime of the unsubverted scheme. Knowing this, a large s would allow a user to detect the ASA by timing the execution of the algorithm. We have shown that the ASA ASub2 allows key recovery with $s = 2$, which is the smallest value we can achieve with this acceptance-rejection framework.

While we haven't included timing attacks in our formalism, it is an important area of future work. Even with $s = 2$, a timing attack would easily detect ASub2. It is not clear if there is any way to modify these techniques to achieve a general ASA which is resistant to timing attacks. Indeed, a value of $s = 1$ would be identical to unsubverted

encryption, and there is no obvious way to implement a non-integer value of s. Perhaps an ASA based on acceptance-rejection techniques could have an expected value of s lower than 2, and avoid detection by timing in this manner. Of the published ASAs we evaluated in Section 3, [BSKC19] and [CHY20] avoided timing detection by using an efficient ASA, where the execution time of the subverted algorithm is approximately the same as that of the unsubverted algorithm, but at the cost of making detectable use of state under state reset attacks. This therefore leaves a gap in the literature: are there ASAs which are undetectable under state resets *and* timing attacks, using acceptance-rejection techniques or otherwise?

5.3.1 Key recovery in the presence of state resets

As for our type 1 ASA from section Section 4, regular state resets would reduce key recovery probability of the ASA in this section as it is currently written. However, in this case, it is possible to modify the scheme to maintain key recovery in the presence of state resets. Making PKE deterministic effectively removes dependence of the ASA on any state at all, since the value of κ can be recomputed without state (keeping state anyway would allow the ASA to do the work of recomputing κ only when the state is reset, mitigating some of the inefficiency of this process). If the ASA does not require state, then key recovery works just as well during state resets as without them.

If PKE was deterministic, this would also allow for the possibility of shorter PKE ciphertexts. For example, Bellare, Boldyreva, and O'Neill provide a deterministic PKE scheme which conserves plaintext length. If $|\kappa| = |k| = 128$, then key recovery would require fewer ciphertexts. For example, the simulated 50% recovery values in Figure 14 would converge to 700 instead of 2600.

Unfortunately, deterministic PKE causes difficulties for our security analysis in Subsection 5.2. Theorem 4 bounds the IND-CPA′ advantage against $\mathsf{ASub2}$ using the IND-CPA security of PKE, which would no longer be appropriate for a deterministic PKE. To address this, we would need to instead use security notions for deterministic PKE. Bellare et al. provide security notions for deterministic PKE under the condition of high-entropy plaintexts [BBO07], which applies to our case (k is the only plaintext encrypted, and is high-entropy). An analogue to Theorem 4 would need to be proven with this new security notion in order to prove that an ASA using deterministic PKE is a type 2 asymmetric ASA.

6 Generalized Modifications to Obtain Asymmetric ASAs

In this section, we will explore how the techniques of Sections 4 and 5 generalize. So far, we have only considered asymmetric ASAs on symmetric encryption. Our asymmetric ASAs could be seen as modifications of existing symmetric ASAs. In this section, we will show that these modifications could be done in a much more general manner, without specifying either the cryptographic primitive or the underlying symmetric ASA.

Flexible ASAs. This generalization will apply only to a subclass of symmetric ASAs. A reader might have noticed that the techniques of [BPR14] and [BJK15] could be used to leak any value, and not just the key k. Indeed, it is this property of a symmetric ASA that will allow for a transformation into an asymmetric one.

A flexible ASA fSub of a cryptographic scheme Λ is an ASA that satisfies these additional constraints:

- $\mathsf{fSub.Alg}_\lambda$ takes an additional parameter $\mu \in M$. We call M the space of leakable values. We will assume that M consists of all bit strings.

$\boxed{\text{A}}\text{SRDET}_{\mathsf{fSub}}(\mathcal{U})$

1. $(xk, ek) \leftarrow\$\ \mathsf{fSub.KeyGen}()$
2. $i \leftarrow 1$
3. $\tau_0 \leftarrow \perp$
4. $b \leftarrow\$\ \{0,1\}$
5. $b' \leftarrow\$\ \mathcal{U}^{\mathcal{O}_{\mathsf{Alg}_\lambda}, \mathsf{Reset}}\boxed{(ek)}$
6. **return** $b = b'$

$\mathsf{Reset}(j),\ 0 \le j < i$

1. **if** $b = 1$ **then**
2. $\quad \tau_i \leftarrow \tau_j$
3. $\quad i \leftarrow i + 1$

$\mathcal{O}_{\mathsf{Alg}_\lambda}(x, \mu)$

1. **if** $b = 0$ **then**
2. $\quad y \leftarrow\$\ \Lambda.\mathsf{Alg}_\lambda(x)$
3. **if** $b = 1$ **then**
4. $\quad (y, \tau_i) \leftarrow\$\ \mathsf{fSub.Alg}_\lambda(x, ek, \tau_{i-1}, \mu)$
5. $\quad i \leftarrow i + 1$
6. **return** y

$\text{ASRDET\$}_{\mathsf{fSub}}(\mathcal{U})$

1. $(xk, ek) \leftarrow\$\ \mathsf{fSub.KeyGen}()$
2. $i \leftarrow 1$
3. $\tau_0 \leftarrow \perp$
4. $b \leftarrow\$\ \{0,1\}$
5. $b' \leftarrow\$\ \mathcal{U}^{\mathcal{O}_{\mathsf{Alg}_\lambda}, \mathsf{Reset}}(ek)$
6. **return** $b = b'$

$\mathsf{Reset}(j),\ 0 \le j < i$

1. **if** $b = 1$ **then**
2. $\quad \tau_i \leftarrow \tau_j$
3. $\quad i \leftarrow i + 1$

$\mathcal{O}_{\mathsf{Alg}_\lambda}(x, \mu)$

1. **if** $b = 0$ **then**
2. $\quad y \leftarrow\$\ \Lambda.\mathsf{Alg}_\lambda(x)$
3. **if** $b = 1$ **then**
4. $\quad \mu \leftarrow\$\ \{0,1\}$
5. $\quad (y, \tau_i) \leftarrow\$\ \mathsf{fSub.Alg}_\lambda(x, ek, \tau_{i-1}, \mu)$
6. $\quad i \leftarrow i + 1$
7. **return** y

Figure 15: The regular and augmented state reset detection games for flexible ASAs is on the left, and the game ASRDET\$ is on the right. The augmented game includes the code in the box, while the regular one does not. The ASRDET\$ game is a modification that generates a random μ on each invocation of $\mathcal{O}_{\mathsf{Alg}_\lambda}$ instead of using the supplied value.

- The subverter $\mathcal{A}$ must be able to recover μ from observation of outputs of $\mathsf{fSub.Alg}_\lambda(\cdot, ek, \tau, \mu)$ (analogous to the requirement of key recovery). This must be possible efficiently, as a function of the length of μ

We define the regular and augmented state reset detectability games $\text{SRDET}_{\mathsf{fSub}}(\mathcal{U})$ and $\text{ASRDET}_{\mathsf{fSub}}(\mathcal{U})$ for a flexible ASA fSub in Figure 15. These are modifications to the games in Figure 2 to include μ. The adversary $\mathcal{U}$ supplies the value for μ when invoking its oracle, allowing the leaked value to be related or unrelated to any secret information intended to be used in the cryptographic scheme.

We also define a game ASRDET\$, shown in Figure 15, which we will use in Theorem 5. This game generates a random bit μ instead of using the value given to the oracle.

For our general treatment in this section, we will work with abstract security games. Let Λ be a cryptographic scheme. Let SEC be a cryptographic game with adversary $\mathcal{A}$, which interacts with $\mathcal{A}$ by providing some oracles to $\mathcal{A}$ based on the algorithms of Λ, and specifies a value $\phi \in [0,1]$. The advantage of $\mathcal{A}$ at the game SEC is defined as $\text{Adv}^{\text{SEC}}_{\Lambda}(\mathcal{A}) = |\Pr[\text{SEC}(\mathcal{A})] - \phi|$. This generalization encompasses the security notion of IND-CPA for symmetric encryption used in Section 5, as well as security notions such as unforgeability on signature schemes and MAC tags, and others.

As in Section 5 with the IND-CPA game, we require a modification to the SEC game when talking about subversions. Let SEC be a security game on a cryptographic primitive Λ. Let fSub be an ASA on Λ. We will assume without loss of generality that all invocations of the algorithm $\Lambda.\mathsf{Alg}_\lambda$ occur within oracles provided to the adversary $\mathcal{A}$. Define SEC$'$ on fSub to be the same as SEC but with the following modifications:

- A state τ (initialized to $\perp$) and embedded key ek (generated with $\mathsf{fSub.KeyGen}$) are assigned at the beginning of the game, and ek is provided to the adversary $\mathcal{A}$ (i.e. $\mathcal{A}$ is invoked with ek as a parameter and its output is ignored).
- For any oracle $\mathcal{O}$ provided to the adversary $\mathcal{A}$ which invokes $\Lambda.\mathsf{Alg}_\lambda$, $\mathcal{O}$ is modified to take μ as an additional parameter. Call these oracles μ-oracles.

$\Gamma_1(\mathsf{fSub}).\mathsf{Alg}_\lambda(x, ek, \tau, \mu)$

1. $(ek', \overline{k}) \leftarrow ek$
2. **if** $\tau = \perp$ **then**
3. $\quad \mu' \leftarrow\$ \mathsf{PKE.Enc}(ek', \mu)$
4. $\quad \sigma \leftarrow 0$
5. $\quad \tau' \leftarrow \perp$
6. **else**
7. $\quad ((\sigma, \mu), \tau') \leftarrow \tau$
8. **if** $\sigma = \mathsf{PKE.clen}$ **then**
9. $\quad \sigma \leftarrow 1$
10. $\quad \mu' \leftarrow\$ \mathsf{PKE.Enc}(ek', \mu)$
11. **else**
12. $\quad \sigma \leftarrow \sigma + 1$
13. $(y, \tau') \leftarrow\$ \mathsf{fSub.Alg}_\lambda(x, \overline{k}, \tau', \mu'[\sigma])$
14. **return** $(y, ((\sigma, \mu'), \tau'))$

$\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda(x, ek, \tau, \mu)$

1. $(ek', \overline{k}) \leftarrow ek$
2. **if** $\tau = \perp$ **then**
3. $\quad \mu' \leftarrow\$ \mathsf{PKE.Enc}(ek', \mu)$
4. $\quad \tau' \leftarrow \perp$
5. **else**
6. $\quad (\mu', \tau') \leftarrow \tau$
7. $(y, \tau') \leftarrow\$ \mathsf{fSub.Alg}_\lambda(x, \overline{k}, \tau', \mu')$
8. **return** $(y, (\mu', \tau'))$

Figure 16: Definitions of Γ_1 and Γ_2, our transformations on flexible symmetric ASAs to obtain flexible asymmetric ASAs.

- Every instance of $\Lambda.\mathsf{Alg}_\lambda$ is replaced by $\mathsf{fSub.Alg}_\lambda$, with the state variable τ being assigned on output, where the input to $\Lambda.\mathsf{Alg}_\lambda$ is given as the first argument to $\mathsf{fSub.Alg}_\lambda$, and the required ek, μ, and τ are provided as inputs.

Essentially, we are modifying every instance of the line $y \leftarrow\$ \Lambda.\mathsf{Alg}_\lambda(x)$ to be of the form $(y, \tau) \leftarrow\$ \mathsf{fSub.Alg}_\lambda(x, ek, \tau, \mu)$. This SEC′ game will be used in Theorem 6.

We will also define another game as a further modification to SEC′. Let S be the tuple of all variables assigned during the game SEC′, before the the first instance where a μ-oracle is provided to $\mathcal{A}$. Let $S' = S \setminus \{\tau, ek\}$. Let g be a function whose domain is the set of all possible tuples S', and whose output is a bit string. We say a function g of this form is a *selection function* for the game SEC′. We define the game $\mathrm{SEC}'^{g}_{\mathsf{fSub}}(\mathcal{A})$ is the same as $\mathrm{SEC}'_{\mathsf{fSub}}(\mathcal{A})$ but where the μ-oracles no longer accept μ as a parameter, and instead, each oracle has the line $\mu \leftarrow g(S')$ at the start of its execution. We will still refer to these oracles as the μ-oracles.

$\mathrm{SEC}'^{g}_{\mathsf{fSub}}$ is an appropriate game with which to evaluate the security of fSub. The selection function g represents what value is targeted as being leaked by the ASA, and the adversary A is tasked with breaking the security of fSub when this value is being leaked. For a secure flexible ASA, we require that $\mathrm{Adv}^{\mathrm{SEC}'^{g}}_{\mathsf{fSub}}$ is small for all selection functions g.

6.1 ASA modifications

We now define our two modifications to flexible symmetric ASAs which result in flexible asymmetric ASAs, one of type 1 and the other of type 2. Let Λ be a cryptographic scheme. Let fSub be a flexible symmetric ASA on Λ, subverting the function $\Lambda.\mathsf{Alg}_\lambda$. Let PKE be a public-key encryption scheme. Define $\Gamma_1(\mathsf{fSub})$ and $\Gamma_2(\mathsf{fSub})$ as the flexible asymmetric ASAs on Λ with $\Gamma_1(\mathsf{fSub}).\mathsf{Alg}_\lambda$ and $\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda$ defined in Figure 16. The subversion-key generation algorithms for both subversions are the same: the extraction key is the private key generated by $\mathsf{PKE.KeyGen}$ and the embedded key is the concatenation of the public key generated by $\mathsf{PKE.KeyGen}$ and the key generated by $\mathsf{fSub.KeyGen}$.

These two modifications are exactly the techniques we used in Section 4 and Section 5 respectively, to build asymmetric ASAs on symmetric encryption, just written for a generic scheme.

Recovery of μ. We will be treating the analysis of recovery of μ for Γ_1 and Γ_2 relatively informally. If fSub is a flexible symmetric ASA, then a subverter $\mathcal{A}$ in possession of the key $\overline{k}$, upon observing outputs of the function $\mathsf{fSub.Alg}_\lambda(\cdot, \overline{k}, \tau, \mu)$, will be able to recover μ.

$\mathcal{D}_1^{\mathcal{O}_{\mathsf{PKE.Enc}}}(pk)$

1. $(xk, (ek', \overline{k})) \leftarrow\$ \Gamma_1(\mathsf{fSub}).\mathsf{KeyGen}()$
2. $ek \leftarrow (pk, \overline{k})$
3. $i \leftarrow 1$
4. $\tau_0 \leftarrow \bot$
5. $b_{\text{det}} \leftarrow\$ \{0,1\}$
6. $b'_{\text{det}} \leftarrow\$ \mathcal{U}_1^{\mathcal{O}_{\mathsf{Alg}_\lambda}, \mathsf{Reset}}(ek)$
7. **if** $b_{\text{det}} = b'_{\text{det}}$
8. **return** 1
9. **else**
10. **return** 0

$\mathsf{Reset}(j),\ 0 \le j < i$

1. **if** $b_{\text{det}} = 1$ **then**
2. $\tau_i \leftarrow \tau_j$
3. $i \leftarrow i + 1$

$\mathcal{O}_{\mathsf{Alg}_\lambda}(x, \mu)$

1. **if** $b_{\text{det}} = 0$ **then**
2. $y \leftarrow\$ \Lambda.\mathsf{Alg}_\lambda(x)$
3. **if** $b_{\text{det}} = 1$ **then**
4. $(y, \tau_i) \leftarrow\$ \mathsf{Helper}(x, ek, \tau_{i-1}, \mu)$
5. $i \leftarrow i + 1$
6. **return** y

$\mathsf{Helper}(x, ek, \tau, \mu)$

1. $(pk, \overline{k}) \leftarrow ek$
2. **if** $\tau = \bot$ **then**
3. $\boxed{\mu' \leftarrow\$ \mathcal{O}_{\mathsf{PKE.Enc}}(\mu)}$
4. $\sigma \leftarrow 0$
5. $\tau' \leftarrow \bot$
6. **else** $((\sigma, \mu), \tau') \leftarrow \tau$
7. **if** $\sigma = \mathsf{PKE.clen}$ **then**
8. $\sigma \leftarrow 1$
9. $\boxed{\mu' \leftarrow\$ \mathcal{O}_{\mathsf{PKE.Enc}}(\mu)}$
10. **else** $\sigma \leftarrow \sigma + 1$
11. $(y, \tau') \leftarrow\$ \mathsf{fSub}.\mathsf{Alg}_\lambda(x, \overline{k}, \tau', \mu')$
12. **return** $(y, ((\sigma, \mu'), \tau'))$

Figure 17: Adversary $\mathcal{D}_1$ for the proof of Theorem 5.

Then $\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda$ enables the same for an adversary in possession of the extraction key xk: the value that fSub is asked to leak will always be a chosen encryption μ' of μ under PKE, and since fSub enables recovery of μ' in this context, $\mathcal{A}$ can recover μ using xk.

The case of Γ_1 is a little bit more complicated. In this case, μ is encrypted to μ', each of whose bits is used exactly once before a new encryption is computed. Each bit is leaked using fSub. In order to conclude that $\mathcal{A}$ is able to recover μ', we require that fSub is capable of leaking single bits with high probability after the observation of only one output. Otherwise, a full value of μ' would not be able to be recovered.

Proving $\Gamma_1(\mathsf{fSub})$ is a type 1 asymmetric ASA. We can now prove the main results of this section. The first deals with the augmented undetectability of $\Gamma_1(\mathsf{fSub})$, showing that it is a type 1 asymmetric ASA.

Theorem 5. *Let Λ be a cryptographic scheme. Let fSub be a flexible symmetric* ASA *on Λ. Let $\mathcal{U}_1$ be an adversary for the* ASRDET *game of Figure 15 on $\Gamma_1(\mathsf{fSub})$. Then there is an adversary $\mathcal{D}_1$ such that $\mathrm{Adv}^{\mathrm{ASRDET}}_{\Gamma_1(\mathsf{fSub})}(\mathcal{U}_1) \le \mathrm{Adv}^{\mathrm{ASRDET\$}}_{\mathsf{fSub}}(\mathcal{U}_1) + 2\mathrm{Adv}^{\mathrm{IND\$}}_{\mathsf{PKE}}(\mathcal{D}_1)$. The running time of $\mathcal{D}_1$ is about that of $\mathcal{U}_1$ plus the running time of $\mathsf{PKE.KeyGen}$ and $\mathsf{fSub.KeyGen}$, and $\mathcal{D}_1$ makes the same number of queries to its own oracle as $\mathcal{U}_1$.*

Proof. We will proceed by a sequence of games. This sequence will be similar to the first part of the proof of Theorem 2. Let L_0 be the augmented state reset detectability game $\mathrm{ASRDET}_{\Gamma_1(\mathsf{fSub})}(\mathcal{U}_1)$. Let L_1 be the same as L_0 but with μ' sampled randomly instead of computed as an encryption of μ. Let L_2 be the augmented state reset detectability game $\mathrm{ASRDET\$}_{\mathsf{fSub}}$.

Let $\mathcal{D}_1$ be an adversary to the IND\$ game for PKE that simulates games L_0 and L_1 for $\mathcal{U}_1$ by using its own $\mathcal{O}_{\mathsf{PKE.Enc}}$ oracle in place of $\mathsf{PKE.Enc}$, given in Figure 17.

Let $b_\$$ denote the bit from the $\mathrm{IND\$}_{\mathsf{PKE}}(\mathcal{D}_1)$ game. Note that if $b_\$ = 1$, then the oracle simulated by $\mathcal{D}_1$ proceeds exactly as in game L_1, and if $b_\$ = 0$, then it proceeds exactly as in game L_0. Thus we have $\Pr[\mathcal{D}_1 \Rightarrow 1 \mid b_\$ = 1] = \Pr[L_1]$, $\Pr[\mathcal{D}_1 \Rightarrow 1 \mid b_\$ = 0] = \Pr[L_0]$, and hence $|\Pr[L_1] - \Pr[L_0]| = 2\mathrm{Adv}^{\mathrm{IND\$}}_{\mathsf{PKE}}(\mathcal{D}_1)$.

From here, we note that, as in the proof of Theorem 2, the first lines (lines 1–10 in adversary $\mathcal{D}_1$'s Helper) of the implementation of $\Gamma_1(\mathsf{fSub}).\mathsf{Alg}_\lambda$ in game L_1 act as bookkeeping in order to use exactly one new random bit μ at a time in $\mathsf{fSub}.\mathsf{Alg}_\lambda$. Substituting these lines for $\mu' \leftarrow\!\!\$\ \{0,1\}$ and moving this line to the first line of the oracle call gives us precisely the game $L_2 = \mathrm{ASRDET\$}_{\mathsf{fSub}}(\mathcal{U}_1)$, and $\Pr[L_1] = \Pr[L_2]$.

Taken together we get

$$\mathrm{Adv}^{\mathrm{ASRDET}}_{\Gamma_1(\mathsf{fSub})}(\mathcal{U}_1) \leq \mathrm{Adv}^{\mathrm{ASRDET\$}}_{\mathsf{fSub}}(\mathcal{U}_1) + 2\mathrm{Adv}^{\mathrm{IND\$}}_{\mathsf{PKE}}(\mathcal{D}_1),$$

as desired. □

Proving $\Gamma_2(\mathsf{fSub})$ is a type 2 asymmetric ASA. Next we deal with both the regular undetectability and the security of $\Gamma_2(\mathsf{fSub})$, proving that $\Gamma_2(\mathsf{fSub})$ is a type 2 asymmetric ASA. The security of $\Gamma_2(\mathsf{fSub})$ is evaluated using the game $\mathrm{SEC}^g_{\Gamma_2(\mathsf{fSub})}$, and an adversary's advantage at this game is bounded by using the game $\mathrm{SEC}'_{\mathsf{fSub}}$. In this sense, we are relating the security of $\Gamma_2(\mathsf{fSub})$ to the security of fSub in the situation where the adversary must provide the value of μ, the "value to be leaked" by fSub. If an adversary must provide the value to be leaked, then we would expect that there is nothing more to be learned for the adversary, unless the scheme is leaking information in other ways. Thus, intuitively, what we require is that fSub is not leaking any additional information; this is codified in the game $\mathrm{SEC}'_{\mathsf{fSub}}$. This provides the intuition for our security result, and the formalization of what is required from fSub to ensure that $\Gamma_2(\mathsf{fSub})$ is secure.

Theorem 6. *Let Λ be a cryptographic scheme. Let fSub be a flexible symmetric* ASA *on Λ. Let* SEC *be a security game on Λ. Let g be a selection function for the game* SEC′*. Let $\mathcal{U}_1$ be an adversary for the* SRDET *game on $\Gamma_2(\mathsf{fSub})$ and $\mathcal{V}_1$ be an adversary for the* SEC^g *game on $\Gamma_2(\mathsf{fSub})$. Then there is an adversary $\mathcal{U}_2$ such that* $\mathrm{Adv}^{\mathrm{SRDET}}_{\Gamma_2(\mathsf{fSub})}(\mathcal{U}_1) = \mathrm{Adv}^{\mathrm{SRDET}}_{\mathsf{fSub}}(\mathcal{U}_2)$*, and adversaries $\mathcal{D}_2$ and $\mathcal{V}_2$ such that* $\mathrm{Adv}^{\mathrm{SEC}^g}_{\Gamma_2(\mathsf{fSub})}(\mathcal{V}_1) \leq \mathrm{Adv}^{\mathrm{SEC}'}_{\mathsf{fSub}}(\mathcal{V}_2) + 2\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{D}_2)$*. The running time of $\mathcal{U}_2$ is about that of $\mathcal{U}_1$ and $\mathcal{U}_2$ makes the same number of queries to its own oracle as $\mathcal{U}_1$. The running time of $\mathcal{D}_2$ is about that of $\mathcal{V}_1$, and $\mathcal{D}_2$ makes the same number of queries to its own oracle as $\mathcal{V}_1$. If the number of oracle queries that $\mathcal{V}_1$ makes is n, then the running time of $\mathcal{V}_2$ is about that of $\mathcal{V}_1$, plus the running time of* $\mathsf{PKE.KeyGen}$*, plus the running time of* $\mathsf{PKE.Enc}$ *times n. For each oracle in the game* SEC*, $\mathcal{V}_2$ makes the same number of queries to its corresponding oracle as $\mathcal{V}_1$ does.*

Proof. For the undetectability result, we will construct the adversary $\mathcal{U}_2$ directly; the construction is given in Figure 18. We claim that $\mathcal{U}_2$ exactly simulates the game $\mathrm{SRDET}_{\Gamma_2(\mathsf{fSub})}$ for $\mathcal{U}_1$.

To see this, observe that for any query to $\mathcal{O}_{\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda}$ that $\mathcal{U}_1$ makes, the value of μ' passed to $\mathcal{O}_{\mathsf{fSub}.\mathsf{Alg}_\lambda}$ is an encryption of μ under PKE. Furthermore, the same encrypted μ' is reused in all invocations of $\mathsf{fSub}.\mathsf{Alg}_\lambda$ unless $\tau_{i-1} = \perp$, in which case it is recomputed. This is exactly the behaviour of the $\mathrm{SRDET}_{\Gamma_2(\mathsf{fSub})}$ game. Hence $\Pr\big[\mathrm{SRDET}_{\Gamma_2(\mathsf{fSub})}(\mathcal{U}_1)\big] = \Pr[\mathrm{SRDET}_{\mathsf{fSub}}(\mathcal{U}_2)]$, giving our result.

For the security inequality, we will proceed by a sequence of games. Let K_0 be the game $\mathrm{SEC}^g_{\Gamma_2(\mathsf{fSub})}(\mathcal{V}_1)$. Let K_1 be the game $\mathrm{SEC}^0_{\Gamma_2(\mathsf{fSub})}(\mathcal{V}_1)$, where the 0 in the superscript represents the function that outputs 0 on all inputs. Let K_2 be the game $\mathrm{SEC}'_{\mathsf{fSub}}(\mathcal{V}_2)$.

Let $\mathcal{D}_2$ be an adversary to the IND-CPA game for PKE that simulates games K_0 and K_1 for $\mathcal{V}_1$ by using its own $\mathcal{O}_{\mathsf{PKE.Enc}}$ oracle in place of $\mathsf{PKE.Enc}$. This is shown in Figure 18.

Let b_{PKE} denote the bit from the $\text{IND-CPA}_{\mathsf{PKE}}(\mathcal{D}_2)$ game. Note that if $b_{\mathrm{PKE}} = 1$, then the oracle simulated by $\mathcal{D}_2$ proceeds exactly as in game K_1, since in K_1 the only value of μ passed to the Helper function is 0. If $b_{\mathrm{PKE}} = 0$, then $\mathcal{D}_2$ proceeds exactly as in game K_0. Thus we have $\Pr[\mathcal{D}_2 \Rightarrow 1 \mid b_{\mathrm{PKE}} = 1] = \Pr[K_1]$, $\Pr[\mathcal{D}_2 \Rightarrow 1 \mid b_{\mathrm{PKE}} = 0] = \Pr[K_0]$, and hence $|\Pr[K_1] - \Pr[K_0]| = 2\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{D}_2)$.

$\mathcal{U}_2^{\mathcal{O}_{\mathsf{fSub}.\mathsf{Alg}_\lambda}, \mathsf{Reset}_2}$

1. $(xk, ek') \leftarrow\$ \mathsf{PKE.KeyGen}()$
2. $i \leftarrow 1$
3. $\tau_0 \leftarrow \perp$
4. $b \leftarrow\$ \mathcal{U}_1^{\mathcal{O}_{\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda}, \mathsf{Reset}_1}$
5. **return** b

$\mathsf{Reset}_1(j),\ 0 \le j < i$

1. $\tau_i \leftarrow \tau_j$
2. $i \leftarrow i + 1$
3. $\mathsf{Reset}_2(j)$

$\mathcal{O}_{\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda}(x, \mu)$

1. **if** $\tau_{i-1} = \perp$ **then**
2. $\quad \mu' \leftarrow\$ \mathsf{PKE.Enc}(ek', \mu)$
3. **else**
4. $\quad \mu' \leftarrow \tau_{i-1}$
5. $\tau_i \leftarrow \mu'$
6. $y \leftarrow\$ \mathcal{O}_{\mathsf{fSub}.\mathsf{Alg}_\lambda}(x, \mu')$
7. $i \leftarrow i + 1$
8. **return** y

$\mathcal{D}_2^{\mathcal{O}_{\mathsf{PKE.Enc}}}(pk)$

1. SEC^g with the first part of
2. $ek = (ek', \overline{k})$, which is generated
3. by $\Gamma_2(\mathsf{fSub}).\mathsf{KeyGen}$, replaced by
4. pk, and $\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda$
5. replaced by Helper.

$\mathsf{Helper}(x, ek, \tau, \mu)$

1. $(pk, \overline{k}) \leftarrow ek$
2. **if** $\tau = \perp$ **then**
3. $\quad \boxed{\mu' \leftarrow\$ \mathcal{O}_{\mathsf{PKE.Enc}}(\mu)}$
4. $\quad \tau' \leftarrow \perp$
5. **else**
6. $\quad (\mu', \tau') \leftarrow \tau$
7. $(y, \tau') \leftarrow\$ \mathsf{fSub}.\mathsf{Alg}_\lambda(x, \overline{k}, \tau', \mu')$
8. **return** $(y, (\mu', \tau'))$

$\mathcal{V}_2(\overline{k})$

1. $(xk, ek') \leftarrow\$ \mathsf{PKE.KeyGen}()$
2. $ek \leftarrow (ek', \overline{k})$
3. $\tau \leftarrow \perp$
4. $\mathcal{V}_1(ek)$

$\mathcal{V}_2^{\mathcal{O}'_j, j \in J \subseteq \{1,\ldots,t\}}(x)$

1. $y \leftarrow\$ \mathcal{V}_1^{\mathcal{O}_j, j \in J \subseteq \{1,\ldots,t\}}(x)$
2. **return** y

$\mathcal{O}_j(x), 1 \le j \le t_1$

1. **if** $\tau = \perp$ **then**
2. $\quad \mu' \leftarrow\$ \mathsf{PKE.Enc}(ek', \mathbf{0})$
3. **else** $\mu' \leftarrow \tau$
4. $\tau \leftarrow \mu'$
5. $y \leftarrow\$ \mathcal{O}'_j(x, \mu')$
6. **return** y

$\mathcal{O}_j(x), t_1 \le j \le t$

1. $y \leftarrow\$ \mathcal{O}'_j(x)$
2. **return** y

Figure 18: Adversaries $\mathcal{U}_2, \mathcal{D}_2$, and $\mathcal{V}_2$ for the proof of Theorem 6. For $\mathcal{U}_2$ and $\mathcal{V}_2$, queries made by $\mathcal{U}_1$ and $\mathcal{V}_1$ to their oracles are answered by using the oracles provided to them as indicated. $\mathcal{D}_2$ runs the indicated code to simulate the SEC^g game, and any oracle query that would use $\Gamma_2(\mathsf{fSub}).\mathsf{Alg}_\lambda(x, ek, \tau, \mu)$ is answered using the code in Helper.

Next, we will construct the adversary $\mathcal{V}_2$ in the game $\mathrm{SEC}'_{\mathsf{fSub}}$ directly. Suppose that the game SEC provides oracles $\mathcal{O}_j$ for $1 \le j \le t$, each with inputs labeled x and outputs labeled y as tuples. After being modified for game SEC′, assume that oracles $\mathcal{O}_1, \ldots \mathcal{O}_{t_1}$ for some $1 \le t_1 \le t$ are μ-oracles, and the rest are not. The adversary $\mathcal{V}_2$ is given in Figure 18.

We claim that $\mathcal{V}_2$ exactly simulates the game $\mathrm{SEC}^0_{\Gamma_2(\mathsf{fSub})}$ for $\mathcal{V}_1$. The argument and construction are very similar to the symmetric detectability result. Observe that for any query to any oracle accepting a parameter μ that $\mathcal{V}_1$ makes, the value of μ' passed to $\mathcal{V}_2$'s oracle $\mathcal{O}'_j$ is an encryption of $\mathbf{0}$ under PKE. Furthermore, the same μ' is reused in all future oracle invocations. All oracles not accepting a parameter μ are provided directly to $\mathcal{V}_1$. This exactly simulates the $\mathrm{SEC}^0_{\Gamma_2(\mathsf{fSub})}$ game. Hence $\Pr[K_1] = \Pr\left[\mathrm{SEC}^0_{\Gamma_2(\mathsf{fSub})}(\mathcal{V}_1)\right] = \Pr\left[\mathrm{SEC}'_{\mathsf{fSub}}(\mathcal{V}_2)\right] = \Pr[K_2]$.

Combining these results, we get

$$\mathrm{Adv}^{\mathrm{SEC}^g}_{\Gamma_2(\mathsf{fSub})}(\mathcal{V}_1) \le \mathrm{Adv}^{\mathrm{SEC}'}_{\mathsf{fSub}}(\mathcal{V}_2) + 2\mathrm{Adv}^{\text{IND-CPA}}_{\mathsf{PKE}}(\mathcal{D}_2),$$

as desired. □

For the symmetric subversions on symmetric encryption from [BPR14] and [BJK15], we can show that the quantities $\mathrm{Adv}^{\mathrm{SEC}'}_{\mathsf{fSub}}$ and $\mathrm{Adv}^{\mathrm{ASRDET\$}}_{\mathsf{fSub}}$ are small for any adversary; the proofs in Section 4 and Section 5 did essentially that. Proving these advantages are small for a given flexible ASA fSub would allow one to draw conclusions about the undetectability and security of $\Gamma_1(\mathsf{fSub})$ and $\Gamma_2(\mathsf{fSub})$ using the theorems in this section.

7 Discussion and Future Work

In this work, we formalized the approach of using state resets to detect ASA, and showed how asymmetric ASAs can be constructed from symmetric ASAs. A key observation of

our work is that many published ASAs are detectable via realistic state reset attacks (such as virtual machine snapshotting) despite having small state. As such, we encourage future threat models to incorporate this notion of state resets when evaluating the detectability of new ASAs.

We identify several topics that warrant further exploration.

Our type 1 and type 2 asymmetric ASAs have running times that are multiple times longer than the algorithms they subvert. This would make them detectable by a user or other adversary who is able to time execution of the algorithm. Furthermore, all of the published ASAs which are more efficient (in the sense of running times closer to those of the underlying algorithm) are, to our knowledge, detectable in the presence of state resets. An interesting direction for future work is modeling detection of ASAs based on running time and developing ASAs that resist both time-based detection and state reset-based detection, or proving that an ASA satisfying both is impossible. Timing attacks are but one way to model increased detection capabilities on the part of the user; other capabilities and their effects on known ASAs are also of interest.

Further improvements to our ASAs are possible. Our type 1 asymmetric ASA modification is dependent on the ability of the underlying ASA to leak single bits reliably, and this requirement could potentially be removed. Perhaps modifications could also be made on "non-flexible" ASAs. Methods to enable key recovery with fewer ciphertexts likely exist, and would be more effective in certain contexts. For example, suppose a user is changing their symmetric key too often for the ASAs in this paper to effectively leak the key, say every 300 messages (leaking a PKE-encrypted value would require at least 400 ciphertexts). Consider an ASA on symmetric encryption that leaks values in 2 stages: first, it leaks a longer encryption of a locally generated shorter key, resulting in a secret key shared with the external subverter. Next, it uses this new key to undetectably leak the targeted secret key directly, requiring interception of fewer ciphertexts. This ASA could be formulated as a generic modification to an underlying symmetric ASA, as we did in Section 6. Such an ASA could be a type 1 or type 2 asymmetric ASA depending on the exact implementation of the leaks, and, after the initial shared key is established, would recover keys more quickly than the ASAs we presented in this paper.

Further to the above, it is an open question whether a stateless type 1 asymmetric ASA exists. While we have argued that state does not lead immediately to detection, the effectiveness of a stateful ASA can be mitigated by using state resets. In fact, any ASA that requires state for key recovery can be fully countered by a state reset after every invocation of the subverted algorithm (as mentioned before, this may have significant impacts on performance). We see no straightforward way of modifying our type 1 asymmetric ASA to make it stateless. We encourage work on discovery of a stateless type 1 ASA or on an impossibility result.

We have not addressed the topic of countermeasures to ASAs significantly in this work. Several different avenues exist in the literature: deterministic algorithms [BPR14, DFP15, BJK15]; reverse firewalls using re-randomization [AMV15]; immunization methods [AFMV19], including a split-program methodology for preventing ASAs [RTYZ17, RTYZ16, TY17]; and so-called self-guarding cryptographic schemes [FM18]. All of these solutions assume some extra trusted component (for example, a trusted firewall system, a period of time where the scheme is not subverted, or an unsubvertable algorithm composition step). Each solution is able to produce significant guarantees on the scheme's resistance to ASAs. These countermeasures work against our ASAs as well, but we nonetheless encourage more work on methods to prevent ASAs which are simple and easy to implement in practice.

References

[AFMV19] Giuseppe Ateniese, Danilo Francati, Bernardo Magri, and Daniele Venturi. Public immunization against complete subversion without random oracles. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 465–485. Springer, Heidelberg, June 2019.

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, October 2015.

[AP19a] Marcel Armour and Bertram Poettering. Substitution attacks against message authentication. *IACR Trans. Symm. Cryptol.*, 2019(3):152–168, 2019.

[AP19b] Marcel Armour and Bertram Poettering. Subverting decryption in AEAD. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 22–41. Springer, Heidelberg, December 2019.

[BBG13] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security, Sep 2013. https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 535–552. Springer, Heidelberg, August 2007.

[BJK15] Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 1431–1440. ACM Press, October 2015.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, August 2014.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[BSKC19] Joonsang Baek, Willy Susilo, Jongkil Kim, and Yang-Wai Chow. Subversion in practice: How to efficiently undermine signatures. *IEEE Access*, 7:68799–68811, 2019.

[BWP+20] Sebastian Berndt, Jan Wichelmann, Claudius Pott, Tim-Henrik Traving, and Thomas Eisenbarth. ASAP: Algorithm substitution attacks on cryptographic protocols. Cryptology ePrint Archive, Report 2020/1452, 2020. https://eprint.iacr.org/2020/1452.

[CHY20] Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 98–128. Springer, Heidelberg, December 2020.

[CNE+14] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J. Bernstein, Jake Maskiewicz, Hovav Shacham, and Matthew Fredrikson. On the practical exploitability of dual EC in TLS implementations. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 319–335. USENIX Association, August 2014.

[Des90] Yvo Desmedt. Abuses in cryptography and how to fight them. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 375–389. Springer, Heidelberg, August 1990.

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598. Springer, Heidelberg, March 2015.

[FM18] Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In Steve Chong and Stephanie Delaune, editors, *CSF 2018Computer Security Foundations Symposium*, pages 76–90. IEEE Computer Society Press, 2018.

[GBPG03] Eu-Jin Goh, Dan Boneh, Benny Pinkas, and Philippe Golle. The design and implementation of protocol-based hidden key recovery. In Colin Boyd and Wenbo Mao, editors, *ISC 2003*, volume 2851 of *LNCS*, pages 165–179. Springer, Heidelberg, October 2003.

[LCWW18] Chi Liu, Rongmao Chen, Yi Wang, and Yongjun Wang. Asymmetric subversion attacks on signature schemes. In Willy Susilo and Guomin Yang, editors, *ACISP 18*, volume 10946 of *LNCS*, pages 376–395. Springer, Heidelberg, July 2018.

[Möl04] Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS 2004*, volume 3193 of *LNCS*, pages 335–351. Springer, Heidelberg, September 2004.

[Pat99] Kenneth G. Paterson. Imprimitive permutation groups and trapdoors in iterated block ciphers. In Lars R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 201–214. Springer, Heidelberg, March 1999.

[Rog04] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004.

[RP97] Vincent Rijmen and Bart Preneel. A family of trapdoor ciphers. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 139–148. Springer, Heidelberg, January 1997.

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, December 2016.

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, October / November 2017.

[SFKR15] Bruce Schneier, Matthew Fredrikson, Tadayoshi Kohno, and Thomas Ristenpart. Surreptitiously weakening cryptographic systems. Cryptology ePrint Archive, Report 2015/097, 2015. http://eprint.iacr.org/2015/097.

[Sim85] Gustavus J. Simmons. The subliminal channel and digital signature. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *EUROCRYPT'84*, volume 209 of *LNCS*, pages 364–378. Springer, Heidelberg, April 1985.

[TY17] Qiang Tang and Moti Yung. Cliptography: Post-snowden cryptography. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2615–2616. ACM Press, October / November 2017.

[YY96] Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Heidelberg, August 1996.

[YY97] Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997.

[YY98] Adam Young and Moti Yung. Monkey: Black-Box symmetric ciphers designed for MONopolizing KEYs. In Serge Vaudenay, editor, *FSE'98*, volume 1372 of *LNCS*, pages 122–133. Springer, Heidelberg, March 1998.

[YY03] Adam L. Young and Moti Yung. Backdoor attacks on black-box ciphers exploiting low-entropy plaintexts. In Reihaneh Safavi-Naini and Jennifer Seberry, editors, *ACISP 03*, volume 2727 of *LNCS*, pages 297–311. Springer, Heidelberg, July 2003.

[YY04] Adam Young and Moti Yung. A subliminal channel in secret block ciphers. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 198–211. Springer, Heidelberg, August 2004.