

# Algorithm Substitution Attacks: Detecting ASAs Using State Reset and Making ASAs Asymmetric

by

Philip Hodges

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Mathematics  
in  
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2021

© Philip Hodges 2021

### **Author's Declaration**

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Statement of Contributions

This thesis is based on research contained in the publication TODO in TODO. This publication was written primarily by Philip Hodges under the supervision of Douglas Stebila, who provided invaluable insights and editorial contributions. Both are co-authors of the published paper. Differences between [] and this thesis were authored solely by Philip Hodges, including the abstract, parts of the introduction, parts of the preliminaries, and some additional pedagogical explanations throughout.

## Abstract

The field of cryptography has made incredible progress in the last several decades. With the formalization of security goals and the methods of provable security, we have achieved many privacy and integrity guarantees in a great variety of situations. However, all guarantees are limited by their assumptions on the model’s adversaries. Snowden’s revelations of the NSA’s participation in the subversion of standardized cryptography have shown that powerful adversaries will not always act in the way that common cryptographic models assume. As such, it is important to continue to expand the capabilities of the adversaries in our models to match the capabilities and intentions of real-world adversaries, and to examine the consequences on the security of our cryptography.

In this thesis, we study Algorithm Substitution Attacks (ASAs), which are one way to model this increase in adversary capability. In an ASA, an algorithm in a cryptographic scheme  $\Lambda$  is substituted for a subverted version. The goal of the adversary is to recover a secret that will allow them to compromise the security of  $\Lambda$ , while requiring that the attack is undetectable to the users of the scheme. This model was first formally described by Bellare, Paterson, and Rogaway (Crypto 2014), and allows for the possibility of a wide variety of cryptographic subversion techniques. Since their paper, many successful ASAs on various cryptographic primitives and potential countermeasures have been demonstrated.

We will address several shortcomings in the existing literature. First, we formalize and study the use of state resets to detect ASAs. While state resets have been considered as a possible detection method since the first papers on ASAs, future works have only informally reasoned about the effect of state resets on ASAs. We show that many published ASAs that use state are detectable with simple practical methods relying on state resets. Second, we add to the study of asymmetric ASAs, where the ability to recover secrets is guaranteed to only the attacker who implemented the ASA. We describe two asymmetric ASAs on symmetric encryption based on modifications to previous ASAs. We also generalize this result, allowing for any symmetric ASA (on any cryptographic scheme) satisfying certain properties to be transformed into an asymmetric ASA. This 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.

## Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Stebila, for his critical role in my development as a researcher. I am eternally grateful for the time spent helping me guide my thoughts and refine my ideas until I could produce work I was truly proud of. Furthermore, the work in this thesis was directly inspired by research of Dr. Stebila's. I could not have asked for a better mentor during my degree.

I would also like to thank both Dr. Stebila and Dr. Menezes for their extensive involvement in my growth as an educator. Both have given me many opportunities to engage with pedagogy in a meaningful way, and shared their many thoughts on the subject with me through innumerable discussions. Most importantly, they both lead by example by always delivering courses created with the utmost care and effort. Thank you for everything you do.

I would like to thank those involved in the generous scholarship programs that have supported me during this degree. This includes everyone involved in the administration and selection committees of the RBC Graduate Scholarship in Cybersecurity and the Queen Elizabeth II Graduate Scholarship in Science and Technology, inside and outside the University of Waterloo. Generous programs such as these make all the difference in the lives of graduate students across Canada.

I would like to thank the entire Department of Combinatorics and Optimization in the Faculty of Mathematics at the University of Waterloo. Every faculty member and member of the administrative staff put a lot of work into the development of graduate students in the department, and the spirit of collegiality this supports cannot be understated.

*Dedicated to Rebecca*

# Table of Contents

List of Figures	ix
List of Tables	xi
<b>1 Introduction</b>	<b>1</b>
1.1 State resets for detection of ASAs . . . . .	4
1.2 Asymmetric ASAs . . . . .	5
<b>2 Preliminaries and Definitions</b>	<b>8</b>
2.1 Games and algorithms . . . . .	8
2.2 Cryptographic schemes . . . . .	9
2.3 Pseudo-random functions and random oracles . . . . .	9
2.4 Algorithm substitution attacks . . . . .	11
<b>3 Using State Reset to Detect ASAs</b>	<b>13</b>
3.1 Detection of ASAs using state reset . . . . .	15
3.1.1 Detecting Ateniese, Magri, and Venturi’s ASA using simple state reset	16
3.1.2 Detecting Baek, Susilo, Kim, and Chow’s ASA using sophisticated state reset . . . . .	17
3.1.3 Detecting Chen, Huang, and Yung’s ASA using state resets . . . . .	18
3.2 Undetectability of Bellare, Paterson, and Rogaway’s ASA . . . . .	20
3.3 Discussion . . . . .	22

<b>4</b>	<b>A Type 1 Asymmetric ASA on Symmetric Encryption</b>	<b>23</b>
4.1	Undetectability of our type 1 asymmetric ASA . . . . .	25
4.2	Key recovery of our type 1 asymmetric ASA . . . . .	27
4.2.1	Key recovery in the presence of state resets . . . . .	28
<b>5</b>	<b>A Type 2 Asymmetric ASA on Symmetric Encryption</b>	<b>32</b>
5.1	Undetectability of our type 2 asymmetric ASA . . . . .	35
5.2	Security of our type 2 asymmetric ASA . . . . .	38
5.3	Key recovery of our type 2 asymmetric ASA . . . . .	40
5.3.1	Key recovery in the presence of state resets . . . . .	42
<b>6</b>	<b>Generalized Modifications to Obtain Asymmetric ASAs</b>	<b>46</b>
6.1	Making symmetric flexible ASAs asymmetric . . . . .	49
<b>7</b>	<b>Discussion and Future Work</b>	<b>56</b>
	<b>References</b>	<b>58</b>



# List of Figures

2.1	The PRF security game. . . . .	10
3.1	The augmented and non-augmented state reset detection games . . . . .	14
3.2	Relationships between detectability definitions in several works . . . . .	15
3.3	ASAs for analysis in Chapter 3 . . . . .	17
3.4	ASA on symmetric encryption by Bellare, Paterson, and Rogaway [BPR14] and game for proof of its undetectability in Theorem 1. . . . .	21
4.1	The ciphertext indistinguishability-from-random game for a public-key en- cryption scheme . . . . .	24
4.2	Type 1 asymmetric ASA on symmetric encryption . . . . .	24
4.3	Games $G_0$ through $G_4$ for the proof of Theorem 2 . . . . .	30
4.4	Adversary $\mathcal{B}$ for the proof of Theorem 2 . . . . .	31
5.1	The IND-CPA games for a symmetric encryption scheme <b>SE</b> and a public- key encryption scheme <b>PKE</b> . . . . .	34
5.2	Type 2 asymmetric ASA on symmetric encryption . . . . .	34
5.3	Games $I_1$ and $I_2$ for the proof of Theorem 3 . . . . .	36
5.4	The IND-CPA' game for asymmetric ASA <b>ASub2</b> . . . . .	38
5.5	Games $J_0$ , $J_1$ , and $J_2$ and adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ for proof of Theorem 4 . .	44
5.6	Simulation for improved practical key recovery for a randomized ASA . . .	45
5.7	Key recovery parameter tradeoff . . . . .	45

6.1	The regular and augmented state reset detection games for flexible ASAs and the game ASRDET\$ . . . . .	47
6.2	Definitions of transformation $\Gamma_1$ and $\Gamma_2$ on flexible symmetric ASAs to obtain flexible asymmetric ASAs . . . . .	49
6.3	Adversary $\mathcal{D}_1$ for the proof of Theorem 5 . . . . .	51
6.4	Adversaries $\mathcal{U}_2$ , $\mathcal{D}_2$ , and $\mathcal{V}_2$ for the proof of Theorem 6 . . . . .	54

# List of Tables

1.1	Comparison of properties of various ASAs . . . . .	7
-----	--	---

# Chapter 1

## Introduction

In cryptography, one of the primary security models we use for reasoning about confidentiality and privacy involves three parties: Alice, Bob, and Eve. Alice and Bob wish to send messages between the two of them. Since their messages are of a private nature, they encrypt them in some way, so that anyone reading the messages is unable to determine what they mean. To illustrate this, we consider the eavesdropper Eve. If Eve is unable to determine significant information about the messages, we say the encryption scheme is secure. This model is very flexible, and a variety of restrictions and/or capabilities can be given to the parties involved. Using this model, we can follow the methodology of provable security in cryptography to come up with very strong guarantees on the privacy afforded by our encryption scheme, subject to the assumptions we make along the way.

It is important to critically examine all of the assumptions we make when proving cryptography secure, and we will consider one in particular. In general, when we implement cryptography in the real world, we wish to apply the security guarantees from our model. An ever-present assumption during this application is that the model accurately reflects the real-world situation in which we are implementing our cryptography. In particular, the capabilities and limitations we give the adversary Eve in our model need to accurately reflect the capabilities and intentions of our expected adversaries in the real world. Furthermore, this can easily become a cat-and-mouse game: if an adversary is effectively eavesdropping using a simple technique, which we then mitigate through implementation of a new scheme with a more robust security analysis that precludes that simple technique, will the adversary simply admit defeat? Or will they choose a new technique outside of the model in which we have proved security?

A particular way that an adversary can work outside of most classical security models

is by undermining the cryptographic schemes being used. If Alice and Bob believe that they are using a secure encryption scheme, but in reality they are not, then Eve may have an advantage. If Eve is able to maliciously influence Alice and Bob to use an insecure scheme without their knowledge, we would call this undermining cryptography. There are many ways in which this could occur, including the introduction of backdoors [RP97, Pat99], for example. More importantly, there are several real-world instances where the undermining of cryptography may have occurred. In the real world, this includes the potential deliberate weakening of DES keys to only 56 bits at the time of standardization [SB88] and the potential that there was a backdoor implemented in the pseudorandom bit generator Dual\_EC\_DRBG [CNE<sup>+</sup>14], which was standardized in 2006 despite knowledge of the possibility of the backdoor at the time. After Snowden’s revelations of the activities of the National Security Agency (NSA) of the United States in 2013 [BBG13], we know that the NSA has worked to undermine cryptography in the past.

In the presence of adversaries that are willing to and capable of working outside of our cryptographic security model by undermining our cryptographic schemes, what are we to do? The ideal solution would be to include such subversion techniques in our model, and re-develop schemes that are once again secure in this model. One way to model such subversion is through 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. This model captures any method that an attacker might employ to undermine cryptography by modifying the algorithms used by the users.

In this thesis, we continue the study of ASAs. We contribute two main improvements to the existing literature. 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.

**Work prior to [BPR14].** The study of subverting a cryptographic algorithm by changing the implementation to differ from the specification is not new. Termed kleptography,

research 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].

**Algorithm substitution attacks (ASAs).** Suppose a user  $\mathcal{U}$  is using a symmetric encryption scheme  $\text{SE}$  for encryption of their communications. An attacker (sometimes referred to as Big Brother) is able to exchange the encryption algorithm  $\text{SE.Enc}$  for an alternative algorithm  $\text{Sub.Enc}$ , which we call the subverted algorithm. While using the algorithm  $\text{Sub.Enc}$ ,  $\mathcal{U}$  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  $\text{Sub.Enc}$  and  $\text{SE.Enc}$ . 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  $\bar{k}$  with the subverted algorithm, and requires that  $\text{SE.Enc}$  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  $\bar{k}$  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  $\bar{k}$ , 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  $\bar{k}$  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.”

PH: decide on whether these last two paragraphs need to be here. Instead we would put a line about where they will be discussed.

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<sup>+</sup>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 Chapter 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 thesis, we will consider two different kinds of asymmetric ASAs. In a *type 1* asymmetric ASA, the subversion is required to be undetectable to an adver-



sary 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 [Chapter 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 [Chapter 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 1.1](#).

Finally, in [Chapter 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.

	[BPR14]	[BJK15]	[BSKC19]	[CHY20]	Our type 1	Our type 2
Asymmetric	○	○	●	●	●	●
<i>No state reset</i>						
Undetectable vs. regular adversary	●	●	●	●	●	●
Undetectable vs. augmented adversary	○	○	●	●	●	○
Secure vs. augmented adversary	○	○	●	●	●	●
<i>State reset</i>						
Undetectable vs. regular adversary (SRDET)	●	●	○	○	●	●
Undetectable vs. augmented adversary (AS-RDET)	○	○	○	○	●	○
Secure vs. augmented adversary	○	○	●	●	●	●
Intercepted transmissions needed	128	$\approx 700$	3	2	400	$\approx 2600$
Runtime multiplier	$\geq 7$	$\geq 2$	$\approx 1$	$\approx 1$	$\geq 9$	$\geq 2$

Table 1.1: Comparison of properties of various ASAs. An augmented adversary refers to an adversary in possession of the embedded subversion key. If applicable,  $|k| = 128$ .

# Chapter 2

## Preliminaries and Definitions

### 2.1 Games and algorithms

Proofs in this work will use the cryptographic game-playing framework [BR06]. In these games, assignment is denoted by  $\leftarrow$ , while random sampling is denoted by  $\leftarrow_{\$}$ . We write  $y \leftarrow_{\$} A(x)$  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  $y \leftarrow A(x; r)$ . We will also write  $A \Rightarrow y$  to indicate that  $A$  returns  $y$ .

We will use min-entropy as a measure of the randomness of an algorithm. Define  $\eta_A$  according to

$$2^{-\eta_A} = \max_{x,y} (\Pr[y \leftarrow A(x; r)]) \quad ,$$

where the probability is taken over the choice of coins  $r$ . The min-entropy of  $A$  is  $\eta_A$ . A null value is denoted  $\perp$ . 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.

The game-playing framework was introduced by Bellare and Rogaway [BR06] as a more structured method for cryptographic proofs, as it easily allows for incremental steps and standardized techniques in proofs. The primary tool is that of a “game transition”, where we bound the difference between the probabilities that two games return true. We use

many game transitions to create a chain of games between two games (representing two security notions of interest), and a total bound between the success probabilities of the two games. We will use several common techniques for game transitions throughout this work, and introduce the details as required.

## 2.2 Cryptographic schemes

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

A symmetric encryption scheme  $\text{SE}$  is composed of three algorithms:  $\text{SE.KeyGen}$ ,  $\text{SE.Enc}$ , and  $\text{SE.Dec}$ .  $\text{SE.KeyGen}$  randomly selects a single secret key  $k$  of length  $\text{SE.klen}$  from  $\{0, 1\}^{\text{SE.klen}}$ .  $\text{SE.Enc}$  is a randomized algorithm with coins  $r \in \{0, 1\}^{\text{SE.rlen}}$  and takes a key and a plaintext  $m \in \{0, 1\}^{\text{SE.mlen}}$  and produces a ciphertext  $c \in \{0, 1\}^{\text{SE.clen}}$ .  $\text{SE.Dec}$  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  $\text{PKE}$  is similarly composed of three algorithms:  $\text{PKE.KeyGen}$ ,  $\text{PKE.Enc}$ , and  $\text{PKE.Dec}$ .  $\text{PKE.KeyGen}$  randomly generates a secret key  $sk$  and a public key  $pk$ .  $\text{PKE.Enc}$  is a randomized algorithm with coins  $r \in \{0, 1\}^{\text{PKE.rlen}}$  and takes a public key and a plaintext  $m \in \{0, 1\}^{\text{PKE.mlen}}$  and produces a ciphertext  $c \in \{0, 1\}^{\text{PKE.clen}}$  (note that we will consider only fixed length ciphertexts for public-key schemes).  $\text{PKE.Dec}$  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  $\delta$ -correct if, for all  $sk, pk$  generated from  $\text{PKE.KeyGen}$  and  $m$ ,

$$\Pr[\text{PKE.Dec}(sk, \text{PKE.Enc}(pk, m)) = m] \geq \delta ,$$

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  $\delta$  value will always be 1 unless otherwise stated.

## 2.3 Pseudo-random functions and random oracles

In this work, 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

$\text{PRF}_F(\mathcal{F})$	$\mathcal{O}_F(x)$
1. $k \leftarrow \$ K_F$	1. <b>if</b> $b = 0$ <b>then</b> $w \leftarrow F(k, x)$
2. $X \leftarrow \emptyset$	2. <b>if</b> $b = 1$ <b>then</b>
3. $b \leftarrow \$ \{0, 1\}$	3. <b>if</b> $x \notin X$ <b>then</b>
4. $b' \leftarrow \$ \mathcal{F}^{\mathcal{O}_F}$	4. $w_x \leftarrow \$ W$
5. <b>return</b> $b = b'$	5. $X \leftarrow X \cup \{x\}$
	6. <b>return</b> $w_x$

Figure 2.1: The PRF security game.

function (PRF). Let  $F : K_F \times \{0, 1\}^* \rightarrow W$  be a function, for some output set  $W$  and key space  $K_F$ . Let the PRF game for  $F$  be as defined in Figure 2.1. For an adversary  $\mathcal{F}$  in the PRF game for  $F$ , we define the advantage of  $\mathcal{F}$  as

$$\text{Adv}_F^{\text{PRF}}(\mathcal{F}) = \left| \Pr[\text{PRF}_F(\mathcal{F})] - \frac{1}{2} \right|.$$

Note that we can also write the advantage as follows:

$$\begin{aligned}
\text{Adv}_F^{\text{PRF}}(\mathcal{F}) &= \left| \Pr[\text{PRF}_F(\mathcal{F})] - \frac{1}{2} \right| \\
&= \left| \Pr[\mathcal{F} \Rightarrow 1 \text{ and } b = 1] + \Pr[\mathcal{F} \Rightarrow 0 \text{ and } b = 0] - \frac{1}{2} \right| \\
&= \left| \frac{1}{2} \Pr[\mathcal{F} \Rightarrow 1 \mid b = 1] + \frac{1}{2} \Pr[\mathcal{F} \Rightarrow 0 \mid b = 0] - \frac{1}{2} \right| \\
&= \left| \frac{1}{2} \Pr[\mathcal{F} \Rightarrow 1 \mid b = 1] + \frac{1}{2} - \frac{1}{2} \Pr[\mathcal{F} \Rightarrow 1 \mid b = 0] - \frac{1}{2} \right| \\
&= \frac{1}{2} |\Pr[\mathcal{F} \Rightarrow 1 \mid b = 1] - \Pr[\mathcal{F} \Rightarrow 1 \mid b = 0]|.
\end{aligned}$$

We will use the equivalence of the above two forms in a variety of contexts. In particular, for game transitions, it will be useful to recognize this equivalence when the  $b = 0$  and  $b = 1$  cases of the PRF game define two other different games that  $\mathcal{F}$  plays. Then the difference between the success probability of the two games can be seen with the above relation to be the advantage of  $\mathcal{F}$  at the PRF game.

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 2.1](#) 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  $\mathcal{O}_F$  behaves as a lazily-sampled random function.)

## 2.4 Algorithm substitution attacks

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

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

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

The idea here is that the algorithm  $\text{Sub.Alg}_\lambda$  is chosen by an adversary  $\mathcal{A}$  who is trying to subvert the security of scheme  $\Lambda$ . A user  $\mathcal{U}$  of  $\Lambda$  will unknowingly use  $\text{Sub.Alg}_\lambda$ , which has the key  $ek$  embedded, in place of  $\Lambda.\text{Alg}_\lambda$ . The adversary  $\mathcal{A}$  will observe  $\mathcal{U}$ 's communication with other users of the scheme  $\Lambda$ , and violate the security of  $\Lambda$  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  $\text{Sub.Alg}_\lambda$ .

Note that we consider here only the subversion of a single algorithm of the scheme  $\Lambda$ . 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](#)].

---

<sup>1</sup>For clarity we make a slight abuse of notation here, writing  $\text{Sub.Enc}(k, m, ek, \tau)$  instead of  $\text{Sub.Enc}(x, ek, \tau)$  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  $\bar{k}$ , 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  $\text{Sub.Alg}_\lambda$  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*  $\Lambda$  (e.g., symmetric encryption), and vice versa. We will discuss the advantages and disadvantages of an asymmetric ASA in [Chapter 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  $\text{Sub.Alg}_\lambda$  (with the embedded key and state implicitly provided), and calculate how well  $\mathcal{U}$  can differentiate  $\text{Sub.Alg}_\lambda$  from  $\Lambda.\text{Alg}_\lambda$ . Formalization of the notion of detectability will be covered in the next chapter.

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  $\text{Sub.Alg}_\lambda$ .

## Chapter 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 Chapter 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 ( $\text{ASRDET}_{\text{Sub}}(\mathcal{U})$  and  $\text{SRDET}_{\text{Sub}}(\mathcal{U})$  respectively) are given in Figure 3.1, for some cryptographic scheme  $\Lambda$  and ASA  $\text{Sub}$ . 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  $\text{Sub.Alg}_\lambda$  or the original algorithm  $\Lambda.\text{Alg}_\lambda$ . 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).



$\boxed{\text{ASRDET}}_{\text{Sub}}(\mathcal{U})$	$\mathcal{O}_{\text{Alg}_\lambda}(x)$
<ol style="list-style-type: none"> <li>1. <math>(xk, ek) \leftarrow \\$ \text{Sub.KeyGen}()</math></li> <li>2. <math>i \leftarrow 1</math></li> <li>3. <math>\tau_0 \leftarrow \perp</math></li> <li>4. <math>b \leftarrow \\$ \{0, 1\}</math></li> <li>5. <math>b' \leftarrow \\$ \mathcal{U}^{\mathcal{O}_{\text{Alg}_\lambda}, \text{Reset}}(\boxed{ek})</math></li> <li>6. <b>return</b> <math>b = b'</math></li> </ol>	<ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 0</math> <b>then</b></li> <li>2.   <math>y \leftarrow \\$ \Lambda.\text{Alg}_\lambda(x)</math></li> <li>3. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>4.   <math>(y, \tau_i) \leftarrow \\$ \text{Sub.Alg}_\lambda(x, ek, \tau_{i-1})</math></li> <li>5.   <math>i \leftarrow i + 1</math></li> <li>6. <b>return</b> <math>y</math></li> </ol>
<hr/> <b>Reset</b> ( $j$ ), $0 \leq j < i$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>2.   <math>\tau_i \leftarrow \tau_j</math></li> <li>3.   <math>i \leftarrow i + 1</math></li> </ol>	

Figure 3.1: 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.

For an adversary  $\mathcal{U}$  in the state reset detectability games, we define the advantages of  $\mathcal{U}$  as

$$\text{Adv}_{\text{Sub}}^{\text{SRDET}}(\mathcal{U}) = |\Pr[\text{SRDET}_{\text{Sub}}(\mathcal{U})] - \frac{1}{2}|$$

and

$$\text{Adv}_{\text{Sub}}^{\text{ASRDET}}(\mathcal{U}) = |\Pr[\text{ASRDET}_{\text{Sub}}(\mathcal{U})] - \frac{1}{2}|.$$

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 3.2. 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.

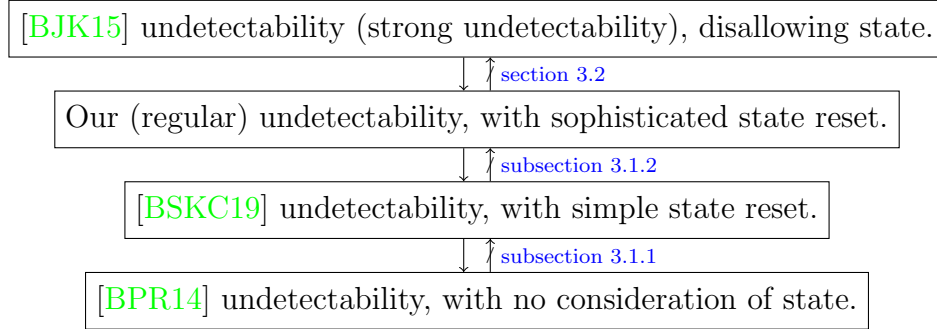


Figure 3.2: Relationships between various detectability games. Arrows indicate that if an ASA is undetectable in the game at the tail of the arrow, then it is also undetectable in the game at the head of the arrow. Crossed arrows indicate that there exists an ASA undetectable in the game at the tail of the arrow but detectable at the head of the arrow.

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

### 3.1 Detection of ASAs using state reset

In this section we will look at several 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  $\text{SIG}$  is composed of three algorithms:  $\text{SIG.KeyGen}$ ,  $\text{SIG.Sign}$ , and  $\text{SIG.Ver}$ .  $\text{SIG.KeyGen}$  randomly selects a secret private key  $sk$  and a public verification key  $pk$  as a pair from the key space  $\mathcal{K}_{\text{SIG}}$ .  $\text{SIG.Sign}$  is a randomized algorithm with coins  $r \in \{0, 1\}^{\text{SIG.rlen}}$ . It takes a private key and a message  $m \in \mathcal{M}_{\text{SIG}}$  and produces a signature  $s \in \mathcal{S}_{\text{SIG}}$ .  $\text{SIG.Ver}$  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  $d = \text{SIG.rlen}$ , 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 3.3a. On each execution of  $\text{Sub.Sign}$ , 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 3.1 with  $\text{SIG}$  as  $\Lambda$  and  $\text{SIG.Sign}$  as  $\Lambda.\text{Alg}_\lambda$  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  $\tau_1 = (G(\bar{k}), d)$ . The detector then calls the reset oracle with  $j = 0$ , and  $\tau_2$  is set to  $\tau_0 = \perp$ . On a second oracle call with the message  $m$  and signing key  $sk$ ,  $\tau_3$  is set to  $(G(\bar{k}), d)$  as before. Let  $s_1$  and  $s_2$  be the two signatures received. Note that the same  $v$  and  $\ell$  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

$\text{Sub.Sign}(sk, m, \bar{k}, \tau)$	$\text{Sub.Sign}(x, m, \bar{k}, \tau)$	$\text{Sub.Encaps}(ek, pk, \tau)$
<pre> 1. if <math>\tau = \perp</math> then 2.   <math>\tau \leftarrow (\bar{k}, 0)</math> 3.   <math>(t, \ell) \leftarrow \tau</math> 4.   if <math>\ell \geq  sk </math> then 5.     <math>\ell \leftarrow 0</math> 6.   <math>(v, t) \leftarrow G(t)</math> 7.   <math>\tilde{r} \leftarrow v \oplus sk[\ell + 1, \ell + d]</math> 8.   <math>\tau \leftarrow (t, \ell + d)</math> 9.   <math>s \leftarrow \text{SIG.Sign}(sk, m; \tilde{r})</math> 10. return <math>(s, \tau)</math> </pre>	<pre> 1. if <math>\tau = \perp</math> then 2.   <math>\tau \leftarrow (0, \perp)</math> 3.   <math>(j, \sigma) \leftarrow \tau</math> 4.   if <math>j = 0 \bmod 2</math> then 5.     <math>\kappa \leftarrow \mathbb{Z}_q</math> 6.     <math>r \leftarrow H_1(g^\kappa)</math> 7.     <math>s \leftarrow \kappa^{-1}(H_2(m) + xr) \bmod q</math> 8.   else 9.     <math>\tilde{\kappa} \leftarrow H_3(\bar{k}, \sigma)</math> 10.    <math>r \leftarrow H_1(g^{\tilde{\kappa}})</math> 11.    <math>s \leftarrow \tilde{\kappa}^{-1}(H_2(m) + xr) \bmod q</math> 12.    <math>\tau \leftarrow (j + 1, r)</math> 13. return <math>((s, r), \tau)</math> </pre>	<pre> 1. if <math>\tau = \perp</math> then 2.   <math>\tau \leftarrow \mathcal{R}_{\text{KEM}}</math> 3. else 4.   <math>t \leftarrow \text{KEM.kgen}(ek, \tau)</math> 5.   <math>\tau \leftarrow F(ek, t)</math> 6.   <math>k \leftarrow \text{KEM.kgen}(pk, \tau)</math> 7.   <math>c \leftarrow \text{KEM.cgen}(\tau)</math> 8. return <math>(c, \tau)</math> </pre>
(a) ASA on signature schemes, by Ateniese et al. [AMV15]	(b) ASA on DSA, by Baek et al. [BSKC19]	(c) ASA on KEMs, by Chen et al. [CHY20]

Figure 3.3: ASAs for analysis in Chapter 3

of the time for an unsubverted scheme, yielding large detection advantage.

### 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 3.1, with SIG as  $\Lambda$  and SIG.Sign as  $\Lambda.\text{Alg}_\lambda$ , 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 SIG.Sign will first sample  $\kappa \leftarrow \mathbb{Z}_q$ , and then return

$$(r, s) \leftarrow (H_1(g^\kappa), \kappa^{-1}(H_2(m) + xr) \bmod q) .$$

Let  $H_3$  be a PRF. The ASA from [BSKC19] is shown in Figure 3.3b. 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  $\kappa$  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  $\sigma$  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  $\tau_1 = (1, H_1(g^\kappa))$  on the first call, for some randomly chosen  $\kappa$ . The detector then calls the reset oracle with  $j = 1$  so that the next state  $\tau_3$  is set to prior state  $\tau_1 = (1, H_1(g^\kappa))$ . 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:  $\text{KEM.KeyGen}$ ,  $\text{KEM.Encaps}$ , and  $\text{KEM.Decaps}$ .  $\text{KEM.KeyGen}$  randomly generates a secret decapsulation key  $sk$  and a public encapsulation key  $pk$ .  $\text{KEM.Encaps}$  is a randomized algorithm with

coins  $r \in \mathcal{R}_{\text{KEM}}$ . It takes a public key and produces a ciphertext  $c \in \mathcal{C}_{\text{KEM}}$ , and a session key  $k \in \mathcal{K}_{\text{KEM}}$ . For the ASA from [CHY20], we require that it decomposes into three components:

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

Finally,  $\text{KEM.Decaps}$  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  $\mathcal{R}_{\text{KEM}}$  and returns a value in  $\mathcal{R}_{\text{KEM}}$ . The ASA from [CHY20] is given in Figure 3.3c. 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  $\tau_i$ . 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  $xk$ :  $t_{i-1} \leftarrow \text{KEM.Decaps}(xk, c_{i-1})$ . Then  $\tau_i \leftarrow F(ek, t_{i-1})$ . This allows one to compute  $k_i \leftarrow \text{KEM.kgen}(pk, \tau_i)$ , the shared key corresponding to the ciphertext  $c_i$ .

The detection game we use is ASRDET from Figure 3.1 with  $\text{KEM}$  as  $\Lambda$  and  $\text{KEM.Encaps}$  as  $\Lambda.\text{Alg}_\lambda$ . 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  $\tau$ . The detector then calls the reset oracle with  $j = 1$ , and  $\tau_3$  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  $\tau$  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 chapter, 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 3.4a, where  $F$  is a PRF which takes a key and a ciphertext of  $\text{SE.Enc}$  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 3.1,  $\text{SRDET}_{\text{Sub}}(\mathcal{U})$ , with symmetric encryption scheme  $\text{SE}$  as  $\Lambda$  and  $\text{SE.Enc}$  as  $\Lambda.\text{Alg}_\lambda$ , where  $\text{Sub.Enc}$  is the algorithm given in Figure 3.4a. If  $n$  is the number of queries that  $\mathcal{U}$  makes to its encryption oracle and  $\eta$  is the min-entropy of  $\text{SE.Enc}$ , then there is an adversary  $\mathcal{F}$  in the  $\text{PRF}_F(\mathcal{F})$  game such that*

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

*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 3.1 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  $c \leftarrow_s \text{Enc}(k, m)$ , and the **Reset** oracle has no effect.

The first game change is standard, and proceeds exactly as described by [BJK15], with

$$|\Pr[H_1] - \Pr[H_0]| = 2\text{Adv}_F^{\text{PRF}}(\mathcal{F})$$

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

$\text{Sub.Enc}(k, m, \bar{k}, \tau)$	$H_2$	$\mathcal{O}_{\text{Enc}}(k, m)$
1. <b>if</b> $\tau = \perp$ <b>then</b> $\tau \leftarrow 0$ 2. <b>else</b> $\tau \leftarrow \tau + 1 \bmod  k $ 3. $j \leftarrow 0$ 4. <b>do</b> 5. $j \leftarrow j + 1$ 6. $r \leftarrow \{0, 1\}^{\text{SE.rlen}}$ 7. $c \leftarrow \text{Enc}(k, m; r)$ 8. $w \leftarrow F(\bar{k}, c)$ 9. <b>until</b> $k[\tau] = w$ <b>or</b> $j = s$ 10. <b>return</b> $(c, \tau)$ (a) ASA of [BPR14], where $s$ is a predetermined parameter.	1. $\bar{k} \leftarrow \text{Sub.KeyGen}()$ 2. $i \leftarrow 1$ 3. $\tau_0 \leftarrow \perp$ 4. $b \leftarrow \{0, 1\}$ 5. $b' \leftarrow \mathcal{U}^{\mathcal{O}_{\text{Enc}}, \text{Reset}}$ 6. <b>return</b> $b = b'$ <hr/> Reset( $j$ ), $0 \leq j < i$ 1. <b>if</b> $b = 1$ <b>then</b> 2. $\tau_i \leftarrow \tau_j$ 3. $i \leftarrow i + 1$	1. <b>if</b> $b = 0$ <b>then</b> $c \leftarrow \text{Enc}(k, m)$ 2. <b>if</b> $b = 1$ <b>then</b> 3. <b>if</b> $\tau = \perp$ <b>then</b> $\tau = 0$ 4. <b>else</b> $\tau \leftarrow \tau + 1 \bmod  k $ 5. $j \leftarrow 0$ 6. <b>do</b> 7. $j \leftarrow j + 1$ 8. $r \leftarrow \{0, 1\}^{\text{SE.rlen}}$ 9. $c \leftarrow \text{SE.Enc}(k, m; r)$ 10. $w \leftarrow \{0, 1\}$ 11. <b>until</b> $k[\tau] = w$ <b>or</b> $j = s$ 12. $\tau_i \leftarrow \tau; i \leftarrow i + 1$ 13. <b>return</b> $c$

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

Figure 3.4: ASA on symmetric encryption by Bellare, Paterson, and Rogaway [BPR14] and game for proof of its undetectability in Theorem 1.

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  $\eta$  is the min-entropy of  $\text{SE.Enc}$ . 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 3.4b. We argue that  $H_2$  is equivalent to  $H_3$ . In particular, we argue that the implementation of  $\text{Sub.Enc}$  in the encryption oracle of  $H_2$  is identical to  $\text{SE.Enc}$ . 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  $\tau$ , and is in fact the same as simply sampling coins  $r$  and returning the resulting ciphertext  $c$ . This is precisely  $\text{SE.Enc}$ . 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

$$\begin{aligned}
\text{Adv}_{\text{Sub}}^{\text{SRDET}}(\mathcal{U}) &= \left| \Pr[H_0] - \frac{1}{2} \right| \\
&= \left| \Pr[H_0] - \Pr[H_1] + \Pr[H_1] - \Pr[H_2] + \Pr[H_2] - \Pr[H_3] + \Pr[H_3] - \frac{1}{2} \right| \\
&\leq \left| \Pr[H_0] - \Pr[H_1] \right| + \left| \Pr[H_1] - \Pr[H_2] \right| + \left| \Pr[H_2] - \Pr[H_3] \right| + \left| \Pr[H_3] - \frac{1}{2} \right| \\
&\leq 2\text{Adv}_{\mathbb{F}}^{\text{PRF}}(\mathcal{F}) + n^2 s^2 \cdot 2^{-\eta-1},
\end{aligned}$$

as desired.  $\square$

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  $2\text{Adv}_{\mathbb{F}}^{\text{PRF}}(\mathcal{F})$  is small for a good PRF  $\mathbb{F}$ . Hence we can conclude from [Theorem 1](#) that the ASA defined in [Figure 3.4a](#) is undetectable under state resets, even to any prior state.

### 3.3 Discussion

The reader may find the results in this chapter 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 [section 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 [section 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.

## Chapter 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 3.2.

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  $\text{PKE}$  be a public-key encryption scheme, and consider the game in Figure 4.1. In this game, adversary  $\mathcal{B}$  is tasked with deciding whether the oracle provided to it is returning encryptions under  $\text{PKE}.\text{Enc}$  or random bits. The advantage of  $\mathcal{B}$  is defined as

$$\text{Adv}_{\text{PKE}}^{\text{IND\$}}(\mathcal{B}) = |\Pr[\text{IND\$}_{\text{PKE}}(\mathcal{B})] - \frac{1}{2}| \ .$$

Informally, we say that the scheme  $\text{PKE}$  is IND\$-secure if the advantage of any efficient adversary  $\mathcal{B}$  is small.

We present our asymmetric ASA,  $\text{ASub}$ , against a symmetric encryption scheme  $\text{SE}$ , in Figure 4.2. This ASA uses an IND\$-secure public-key encryption scheme  $\text{PKE}$ , 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

$\text{IND}_{\text{PKE}}^{\$}(\mathcal{B})$	$\mathcal{O}_{\text{PKE.Enc}}(m)$
<ol style="list-style-type: none"> <li>1. <math>(pk, sk) \leftarrow \\$ \text{PKE.KeyGen}()</math></li> <li>2. <math>b \leftarrow \\$ \{0, 1\}</math></li> <li>3. <math>b' \leftarrow \\$ \mathcal{B}^{\mathcal{O}_{\text{PKE.Enc}}(pk)}</math></li> <li>4. <b>return</b> <math>b = b'</math></li> </ol>	<ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 0</math> <b>then</b></li> <li>2.   <math>c \leftarrow \\$ \text{PKE.Enc}(pk, m)</math></li> <li>3. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>4.   <math>c \leftarrow \\$ \{0, 1\}^{\text{PKE.clen}}</math></li> <li>5. <b>return</b> <math>c</math></li> </ol>

Figure 4.1: The ciphertext indistinguishability-from-random game for a public-key encryption scheme PKE.

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.

$\text{ASub.Enc}(k, m, ek, \tau)$
<ol style="list-style-type: none"> <li>1. <b>if</b> <math>\tau = \perp</math> <b>then</b></li> <li>2.   <math>\sigma \leftarrow 0; \kappa \leftarrow \\$ \text{PKE.Enc}(ek, k)</math></li> <li>3. <b>else</b> <math>(\sigma, \kappa) \leftarrow \tau</math></li> <li>4. <b>if</b> <math>\sigma = \text{PKE.clen}</math> <b>then</b></li> <li>5.   <math>\sigma \leftarrow 1; \kappa \leftarrow \\$ \text{PKE.Enc}(ek, k)</math></li> <li>6. <b>else</b> <math>\sigma \leftarrow \sigma + 1</math></li> <li>7. <math>j \leftarrow 0</math></li> <li>8. <b>do</b></li> <li>9.   <math>j \leftarrow j + 1</math></li> <li>10.   <math>r \leftarrow \\$ \{0, 1\}^{\text{SE.rlen}}</math></li> <li>11.   <math>c \leftarrow \text{SE.Enc}(k, m; r)</math></li> <li>12.   <math>w \leftarrow \text{F}(ek, c)</math></li> <li>13. <b>until</b> <math>\kappa[\sigma] = w</math> <b>or</b> <math>j = s</math></li> <li>14.   <math>\tau \leftarrow (\sigma, \kappa)</math></li> <li>15. <b>return</b> <math>(c, \tau)</math></li> </ol>

Figure 4.2: Type 1 asymmetric ASA on symmetric encryption.

and hence no third party is able to exploit the ASA either. In Chapter 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

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 chapter, 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  $ek$  is able to detect the ASA,

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  $\text{PKE.Enc}$  is spread out over many calls to  $\text{ASub.Enc}$ , 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  $\text{ASub}$  of [Figure 4.2](#) is undetectable in the augmented state reset detectability game  $\text{ASRDET}$ , when modeling  $F$  as a random oracle.

**Theorem 2.** *Let  $\mathcal{U}$  be an adversary in the augmented state reset detectability game in [Figure 3.1](#),  $\text{ASRDET}_{\text{ASub}}(\mathcal{U})$ , with symmetric encryption scheme  $\text{SE}$  as  $\Lambda$  and  $\text{SE.Enc}$  as  $\Lambda.\text{Alg}_\lambda$ , where  $\text{ASub.Enc}$  is the algorithm given in [Figure 4.2](#). 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  $\eta$  is the min-entropy of  $\text{SE.Enc}$ , then there is an adversary  $\mathcal{B}$  against the  $\text{IND\$}$ -security of  $\text{PKE}$  such that*

$$\text{Adv}_{\text{ASub}}^{\text{ASRDET}}(\mathcal{U}) \leq 2\text{Adv}_{\text{PKE}}^{\text{IND\$}}(\mathcal{B}) + ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1}.$$

*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 3.1](#), 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 4.3](#), as  $G_0$ . The oracle  $H$  only takes input  $c$ , since  $ek$  is fixed throughout the game.

We proceed by a sequence of games  $G_0, \dots, G_4$ , as shown in [Figure 4.3](#). Let  $G_0$  be the undetectability game with the random oracle implementation.  $G_1$  is the same as  $G_0$  but with  $\kappa$  sampled randomly instead of computed as an encryption of  $k$ .  $G_2$  and  $G_3$  are shown in [Figure 4.3](#).  $G_4$  is the augmented state reset detectability game where the encryption oracle is replaced by an oracle that simply returns  $\text{SE.Enc}(k, m)$ .

Let  $\mathcal{B}$ , defined in Figure 4.4, be an adversary to the IND\$ game for PKE. Acting as a challenger,  $\mathcal{B}$  simulates the augmented detection game for  $\mathcal{U}$ , in particular using the PKE.Enc oracle and public key given to it to simulate ASub.Enc. Specifically, instead of PKE.Enc being used in the subverted algorithm,  $\mathcal{B}$  uses its provided oracle.

Let  $b_{\S}$  denote the bit from the  $\text{IND}_{\text{PKE}}^{\$}(\mathcal{B})$  game. Note that if  $b_{\S} = 1$ , then the Enc oracle simulated by  $\mathcal{B}$  proceeds exactly as in game  $G_1$ , and if  $b_{\S} = 0$ , then it proceeds exactly as in game  $G_0$ . Thus we have

$$\begin{aligned}\Pr[\mathcal{B} \Rightarrow 1 \mid b_{\S} = 1] &= \Pr[G_1] \quad , \\ \Pr[\mathcal{B} \Rightarrow 1 \mid b_{\S} = 0] &= \Pr[G_0] \quad ,\end{aligned}$$

and hence

$$|\Pr[G_1] - \Pr[G_0]| = 2\text{Adv}_{\text{PKE}}^{\text{IND}\$}(\mathcal{B}) \quad .$$

Next, consider  $G_2$  shown in Figure 4.3. We claim that  $\Pr[G_2] = \Pr[G_1]$ . To see this, note that lines 1–10 of the **Helper**<sub>1</sub> function simply act as bookkeeping in order to use a single bit of  $\kappa$  for each encryption, and to generate a new  $\kappa$  once we've iterated through its bits. The index  $\sigma$  is used to iterate through  $\kappa$  and  $\kappa$  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 4.3, 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  $\Pr[\text{Col}]$  from above by considering the case where all the random oracle queries happen before the encryption oracle queries. Let  $\eta$  be the min-entropy of SE.Enc. Then we have that

$$\Pr[\text{Col}] \leq \left( \binom{ns + q}{2} - \binom{q}{2} \right) \cdot 2^{-\eta} = ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1} \quad .$$

By the Fundamental Lemma of Game-Playing [BR06], we have

$$|\Pr[G_3] - \Pr[G_2]| \leq \Pr[\text{Col}] \leq ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1} \quad .$$

Finally,  $G_4$  is the detectability game where the encryption oracle is replaced by an oracle that simply returns SE.Enc( $k, m$ ). In game  $G_3$ , since the loop condition is no longer

dependent on the selection of  $c$ , the **Helper**<sub>3</sub> is identical to **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

$$\begin{aligned}
\text{Adv}_{\text{ASub.Enc}}^{\text{ASRDET}}(\mathcal{U}) &= \left| \Pr[G_0] - \frac{1}{2} \right| \\
&= \left| \Pr[G_0] - \Pr[G_1] + \Pr[G_1] - \Pr[G_2] + \Pr[G_2] - \Pr[G_3] + \Pr[G_3] - \frac{1}{2} \right| \\
&\leq \left| \Pr[G_0] - \Pr[G_1] \right| + \left| \Pr[G_1] - \Pr[G_2] \right| + \left| \Pr[G_2] - \Pr[G_3] \right| + \left| \Pr[G_3] - \frac{1}{2} \right| \\
&\leq 2\text{Adv}_{\text{PKE}}^{\text{IND\$}}(\mathcal{B}) + ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1} + \left| \Pr[G_4] - \frac{1}{2} \right| \\
&= 2\text{Adv}_{\text{PKE}}^{\text{IND\$}}(\mathcal{B}) + ((ns)^2 + 2nsq - ns) \cdot 2^{-\eta-1},
\end{aligned}$$

as desired.  $\square$

Assuming that **PKE** is **IND\$**-secure and that  $n$  and  $q$  are small in comparison to  $2^\eta$ , which is the case for a sufficiently randomized encryption scheme, [Theorem 2](#) shows that the **ASA** given by the subversion in [Figure 4.2](#) 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 4.2](#) is to recover the secret key  $k$  used in the encryption algorithm **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  $c_1, \dots, c_n$ .

2. Group ciphertexts together into consecutive groups of size  $\text{PKE.clen}$ .
3. For each ciphertext  $c_\sigma$  in each group, compute  $w_\sigma = F(ek, c_\sigma)$ .
4. For each resulting group of bits, concatenate all the  $w_\sigma$  to obtain  $\kappa$  and compute  $k' = \text{PKE.Dec}(xk, \kappa)$ .
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  $\text{SE.Enc}$  so that no two of the ciphertexts  $c_1, \dots, c_n$  are the same. The effect of these assumptions can be quantified<sup>1</sup>, but we will omit those details here, and simply note that such quantification will result in  $2\text{Adv}_F^{\text{PRF}}(\mathcal{F}) + n^2 s^2 \cdot 2^{-\eta-1}$ , where  $\eta$  is the min-entropy of  $\text{SE.Enc}$  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  $\text{PKE}$  is a  $\delta$ -correct public-key encryption scheme. Let  $n$  be the total number of ciphertexts intercepted by the subverter. For each group of  $\text{PKE.clen}$  ciphertexts (of which there are  $\lfloor n/\text{PKE.clen} \rfloor$ ), the probability that every ciphertext  $c$  in the group was chosen to successfully leak a bit ( $\kappa[\sigma] = F(ek, c)$ ) is  $(1 - 2^{-s})^{\text{PKE.clen}}$  (this is where we assume each  $w$  is random), and the probability that the group decrypts correctly is  $\delta$ . Hence the probability that one of the keys  $k'$  obtained in step 5 is the key  $k$  is

$$P_{kr} = 1 - \left(1 - \delta(1 - 2^{-s})^{\text{PKE.clen}}\right)^{\lfloor \frac{n}{\text{PKE.clen}} \rfloor}.$$

As an example, suppose  $\text{PKE.clen} = 400$  (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  $\delta$  is lower,  $\text{PKE.clen}$  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  $\text{PKE}$  ciphertext  $\kappa$  must be exfiltrated before the subverter is

---

<sup>1</sup>There are several examples of this in the literature, as well as in our proof of [Theorem 2](#) in the previous section.

able to decrypt it to recover  $k$ , reset of state would restart the process of key recovery. Since  $\kappa$  is required to be indistinguishable from random, we do not have the option of reconstructing the same  $\kappa$  after a state reset. With `PKE.clen` = 400, a state reset after every 399 encryptions would successfully thwart this ASA.

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.



$G_{0,1,2,3,4}(\mathcal{U})$

1.  $ek, xk \leftarrow \text{ASub.KeyGen}()$
2.  $C \leftarrow \emptyset$
3.  $\boxed{i \leftarrow 1}_{0,1}$
4.  $\boxed{\tau_0 \leftarrow \perp}_{0,1}$
5.  $b \leftarrow \{0, 1\}$
6.  $\boxed{b' \leftarrow \mathcal{U}^{\mathcal{O}_{\text{Enc}}, \text{Reset}, \text{H}}(ek)}_{0,1}$
7.  $\boxed{b' \leftarrow \mathcal{U}^{\mathcal{O}_{\text{Enc}}, \text{H}}(ek)}_{2,3}$
8.  $\boxed{b' \leftarrow \mathcal{U}^{\text{SE.Enc}, \text{H}}(ek)}_4$
9. **return**  $b = b'$

$\text{Reset}(j), 0 \leq j < i$

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

$\text{H}(c)$

1. **if**  $c \notin C$  **then**
2.    $w_c \leftarrow \{0, 1\}$
3.    $C \leftarrow C \cup \{c\}$
4. **return**  $w_c$

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

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

$\text{Helper}_{0/1}(k, m, ek, \tau)$

1. **if**  $\tau = \perp$  **then**
2.    $\boxed{\kappa \leftarrow \text{PKE.Enc}(ek, k)}_0$
3.    $\boxed{\kappa \leftarrow \{0, 1\}^{\text{PKE.clen}}}_1$
4.    $\sigma \leftarrow 0$
5. **else**  $(\sigma, \kappa) \leftarrow \tau$
6. **if**  $\sigma = \text{PKE.clen}$  **then**
7.    $\sigma \leftarrow 1$
8.    $\boxed{\kappa \leftarrow \text{PKE.Enc}(ek, k)}_0$
9.    $\boxed{\kappa \leftarrow \{0, 1\}^{\text{PKE.clen}}}_1$
10. **else**  $\sigma \leftarrow \sigma + 1$
11.    $j \leftarrow 0$
12.   **do**
13.      $j \leftarrow j + 1$
14.      $r \leftarrow \{0, 1\}^{\text{SE.rlen}}$
15.      $c \leftarrow \text{SE.Enc}(k, m; r)$
16.     **if**  $c \notin C$  **then**
17.        $w_c \leftarrow \{0, 1\}$
18.        $C \leftarrow C \cup \{c\}$
19.        $w \leftarrow w_c$
20.     **until**  $\kappa[\sigma] = w$  **or**  $j = s$
21.      $\tau \leftarrow (\sigma, \kappa)$
22. **return**  $(c, \tau)$

$\text{Helper}_{2/3}(k, m, ek)$

1.  $\kappa \leftarrow \{0, 1\}$
2.  $j \leftarrow 0$
3. **do**
4.    $j \leftarrow j + 1$
5.    $r \leftarrow \{0, 1\}^{\text{SE.rlen}}$
6.    $c \leftarrow \text{SE.Enc}(k, m; r)$
7.    $\boxed{\text{if } c \notin C \text{ then}}_2$
8.      $w_c \leftarrow \{0, 1\}$
9.      $C \leftarrow C \cup \{c\}$
10.    $\boxed{\text{else bad} \leftarrow \text{true}}_2$
11.    $w \leftarrow w_c$
12. **until**  $\kappa = w$  **or**  $j = s$
13. **return**  $c$

Figure 4.3: 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.

$\mathcal{B}^{\mathcal{O}_{\text{PKE.Enc}}}(pk)$ <hr/> <ol style="list-style-type: none"> <li>1. <math>i \leftarrow 1</math></li> <li>2. <math>\tau_0 \leftarrow \perp</math></li> <li>3. <math>C \leftarrow \emptyset</math></li> <li>4. <math>b_{\text{det}} \leftarrow \{0, 1\}</math></li> <li>5. <math>b'_{\text{det}} \leftarrow \mathcal{U}^{\mathcal{O}_{\text{Enc}}, \text{Reset}, \text{H}}(pk)</math></li> <li>6. <b>if</b> <math>b_{\text{det}} = b'_{\text{det}}</math></li> <li>7.     <b>return</b> 1</li> <li>8. <b>else</b></li> <li>9.     <b>return</b> 0</li> </ol>	$\mathcal{O}_{\text{Enc}}(k, m)$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b_{\text{det}} = 0</math> <b>then</b></li> <li>2.     <math>c \leftarrow \text{SE.Enc}(k, m)</math></li> <li>3. <b>if</b> <math>b_{\text{det}} = 1</math> <b>then</b></li> <li>4.     <math>(c, \tau_i) \leftarrow \text{Helper}(k, m, pk, \tau_{i-1})</math></li> <li>5.     <math>i \leftarrow i + 1</math></li> <li>6. <b>return</b> <math>c</math></li> </ol>
$\text{Reset}(j), 0 \leq j < i$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b_{\text{det}} = 1</math> <b>then</b></li> <li>2.     <math>\tau_i \leftarrow \tau_j</math></li> <li>3.     <math>i \leftarrow i + 1</math></li> </ol>	$\text{Helper}(k, m, pk, \tau)$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>\tau = \perp</math> <b>then</b></li> <li>2.     <div style="border: 1px solid black; padding: 2px;"><math>\kappa \leftarrow \mathcal{O}_{\text{PKE.Enc}}(k)</math></div></li> <li>3.     <math>\sigma \leftarrow 0</math></li> <li>4. <b>else</b></li> <li>5.     <math>(\sigma, \kappa) \leftarrow \tau</math></li> <li>6. <b>if</b> <math>\sigma = \text{PKE.clen}</math> <b>then</b></li> <li>7.     <math>\sigma \leftarrow 1</math></li> <li>8.     <div style="border: 1px solid black; padding: 2px;"><math>\kappa \leftarrow \mathcal{O}_{\text{PKE.Enc}}(k)</math></div></li> <li>9. <b>else</b></li> <li>10.     <math>\sigma \leftarrow \sigma + 1</math></li> <li>11.     <math>j \leftarrow 0</math></li> <li>12. <b>do</b></li> <li>13.     <math>j \leftarrow j + 1</math></li> <li>14.     <math>r \leftarrow \{0, 1\}^{\text{SE.rlen}}</math></li> <li>15.     <math>c \leftarrow \text{SE.Enc}(k, m; r)</math></li> <li>16.     <b>if</b> <math>c \notin C</math> <b>then</b></li> <li>17.         <math>w_c \leftarrow \{0, 1\}</math></li> <li>18.         <math>C \leftarrow C \cup \{c\}</math></li> <li>19.     <math>w \leftarrow w_c</math></li> <li>20.     <b>until</b> <math>\kappa[\sigma] = w</math> <b>or</b> <math>j = s</math></li> <li>21.     <math>\tau \leftarrow (\sigma, \kappa)</math></li> <li>22. <b>return</b> <math>(c, \tau)</math></li> </ol>
$\text{H}(c)$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>c \notin C</math> <b>then</b></li> <li>2.     <math>w_c \leftarrow \{0, 1\}</math></li> <li>3.     <math>C \leftarrow C \cup \{c\}</math></li> <li>4. <b>return</b> <math>w_c</math></li> </ol>	

Figure 4.4: Adversary  $\mathcal{B}$  for the proof of [Theorem 2](#).

## Chapter 5

# A Type 2 Asymmetric ASA on Symmetric Encryption

In the last chapter, 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 chapter 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 3.1](#), 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  $\mathbf{ASub.Alg}_\lambda$  preserve security properties of the original algorithm  $\mathbf{A.Alg}_\lambda$ .<sup>1</sup>

The situation described above admittedly has stronger behavioural assumptions on the involved parties than the one we covered in [Chapter 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 [Chapter 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 asymmetric encryption. This is a weaker form of security than IND\$ we used in [Chapter 4](#) (which implicitly was in chosen-plaintext form), but it is all we will require in this chapter.

The IND-CPA games for symmetric encryption scheme  $\mathbf{SE}$  and public-key encryption scheme  $\mathbf{PKE}$  are given in [Figure 5.1](#).

The advantage of  $\mathcal{B}$  at each of these games is

$$\text{Adv}_{\mathbf{PKE}}^{\text{IND-CPA}}(\mathcal{B}) = |\Pr[\text{IND-CPA}_{\mathbf{PKE}}(\mathcal{B})] - \frac{1}{2}|$$

and

$$\text{Adv}_{\mathbf{SE}}^{\text{IND-CPA}}(\mathcal{B}) = |\Pr[\text{IND-CPA}_{\mathbf{SE}}(\mathcal{B})] - \frac{1}{2}|.$$

Informally, we say that the schemes  $\mathbf{PKE}$  and  $\mathbf{SE}$  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  $\mathbf{ASub2}$ . Let  $\mathbf{SE}$  be a symmetric encryption scheme. Let  $\mathbf{PKE}$  be an IND-CPA-secure public-key encryption scheme, and let  $\mathbf{F}$  be a

---

<sup>1</sup>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  $\bar{k}$ . However, they only considered symmetric ASAs; the asymmetric case is certainly different.

IND-CPA <sub>SE</sub> ( $\mathcal{B}$ )	$\mathcal{O}_{\text{SE.Enc}}(m)$	IND-CPA <sub>PKE</sub> ( $\mathcal{B}$ )	$\mathcal{O}_{\text{PKE.Enc}}(m)$
1. $k \leftarrow \$ \text{SE.KeyGen}()$ 2. $b \leftarrow \$ \{0, 1\}$ 3. $b' \leftarrow \$ \mathcal{B}^{\mathcal{O}_{\text{SE.Enc}}}$ 4. <b>return</b> $b = b'$	1. <b>if</b> $b = 0$ <b>then</b> 2. $c \leftarrow \$ \text{SE.Enc}(k, m)$ 3. <b>if</b> $b = 1$ <b>then</b> 4. $c \leftarrow \$ \text{SE.Enc}(k, 0)$ 5. <b>return</b> $c$	1. $(pk, sk) \leftarrow \$ \text{PKE.KeyGen}()$ 2. $b \leftarrow \$ \{0, 1\}$ 3. $b' \leftarrow \$ \mathcal{B}^{\mathcal{O}_{\text{PKE.Enc}}(pk)}$ 4. <b>return</b> $b = b'$	1. <b>if</b> $b = 0$ <b>then</b> 2. $c \leftarrow \$ \text{PKE.Enc}(pk, m)$ 3. <b>if</b> $b = 1$ <b>then</b> 4. $c \leftarrow \$ \text{PKE.Enc}(pk, 0)$ 5. <b>return</b> $c$

Figure 5.1: The IND-CPA games for (left) a symmetric encryption scheme SE and (right) a public-key encryption scheme PKE.

PRF with output space  $\{1, \dots, \text{PKE.clen}\} \times \{0, 1\}$ . Then ASUB2, an ASA on SE is shown in Figure 5.2, 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 Chapter 4 and to the ASA of [BJK15]. In fact, the ASA ASUB essentially consists of encrypting  $k$  to  $\kappa$  using IND\$-secure public-key encryption, and then leaking  $\kappa$  using the same techniques as [BPR14]. The ASA ASUB2, on the other hand, consists of encrypting  $k$  to  $\kappa$  using IND-CPA-secure public-key encryption, and then leaking  $\kappa$  using the same techniques as [BJK15]. We will explore some of these parallels in Chapter 6.

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  $\kappa$ , and the subverter can obtain  $k$  from  $\kappa$  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  $\kappa$  they obtain by this process from random bits. Making sure to never leak the same bit of  $\kappa$  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  $\kappa$  for more than  $|\kappa|$  encryptions is required: randomization of the leaked bit of  $\kappa$  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 key in the

ASUB2.Enc( $k, m, ek, \tau$ )
1. <b>if</b> $\tau = \perp$ <b>then</b> 2. $\kappa \leftarrow \$ \text{PKE.Enc}(ek, k); \tau \leftarrow \kappa$ 3. <b>else</b> $\kappa \leftarrow \tau$ 4. $j \leftarrow 0$ 5. <b>do</b> 6. $j \leftarrow j + 1$ 7. $r \leftarrow \$ \{0, 1\}^{\text{SE.rlen}}$ 8. $c \leftarrow \text{SE.Enc}(k, m; r)$ 9. $(\sigma, w) \leftarrow \text{F}(ek, c)$ 10. <b>until</b> $\kappa[\sigma] = w$ <b>or</b> $j = s$ 11. <b>return</b> $(c, \tau)$

Figure 5.2: Type 2 asymmetric ASA on symmetric encryption.

same way as the subverter.

In the rest of this chapter, 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 3.1](#),  $\text{SRDET}_{\text{ASub2}}(\mathcal{U})$ , with symmetric encryption scheme SE as  $\Lambda$  and SE.Enc as  $\Lambda.\text{Alg}_\lambda$ , where ASub2.Enc is the algorithm given in [Figure 5.2](#). If  $n$  is the number of queries that  $\mathcal{U}$  makes to its encryption oracle, and  $\eta$  is the min-entropy of SE.Enc, then there is an adversary  $\mathcal{F}$  against the PRF-security of  $\mathbf{F}$  such that*

$$\text{Adv}_{\text{ASub}}^{\text{SRDET}}(\mathcal{U}) \leq 2\text{Adv}_{\mathbf{F}}^{\text{PRF}}(\mathcal{F}) + (ns)^2 \cdot 2^{-\eta-1} .$$

*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 3.1](#), with all the substitutions in the theorem statement. Let  $I_1$  be the same as  $I_0$  but with  $\mathbf{F}$  replaced by lazy random sampling; this is given in [Figure 5.3](#). Let  $I_2$  be the same as  $I_1$  but with  $w$  and  $\sigma$  sampled randomly instead; this is also given in [Figure 5.3](#). Let  $I_3$  be the regular state reset detectability game where the encryption oracle is replaced by an oracle that simply returns  $\text{SE.Enc}(k, m)$ , 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  $\mathbf{F}$ , so we omit the detailed reduction. Instead, note that an adversary  $\mathcal{F}$  in the  $\text{PRF}_{\mathbf{F}}(\mathcal{F})$  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  $\mathbf{F}$ . Let  $b_{\text{PRF}}$  be the challenge bit in the PRF game. Then if  $b_{\text{PRF}} = 1$ , the **Enc** oracle simulated by  $\mathcal{F}$  proceeds exactly as in game  $I_1$ , and if  $b_{\text{PRF}} = 0$ , then it proceeds exactly as in game  $I_0$ . Thus we have

$$\Pr[\mathcal{F} \Rightarrow 1 \mid b_{\text{PRF}} = 1] = \Pr[I_1] ,$$

$I_{1/2}(\mathcal{U})$ <hr/> 1. $(ek, xk) \leftarrow \text{\$} \text{ASub2.KeyGen}()$ 2. $C \leftarrow \emptyset$ 3. $i \leftarrow 1$ 4. $\tau_0 \leftarrow \perp$ 5. $b \leftarrow \text{\$} \{0, 1\}$ 6. $b' \leftarrow \text{\$} \mathcal{U}^{\text{O}_{\text{Enc}}, \text{Reset}}$ 7. <b>return</b> $b = b'$  $\text{Reset}(j), 0 \leq j < i$ <hr/> 1. <b>if</b> $b = 1$ <b>then</b> 2. $\tau_i \leftarrow \tau_j$ 3. $i \leftarrow i + 1$  $\text{O}_{\text{Enc}}(k, m)$ <hr/> 1. <b>if</b> $b = 0$ <b>then</b> $c \leftarrow \text{\$} \text{Enc}(k, m)$ 2. <b>if</b> $b = 1$ <b>then</b> 3. $(c, \tau_i) \leftarrow \text{\$} \text{Helper}(k, m, ek, \tau_{i-1})$ 4. $i \leftarrow i + 1$ 5. <b>return</b> $c$	$\text{Helper}(k, m, ek, \tau)$ <hr/> 1. <b>if</b> $\tau = \perp$ <b>then</b> 2. $\kappa \leftarrow \text{\$} \text{PKE.Enc}(ek, k); \tau \leftarrow \kappa$ 3. <b>else</b> $\kappa \leftarrow \tau$ 4. $j \leftarrow 0$ 5. <b>do</b> 6. $j \leftarrow j + 1$ 7. $r \leftarrow \text{\$} \{0, 1\}^{\text{SE.rlen}}$ 8. $c \leftarrow \text{SE.Enc}(k, m; r)$ 9. <div style="border: 1px solid black; padding: 2px; display: inline-block;"> <b>if</b> <math>c \notin C</math> <b>then</b>  10.         <math>(\sigma_c, w_c) \leftarrow \text{\\$} \{0, \dots, \text{PKE.clen}\} \times \{0, 1\}</math>  11.         <math>C \leftarrow C \cup \{c\}</math>  12.         <div style="border: 1px solid black; padding: 2px; display: inline-block;"> <b>else bad</b> <math>\leftarrow \text{true}</math> </div>  13.         <math>\sigma, w \leftarrow \sigma_c, w_c</math>  14.     <b>until</b> <math>\kappa[\sigma] = w</math> <b>or</b> <math>j = s</math>  15. <b>return</b> <math>(c, \tau)</math> </div>
--	--

Figure 5.3: 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).

$$\Pr[\mathcal{F} \Rightarrow 1 \mid b_{\text{PRF}} = 0] = \Pr[I_0] \quad ,$$

and hence

$$|\Pr[I_1] - \Pr[I_0]| = 2\text{Adv}_{\text{PKE}}^{\text{PRF}}(\mathcal{F}) \quad .$$

Now consider games  $I_1$  and  $I_2$ . In game  $I_2$ , we replace the selection of  $w$  and  $\sigma$  in the encryption oracle with true random sampling of  $w$  and  $\sigma$ , 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  $\Pr[\text{Col}]$ : let  $\eta$  be the min-entropy of  $\text{SE.Enc}$ . Then we have that

$$\Pr[\text{Col}] \leq \binom{ns}{2} \cdot 2^{-\eta} \leq (ns)^2 \cdot 2^{-\eta-1} \quad .$$

By the Fundamental Lemma of Game-Playing [BR06], we have

$$|\Pr[I_2] - \Pr[I_1]| \leq \Pr[\text{Col}] \leq (ns)^2 \cdot 2^{-\eta-1} \quad .$$

Finally,  $I_3$  was defined as the detectability game where the encryption oracle is replaced by an oracle that simply returns  $\text{SE.Enc}(k, m)$ . In game  $I_2$ , since the loop condition is no longer dependent on the selection of  $c$ , the implementation  $\text{ASub2.Enc}_2$  is identical to  $\text{SE.Enc}(k, m)$ . 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

$$\begin{aligned} \text{Adv}_{\text{ASub2}}^{\text{SRDET}}(\mathcal{U}) &= |\Pr[I_0] - \frac{1}{2}| \\ &= |\Pr[I_0] - \Pr[I_1] + \Pr[I_1] - \Pr[I_2] + \Pr[I_2] - \frac{1}{2}| \\ &\leq |\Pr[I_0] - \Pr[I_1]| + |\Pr[I_1] - \Pr[I_2]| + |\Pr[I_2] - \frac{1}{2}| \\ &\leq 2\text{Adv}_{\text{F}}^{\text{PRF}}(\mathcal{F}) + (ns)^2 \cdot 2^{-\eta-1} + |\Pr[I_3] - \frac{1}{2}| \\ &= 2\text{Adv}_{\text{F}}^{\text{PRF}}(\mathcal{F}) + (ns)^2 \cdot 2^{-\eta-1}, \end{aligned}$$

as desired. □



$\text{IND-CPA}'_{\text{ASub2}}(\mathcal{V})$	$\mathcal{O}_{\text{ASub2.Enc}}(m)$
1. $(ek, xk) \leftarrow \text{ASub2.KeyGen}()$	1. <b>if</b> $b = 0$ <b>then</b>
2. $k \leftarrow \text{SE.KeyGen}()$	2. $(c, \tau) \leftarrow \text{ASub2.Enc}(k, m, ek, \tau)$
3. $\tau \leftarrow \perp$	3. <b>if</b> $b = 1$ <b>then</b>
4. $b \leftarrow \{0, 1\}$	4. $(c, \tau) \leftarrow \text{ASub2.Enc}(k, \mathbf{0}, ek, \tau)$
5. $b' \leftarrow \mathcal{V}^{\mathcal{O}_{\text{ASub2.Enc}}(ek)}$	5. <b>return</b> $c$
6. <b>return</b> $b = b'$	

Figure 5.4: The IND-CPA' game for asymmetric ASA ASub2.

## 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 5.4. This game contains the required modifications to Figure 5.1 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 5.2. Let  $\mathcal{V}$  be an adversary in the game  $\text{IND-CPA}'_{\text{ASub2.Enc}}(\mathcal{V})$ . 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*

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

*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  $\text{IND-CPA}'_{\text{ASub2.Enc}}(\mathcal{V})$  game. Let  $J_1$  be the same as  $J_0$  but with  $\kappa$  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 5.5.

Let  $\mathcal{B}_1$ , defined in Figure 5.5, 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  $b_{\text{pke}}$  denote the bit from the  $\text{IND-CPA}_{\text{PKE}}(\mathcal{B}_1)$  game. Note that if  $b_{\text{pke}} = 1$ , then the **Enc** oracle simulated by  $\mathcal{B}_1$  proceeds exactly as in game  $J_1$ , and if  $b_{\text{pke}} = 0$ , then it proceeds exactly as in game  $J_0$ . Thus we have

$$\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{\text{pke}} = 1] = \Pr[J_1] \quad ,$$

$$\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{\text{pke}} = 0] = \Pr[J_0] \quad ,$$

and hence

$$|\Pr[J_1] - \Pr[J_0]| = 2\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(\mathcal{B}_1) \quad .$$

Let  $\mathcal{B}_2$ , defined in Figure 5.5, be an adversary to the  $\text{IND-CPA}$  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**<sub>1</sub>.

Game  $J_2$  is given in Figure 5.5. The only change from  $J_1$  is that the oracle will now always use **0** in the place of  $m$ .

Let  $b_{\text{se}}$  denote the bit from the  $\text{IND-CPA}_{\text{SE}}(\mathcal{B}_2)$  game. Note that if  $b_{\text{se}} = 1$ , then the **Enc** oracle simulated by  $\mathcal{B}_2$  proceeds exactly as in game  $J_2$ , and if  $b_{\text{se}} = 0$ , then it proceeds exactly as in game  $J_1$ . Thus we have

$$\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{\text{se}} = 1] = \Pr[J_2] \quad ,$$

$$\Pr[\mathcal{B}_1 \Rightarrow 1 \mid b_{\text{se}} = 0] = \Pr[J_1] \quad ,$$

and hence

$$|\Pr[J_2] - \Pr[J_1]| = 2\text{Adv}_{\text{SE}}^{\text{IND-CPA}}(\mathcal{B}_2) \quad .$$

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

$$\begin{aligned} \text{Adv}_{\text{ASub2}}^{\text{IND-CPA}}(\mathcal{V}) &= |\Pr[J_0] - \frac{1}{2}| \\ &= |\Pr[J_0] - \Pr[J_1] + \Pr[J_1] - \Pr[J_2] + \Pr[J_2] - \frac{1}{2}| \\ &\leq |\Pr[J_0] - \Pr[J_1]| + |\Pr[J_1] - \Pr[J_2]| + |\Pr[J_2] - \frac{1}{2}| \\ &\leq 2\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(\mathcal{B}_1) + 2\text{Adv}_{\text{SE}}^{\text{IND-CPA}}(\mathcal{B}_2), \end{aligned}$$

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

The probability of success here can be calculated by making use of a coupon-collector problem analysis.<sup>2</sup> Suppose PKE is  $\delta$ -correct. The probability that every ciphertext was chosen to successfully leak a bit ( $F(ek, c) = \kappa[\sigma]$ ) is  $(1 - 2^{-s})^n$ . Let  $\ell$  be the length of  $\kappa$ . Then the probability that a given index of  $\kappa$  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  $\kappa$  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  $\text{PKE.clen} \approx 400$ , 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

---

<sup>2</sup>[https://en.wikipedia.org/wiki/Coupon\\_collector%27s\\_problem](https://en.wikipedia.org/wiki/Coupon_collector%27s_problem)

ciphertext  $c$  is returned such that  $F(ek, c) \neq \kappa[\sigma]$ , 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  $1 - 2^{-s}$ . Pseudocode for our simulation is given in [Figure 5.6](#). With values  $klen = 400, n = 14000$ , and  $s = 2$ , 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 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 5.7](#). 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 [Chapter 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 Chapter 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 Chapter 4, regular state resets would reduce key recovery probability of the ASA in this chapter 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  $\kappa$  can be recomputed without state (keeping state anyway would allow the ASA to do the work of recomputing  $\kappa$  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 5.7 would converge to 700 instead of 2600.

Unfortunately, deterministic PKE causes difficulties for our security analysis in sec-

tion 5.2. Theorem 4 bounds the IND-CPA' advantage against 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.

$J_{0,1,2}(\mathcal{V})$ <hr/> 1. $(ek, xk) \leftarrow \$ \text{ASub2.KeyGen}()$ 2. $k \leftarrow \$ \text{SE.KeyGen}()$ 3. $\tau \leftarrow \perp$ 4. $b \leftarrow \$ \{0, 1\}$ 5. $b' \leftarrow \$ \mathcal{V}^{\mathcal{O}_{\text{Enc}}}(ek)$ 6. <b>return</b> $b = b'$	$\mathcal{B}_1^{\mathcal{O}_{\text{PKE.Enc}}}(pk)$ <hr/> 1. $k \leftarrow \$ \text{SE.KeyGen}()$ 2. $\tau \leftarrow \perp$ 3. $b_1 \leftarrow \$ \{0, 1\}$ 4. $b'_1 \leftarrow \$ \mathcal{V}^{\mathcal{O}_{\text{Enc}}}(pk)$ 5. <b>if</b> $b_1 = b'_1$ <b>return</b> 1 6. <b>else return</b> 0	$\mathcal{B}_2^{\mathcal{O}_{\text{SE.Enc}}}$ <hr/> 1. $(ek, xk) \leftarrow \$ \text{PKE.KeyGen}()$ 2. $\tau \leftarrow \perp$ 3. $b_2 \leftarrow \$ \{0, 1\}$ 4. $b'_2 \leftarrow \$ \mathcal{V}^{\mathcal{O}_{\text{Enc}}}(ek)$ 5. <b>if</b> $b_2 = b'_2$ <b>return</b> 1 6. <b>else return</b> 0
$\mathcal{O}_{\text{Enc}}(m)$ <hr/> 1. <b>if</b> $b = 0$ <b>then</b> 2. $(c, \tau) \leftarrow \$ \text{Helper}(k, m, ek, \tau)$ 3. <b>if</b> $b = 1$ <b>then</b> 4. $(c, \tau) \leftarrow \$ \text{Helper}(k, \mathbf{0}, ek, \tau)$ 5. <b>return</b> $c$	$\mathcal{O}_{\text{Enc}}(m)$ <hr/> 1. <b>if</b> $b_1 = 0$ <b>then</b> 2. $(c, \tau) \leftarrow \$ \text{Helper}(k, m, pk, \tau)$ 3. <b>if</b> $b_1 = 1$ <b>then</b> 4. $(c, \tau) \leftarrow \$ \text{Helper}(k, \mathbf{0}, pk, \tau)$ 5. <b>return</b> $c$	$\mathcal{O}_{\text{Enc}}(m)$ <hr/> 1. <b>if</b> $b_2 = 0$ <b>then</b> 2. $(c, \tau) \leftarrow \$ \text{Helper}(k, m, ek, \tau)$ 3. <b>if</b> $b_2 = 1$ <b>then</b> 4. $(c, \tau) \leftarrow \$ \text{Helper}(k, \mathbf{0}, ek, \tau)$ 5. <b>return</b> $c$
$\text{Helper}(k, m, ek, \tau)$ <hr/> 1. <b>if</b> $\tau = \perp$ <b>then</b> 2. $\kappa \leftarrow \$ \text{PKE.Enc}(ek, k) \bigg _0$ 3. $\kappa \leftarrow \$ \text{PKE.Enc}(ek, \mathbf{0}) \bigg _{1,2}$ 4. $\tau \leftarrow \kappa$ 5. <b>else</b> $\kappa \leftarrow \tau$ 6. $j \leftarrow 0$ 7. <b>do</b> 8. $j \leftarrow j + 1$ 9. $r \leftarrow \$ \{0, 1\}^{\text{SE.rlen}}$ 10. $c \leftarrow \text{SE.Enc}(k, m; r) \bigg _{0,1}$ 11. $c \leftarrow \text{SE.Enc}(k, \mathbf{0}; r) \bigg _2$ 12. $(\sigma, w) \leftarrow \text{F}(ek, c)$ 13. <b>until</b> $\kappa[\sigma] = w$ <b>or</b> $j = s$ 14. <b>return</b> $(c, \tau)$	$\text{Helper}(k, m, pk, \tau)$ <hr/> 1. <b>if</b> $\tau = \perp$ <b>then</b> 2. $\kappa \leftarrow \$ \mathcal{O}_{\text{PKE.Enc}}(k) \bigg _0$ 3. $\tau \leftarrow \kappa$ 4. <b>else</b> $\kappa \leftarrow \tau$ 5. $j \leftarrow 0$ 6. <b>do</b> 7. $j \leftarrow j + 1$ 8. $r \leftarrow \$ \{0, 1\}^{\text{SE.rlen}}$ 9. $c \leftarrow \text{SE.Enc}(k, m; r)$ 10. $(\sigma, w) \leftarrow \text{F}(\overline{pk}, c)$ 11. <b>until</b> $\kappa[\sigma] = w$ <b>or</b> $j = s$ 12. <b>return</b> $(c, \tau)$	$\text{Helper}(k, m, ek, \tau)$ <hr/> 1. <b>if</b> $\tau = \perp$ <b>then</b> 2. $\kappa \leftarrow \$ \text{PKE.Enc}(ek, \mathbf{0})$ 3. $\tau \leftarrow \kappa$ 4. <b>else</b> $\kappa \leftarrow \tau$ 5. $j \leftarrow 0$ 6. <b>do</b> 7. $j \leftarrow j + 1$ 8. $c \leftarrow \mathcal{O}_{\text{SE.Enc}}(m)$ 9. $(\sigma, w) \leftarrow \text{F}(ek, c)$ 10. <b>until</b> $\kappa[\sigma] = w$ <b>or</b> $j = s$ 11. <b>return</b> $(c, \tau)$

Figure 5.5: 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.

### Simulation( $klen, n, s, num$ )

---

```

1. successes  $\leftarrow 0$ 
2. repeat  $num$  times
3.   for  $i$  in  $\{1, \dots, klen\}$  do
4.     recovered-key[ $i$ ]  $\leftarrow (0, 0)$ 
5.   repeat  $n$  times
6.      $r_1 \leftarrow \mathcal{s} \{1, \dots, klen\}$ 
7.      $r_2 \leftarrow \mathcal{s} \{1, \dots, 2^s\}$ 
8.     if  $r_2 \neq 1$  then
9.       recovered-key[ $r_1$ ][1]  $\leftarrow$  recovered-key[ $r_1$ ][1] + 1
10.      recovered-key[ $r_1$ ][2]  $\leftarrow$  recovered-key[ $r_1$ ][2] + 1
11.   succ  $\leftarrow True$ 
12.   for  $i$  in  $\{1, \dots, n\}$  do
13.     if recovered-key[ $i$ ][1]  $\leq$  recovered-key[ $i$ ][2]/2 then
14.       succ  $\leftarrow False$ 
15.   if succ then
16.     successes  $\leftarrow$  successes + 1
17. return successes/ $num$ 

```

Figure 5.6: Simulation for improved practical key recovery for a randomized ASA. For each of  $n$  ciphertexts, an encoding error occurs with probability  $2^{-s}$ . If more than half of the samples for a given index of a key of length  $klen$  encounter encoding errors, then the trial fails. After  $num$  trials, the returned value is the fraction which succeeded.

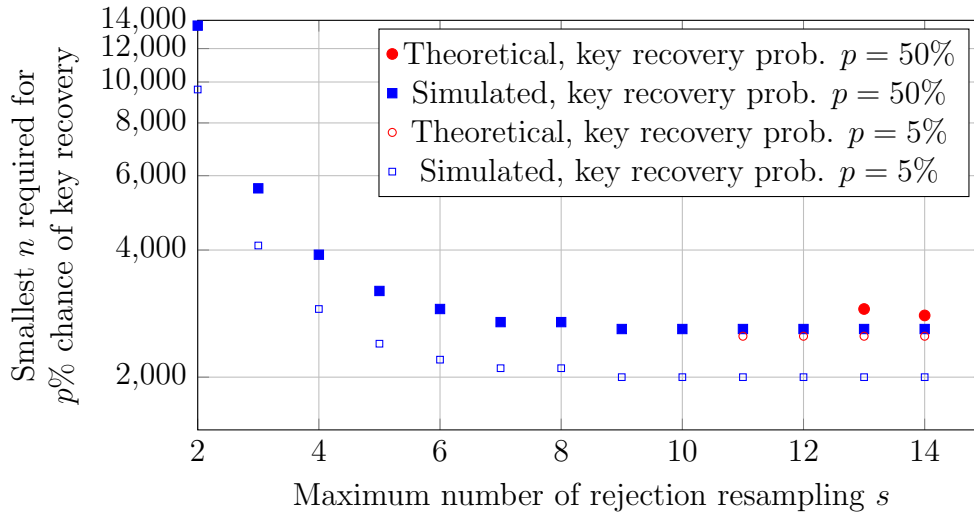


Figure 5.7: Key recovery parameter tradeoff, showing the smallest number of ciphertexts  $n$  (rounded up to the nearest 100) that result in 50% and 5% probability of key recovery, for  $\kappa$  of length 400, and the value of  $s$  on the x-axis.



# Chapter 6

## Generalized Modifications to Obtain Asymmetric ASAs

In this chapter, we will explore how the techniques of [Chapter 4](#) and [Chapter 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 chapter, 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  $\text{fSub}$  of a cryptographic scheme  $\Lambda$  is an ASA that satisfies these additional constraints:

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

We define the regular and augmented state reset detectability games  $\text{SRDET}_{\text{fSub}}(\mathcal{U})$  and  $\text{ASRDET}_{\text{fSub}}(\mathcal{U})$  for a flexible ASA  $\text{fSub}$  in [Figure 6.1](#). These are modifications to the

$\boxed{\text{ASRDET}}_{\text{fSub}}(\mathcal{U})$ <hr/> <ol style="list-style-type: none"> <li>1. <math>(xk, ek) \leftarrow \text{fSub.KeyGen}()</math></li> <li>2. <math>i \leftarrow 1</math></li> <li>3. <math>\tau_0 \leftarrow \perp</math></li> <li>4. <math>b \leftarrow \{0, 1\}</math></li> <li>5. <math>b' \leftarrow \mathcal{U}^{\mathcal{O}_{\text{Alg}_\lambda}, \text{Reset}}(\boxed{ek})</math></li> <li>6. <b>return</b> <math>b = b'</math></li> </ol> $\text{Reset}(j), 0 \leq j < i$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>2.     <math>\tau_i \leftarrow \tau_j</math></li> <li>3.     <math>i \leftarrow i + 1</math></li> </ol> $\mathcal{O}_{\text{Alg}_\lambda}(x, \mu)$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 0</math> <b>then</b></li> <li>2.     <math>y \leftarrow \Lambda.\text{Alg}_\lambda(x)</math></li> <li>3. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>4.     <math>(y, \tau_i) \leftarrow \text{fSub.Alg}_\lambda(x, ek, \tau_{i-1}, \mu)</math></li> <li>5.     <math>i \leftarrow i + 1</math></li> <li>6. <b>return</b> <math>y</math></li> </ol>	$\text{ASRDET\$}_{\text{fSub}}(\mathcal{U})$ <hr/> <ol style="list-style-type: none"> <li>1. <math>(xk, ek) \leftarrow \text{fSub.KeyGen}()</math></li> <li>2. <math>i \leftarrow 1</math></li> <li>3. <math>\tau_0 \leftarrow \perp</math></li> <li>4. <math>b \leftarrow \{0, 1\}</math></li> <li>5. <math>b' \leftarrow \mathcal{U}^{\mathcal{O}_{\text{Alg}_\lambda}, \text{Reset}}(ek)</math></li> <li>6. <b>return</b> <math>b = b'</math></li> </ol> $\text{Reset}(j), 0 \leq j < i$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>2.     <math>\tau_i \leftarrow \tau_j</math></li> <li>3.     <math>i \leftarrow i + 1</math></li> </ol> $\mathcal{O}_{\text{Alg}_\lambda}(x, \mu)$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b = 0</math> <b>then</b></li> <li>2.     <math>y \leftarrow \Lambda.\text{Alg}_\lambda(x)</math></li> <li>3. <b>if</b> <math>b = 1</math> <b>then</b></li> <li>4.     <math>\mu \leftarrow \{0, 1\}</math></li> <li>5.     <math>(y, \tau_i) \leftarrow \text{fSub.Alg}_\lambda(x, ek, \tau_{i-1}, \mu)</math></li> <li>6.     <math>i \leftarrow i + 1</math></li> <li>7. <b>return</b> <math>y</math></li> </ol>
--	---

Figure 6.1: 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  $\mu$  on each invocation of  $\mathcal{O}_{\text{Alg}_\lambda}$  instead of using the supplied value.

games in Figure 3.1 to include  $\mu$ . The adversary  $\mathcal{U}$  supplies the value for  $\mu$  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 6.1, which we will use in Theorem 5. This game generates a random bit  $\mu$  instead of using the value given to the oracle.

For our general treatment in this chapter, we will work with abstract security games. Let  $\Lambda$  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  $\Lambda$ , and specifies a value  $\phi \in [0, 1]$ . The advantage of  $\mathcal{A}$  at the game SEC is defined as

$$\text{Adv}_\Lambda^{\text{SEC}}(\mathcal{A}) = |\Pr[\text{SEC}(\mathcal{A})] - \phi|.$$

This generalization encompasses the security notion of IND-CPA for symmetric encryption

used in [Chapter 5](#), as well as security notions such as unforgeability on signature schemes and MAC tags, and others.

As in [Chapter 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  $\Lambda$ . Let  $\text{fSub}$  be an ASA on  $\Lambda$ . We will assume without loss of generality that all invocations of the algorithm  $\Lambda.\text{Alg}_\lambda$  occur within oracles provided to the adversary  $\mathcal{A}$ . Define  $\text{SEC}'$  on  $\text{fSub}$  to be the same as SEC but with the following modifications:

- A state  $\tau$  (initialized to  $\perp$ ) and embedded key  $ek$  (generated with  $\text{fSub.KeyGen}$ ) 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  $\Lambda.\text{Alg}_\lambda$ ,  $\mathcal{O}$  is modified to take  $\mu$  as an additional parameter. Call these oracles  $\mu$ -oracles.
- Every instance of  $\Lambda.\text{Alg}_\lambda$  is replaced by  $\text{fSub.Alg}_\lambda$ , with the state variable  $\tau$  being assigned on output, where the input to  $\Lambda.\text{Alg}_\lambda$  is given as the first argument to  $\text{fSub.Alg}_\lambda$ , and the required  $ek$ ,  $\mu$ , and  $\tau$  are provided as inputs.

Essentially, we are modifying every instance of the line

$$y \leftarrow \Lambda.\text{Alg}_\lambda(x)$$

to be of the form

$$(y, \tau) \leftarrow \text{fSub.Alg}_\lambda(x, ek, \tau, \mu) .$$

This  $\text{SEC}'$  game will be used in [Theorem 6](#).

We will also define another game as a further modification to  $\text{SEC}'$ . Let  $S$  be the tuple of all variables assigned during the game  $\text{SEC}'$ , before the first instance where a  $\mu$ -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  $\text{SEC}'$ . We define the game  $\text{SEC}_{\text{fSub}}^g(\mathcal{A})$  is the same as  $\text{SEC}'_{\text{fSub}}(\mathcal{A})$  but where the  $\mu$ -oracles no longer accept  $\mu$  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  $\mu$ -oracles.

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

$\Gamma_1(\text{fSub}).\text{Alg}_\lambda(x, ek, \tau, \mu)$	$\Gamma_2(\text{fSub}).\text{Alg}_\lambda(x, ek, \tau, \mu)$
<ol style="list-style-type: none"> <li>1. <math>(ek', \bar{k}) \leftarrow ek</math></li> <li>2. <b>if</b> <math>\tau = \perp</math> <b>then</b></li> <li>3.   <math>\mu' \leftarrow \\$ \text{PKE.Enc}(ek', \mu)</math></li> <li>4.   <math>\sigma \leftarrow 0</math></li> <li>5.   <math>\tau' \leftarrow \perp</math></li> <li>6. <b>else</b></li> <li>7.   <math>((\sigma, \mu), \tau') \leftarrow \tau</math></li> <li>8.   <b>if</b> <math>\sigma = \text{PKE.clen}</math> <b>then</b></li> <li>9.     <math>\sigma \leftarrow 1</math></li> <li>10.   <math>\mu' \leftarrow \\$ \text{PKE.Enc}(ek', \mu)</math></li> <li>11. <b>else</b></li> <li>12.   <math>\sigma \leftarrow \sigma + 1</math></li> <li>13. <math>(y, \tau') \leftarrow \\$ \text{fSub.Alg}_\lambda(x, \bar{k}, \tau', \mu'[\sigma])</math></li> <li>14. <b>return</b> <math>(y, ((\sigma, \mu'), \tau'))</math></li> </ol>	<ol style="list-style-type: none"> <li>1. <math>(ek', \bar{k}) \leftarrow ek</math></li> <li>2. <b>if</b> <math>\tau = \perp</math> <b>then</b></li> <li>3.   <math>\mu' \leftarrow \\$ \text{PKE.Enc}(ek', \mu)</math></li> <li>4.   <math>\tau' \leftarrow \perp</math></li> <li>5. <b>else</b></li> <li>6.   <math>(\mu', \tau') \leftarrow \tau</math></li> <li>7. <math>(y, \tau') \leftarrow \\$ \text{fSub.Alg}_\lambda(x, \bar{k}, \tau', \mu')</math></li> <li>8. <b>return</b> <math>(y, (\mu', \tau'))</math></li> </ol>

Figure 6.2: Definitions of transformation  $\Gamma_1$  and  $\Gamma_2$  on flexible symmetric ASAs to obtain flexible asymmetric ASAs.

## 6.1 Making symmetric flexible ASAs asymmetric

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  $\Lambda$  be a cryptographic scheme. Let  $\text{fSub}$  be a flexible symmetric ASA on  $\Lambda$ , subverting the function  $\Lambda.\text{Alg}_\lambda$ . Let  $\text{PKE}$  be a public-key encryption scheme. Define  $\Gamma_1(\text{fSub})$  and  $\Gamma_2(\text{fSub})$  as the flexible asymmetric ASAs on  $\Lambda$  with  $\Gamma_1(\text{fSub}).\text{Alg}_\lambda$  and  $\Gamma_2(\text{fSub}).\text{Alg}_\lambda$  defined in Figure 6.2. The subversion-key generation algorithms for both subversions are the same: the extraction key is the private key generated by  $\text{PKE.KeyGen}$  and the embedded key is the concatenation of the public key generated by  $\text{PKE.KeyGen}$  and the key generated by  $\text{fSub.KeyGen}$ .

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

**Recovery of  $\mu$ .** We will be treating the analysis of recovery of  $\mu$  for  $\Gamma_1$  and  $\Gamma_2$  relatively informally. If  $\text{fSub}$  is a flexible symmetric ASA, then a subverter  $\mathcal{A}$  in possession of the key  $\bar{k}$ , upon observing outputs of the function  $\text{fSub.Alg}_\lambda(\cdot, \bar{k}, \tau, \mu)$ , will be able to recover  $\mu$ . Then  $\Gamma_2(\text{fSub}).\text{Alg}_\lambda$  enables the same for an adversary in possession of the extraction key  $xk$ : the value that  $\text{fSub}$  is asked to leak will always be a chosen encryption  $\mu'$  of  $\mu$  under  $\text{PKE}$ , and since  $\text{fSub}$  enables recovery of  $\mu'$  in this context,  $\mathcal{A}$  can recover  $\mu$  using  $xk$ .

The case of  $\Gamma_1$  is a little bit more complicated. In this case,  $\mu$  is encrypted to  $\mu'$ , 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  $\mu'$ , 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  $\mu'$  would not be able to be recovered.

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

**Theorem 5.** *Let  $\Lambda$  be a cryptographic scheme. Let **fSub** be a flexible symmetric ASA on  $\Lambda$ . Let  $\mathcal{U}_1$  be an adversary for the ASRDET game of [Figure 6.1](#) on  $\Gamma_1(\text{fSub})$ . Then there is an adversary  $\mathcal{D}_1$  such that*

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

*The running time of  $\mathcal{D}_1$  is about that of  $\mathcal{U}_1$  plus the running time of **PKE.KeyGen** and **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  $\text{ASRDET}_{\Gamma_1(\text{fSub})}(\mathcal{U}_1)$ . Let  $L_1$  be the same as  $L_0$  but with  $\mu'$  sampled randomly instead of computed as an encryption of  $\mu$ . Let  $L_2$  be the augmented state reset detectability game  $\text{ASRDET\$}_{\text{fSub}}$ .

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  $\mathcal{O}_{\text{PKE.Enc}}$  oracle in place of **PKE.Enc**, given in [Figure 6.3](#).

Let  $b_{\$}$  denote the bit from the  $\text{IND\$}_{\text{PKE}}(\mathcal{D}_1)$  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

$$|\Pr[L_1] - \Pr[L_0]| = 2\text{Adv}_{\text{PKE}}^{\text{IND\$}}(\mathcal{D}_1) .$$

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  $\Gamma_1(\text{fSub}).\text{Alg}_{\lambda}$  in game  $L_1$  act as book-keeping in order to use exactly one new random bit  $\mu$  at a time in **fSub.Alg $_{\lambda}$** . Substituting

$\mathcal{D}_1^{\mathcal{O}_{\text{PKE.Enc}}}(pk)$ <hr/> <ol style="list-style-type: none"> <li>1. <math>(xk, (ek', \bar{k})) \leftarrow \Gamma_1(\text{fSub}).\text{KeyGen}()</math></li> <li>2. <math>ek \leftarrow (pk, \bar{k})</math></li> <li>3. <math>i \leftarrow 1</math></li> <li>4. <math>\tau_0 \leftarrow \perp</math></li> <li>5. <math>b_{\text{det}} \leftarrow \{0, 1\}</math></li> <li>6. <math>b'_{\text{det}} \leftarrow \mathcal{U}_1^{\mathcal{O}_{\text{Alg}_\lambda}, \text{Reset}}(ek)</math></li> <li>7. <b>if</b> <math>b_{\text{det}} = b'_{\text{det}}</math></li> <li>8.     <b>return</b> 1</li> <li>9. <b>else</b></li> <li>10.    <b>return</b> 0</li> </ol> $\text{Reset}(j), 0 \leq j < i$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b_{\text{det}} = 1</math> <b>then</b></li> <li>2.     <math>\tau_i \leftarrow \tau_j</math></li> <li>3.     <math>i \leftarrow i + 1</math></li> </ol>	$\mathcal{O}_{\text{Alg}_\lambda}(x, \mu)$ <hr/> <ol style="list-style-type: none"> <li>1. <b>if</b> <math>b_{\text{det}} = 0</math> <b>then</b></li> <li>2.     <math>y \leftarrow \Lambda.\text{Alg}_\lambda(x)</math></li> <li>3. <b>if</b> <math>b_{\text{det}} = 1</math> <b>then</b></li> <li>4.     <math>(y, \tau_i) \leftarrow \text{Helper}(x, ek, \tau_{i-1}, \mu)</math></li> <li>5.     <math>i \leftarrow i + 1</math></li> <li>6. <b>return</b> <math>y</math></li> </ol> $\text{Helper}(x, ek, \tau, \mu)$ <hr/> <ol style="list-style-type: none"> <li>1. <math>(pk, \bar{k}) \leftarrow ek</math></li> <li>2. <b>if</b> <math>\tau = \perp</math> <b>then</b></li> <li>3.     <math>\mu' \leftarrow \mathcal{O}_{\text{PKE.Enc}}(\mu)</math></li> <li>4.     <math>\sigma \leftarrow 0</math></li> <li>5.     <math>\tau' \leftarrow \perp</math></li> <li>6. <b>else</b> <math>((\sigma, \mu), \tau') \leftarrow \tau</math></li> <li>7. <b>if</b> <math>\sigma = \text{PKE.clen}</math> <b>then</b></li> <li>8.     <math>\sigma \leftarrow 1</math></li> <li>9.     <math>\mu' \leftarrow \mathcal{O}_{\text{PKE.Enc}}(\mu)</math></li> <li>10. <b>else</b> <math>\sigma \leftarrow \sigma + 1</math></li> <li>11. <math>(y, \tau') \leftarrow \text{fSub}.\text{Alg}_\lambda(x, \bar{k}, \tau', \mu')</math></li> <li>12. <b>return</b> <math>(y, ((\sigma, \mu'), \tau'))</math></li> </ol>
---	---

Figure 6.3: Adversary  $\mathcal{D}_1$  for the proof of [Theorem 5](#).

these lines for  $\mu' \leftarrow_{\$} \{0, 1\}$  and moving this line to the first line of the oracle call gives us precisely the game  $L_2 = \text{ASRDET}_{\text{fSub}}^{\$}(\mathcal{U}_1)$ , and  $\Pr[L_1] = \Pr[L_2]$ .

Taken together we get

$$\begin{aligned}
\text{Adv}_{\Gamma_1(\text{fSub})}^{\text{ASRDET}}(\mathcal{U}_1) &= |\Pr[L_0] - \frac{1}{2}| \\
&= |\Pr[L_0] - \Pr[L_1] + \Pr[L_1] - \frac{1}{2}| \\
&\leq |\Pr[L_0] - \Pr[L_1]| + |\Pr[L_1] - \frac{1}{2}| \\
&= \text{Adv}_{\text{fSub}}^{\text{ASRDET}^{\$}}(\mathcal{U}_1) + 2\text{Adv}_{\text{PKE}}^{\text{IND}^{\$}}(\mathcal{D}_1),
\end{aligned}$$

as desired.  $\square$

**Proving  $\Gamma_2(\text{fSub})$  is a type 2 asymmetric ASA.** Next we deal with both the regular undetectability and the security of  $\Gamma_2(\text{fSub})$ , proving that  $\Gamma_2(\text{fSub})$  is a type 2 asymmetric ASA. The security of  $\Gamma_2(\text{fSub})$  is evaluated using the game  $\text{SEC}_{\Gamma_2(\text{fSub})}^g$ , and an adversary's advantage at this game is bounded by using the game  $\text{SEC}'_{\text{fSub}}$ . In this sense, we are relating the security of  $\Gamma_2(\text{fSub})$  to the security of  $\text{fSub}$  in the situation where the adversary must provide the value of  $\mu$ , the “value to be leaked” by  $\text{fSub}$ . 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  $\text{fSub}$  is not leaking any additional information; this is codified in the game  $\text{SEC}'_{\text{fSub}}$ . This provides the intuition for our security result, and the formalization of what is required from  $\text{fSub}$  to ensure that  $\Gamma_2(\text{fSub})$  is secure.

**Theorem 6.** *Let  $\Lambda$  be a cryptographic scheme. Let  $\text{fSub}$  be a flexible symmetric ASA on  $\Lambda$ . Let  $\text{SEC}$  be a security game on  $\Lambda$ . Let  $g$  be a selection function for the game  $\text{SEC}'$ . Let  $\mathcal{U}_1$  be an adversary for the  $\text{SRDET}$  game on  $\Gamma_2(\text{fSub})$  and  $\mathcal{V}_1$  be an adversary for the  $\text{SEC}^g$  game on  $\Gamma_2(\text{fSub})$ . Then there is an adversary  $\mathcal{U}_2$  such that*

$$\text{Adv}_{\Gamma_2(\text{fSub})}^{\text{SRDET}}(\mathcal{U}_1) = \text{Adv}_{\text{fSub}}^{\text{SRDET}}(\mathcal{U}_2) ,$$

and adversaries  $\mathcal{D}_2$  and  $\mathcal{V}_2$  such that

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

*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  $\text{PKE.KeyGen}$ , plus the running time of  $\text{PKE.Enc}$  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 6.4. We claim that  $\mathcal{U}_2$  exactly simulates the game  $\text{SRDET}_{\Gamma_2(\text{fSub})}$  for  $\mathcal{U}_1$ .

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

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

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  $\mathcal{O}_{\text{PKE}. \text{Enc}}$  oracle in place of  $\text{PKE}. \text{Enc}$ . This is shown in Figure 6.4.

Let  $b_{\text{PKE}}$  denote the bit from the  $\text{IND-CPA}_{\text{PKE}}(\mathcal{D}_2)$  game. Note that if  $b_{\text{PKE}} = 1$ , then the oracle simulated by  $\mathcal{D}_2$  proceeds exactly as in game  $K_1$ , since in  $K_1$  the only value of  $\mu$  passed to the **Helper** function is 0. If  $b_{\text{PKE}} = 0$ , then  $\mathcal{D}_2$  proceeds exactly as in game  $K_0$ . Thus we have

$$\begin{aligned} \Pr[\mathcal{D}_2 \Rightarrow 1 \mid b_{\text{PKE}} = 1] &= \Pr[K_1] \quad , \\ \Pr[\mathcal{D}_2 \Rightarrow 1 \mid b_{\text{PKE}} = 0] &= \Pr[K_0] \quad , \end{aligned}$$

and hence

$$|\Pr[K_1] - \Pr[K_0]| = 2\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(\mathcal{D}_2) \quad .$$

Next, we will construct the adversary  $\mathcal{V}_2$  in the game  $\text{SEC}'_{\text{fSub}}$  directly. Suppose that the game SEC provides oracles  $\mathcal{O}_j$  for  $1 \leq j \leq t$ , each with inputs labeled  $x$  and outputs labeled  $y$  as tuples. After being modified for game  $\text{SEC}'$ , assume that oracles  $\mathcal{O}_1, \dots, \mathcal{O}_{t_1}$  for some  $1 \leq t_1 \leq t$  are  $\mu$ -oracles, and the rest are not. The adversary  $\mathcal{V}_2$  is given in Figure 6.4.

We claim that  $\mathcal{V}_2$  exactly simulates the game  $\text{SEC}_{\Gamma_2(\text{fSub})}^0$  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  $\mu$  that  $\mathcal{V}_1$  makes, the value of  $\mu'$  passed to  $\mathcal{V}_2$ 's oracle  $\mathcal{O}'_j$  is an encryption of  $\mathbf{0}$  under PKE. Furthermore, the same  $\mu'$  is reused in all



$\mathcal{U}_2^{\mathcal{O}_{\text{fSub}. \text{Alg}_\lambda}, \text{Reset}_2}$ <hr/> 1. $(xk, ek') \leftarrow \$ \text{PKE.KeyGen}()$ 2. $i \leftarrow 1$ 3. $\tau_0 \leftarrow \perp$ 4. $b \leftarrow \$ \mathcal{U}_1^{\mathcal{O}_{\Gamma_2(\text{fSub}). \text{Alg}_\lambda}, \text{Reset}_1}$ 5. <b>return</b> $b$ <hr/> <b>Reset<sub>1</sub>(j), 0 ≤ j &lt; i</b> <hr/> 1. $\tau_i \leftarrow \tau_j$ 2. $i \leftarrow i + 1$ 3. <b>Reset<sub>2</sub>(j)</b> <hr/> $\mathcal{O}_{\Gamma_2(\text{fSub}). \text{Alg}_\lambda}(x, \mu)$ <hr/> 1. <b>if</b> $\tau_{i-1} = \perp$ <b>then</b> 2. $\mu' \leftarrow \$ \text{PKE.Enc}(ek', \mu)$ 3. <b>else</b> 4. $\mu' \leftarrow \tau_{i-1}$ 5. $\tau_i \leftarrow \mu'$ 6. $y \leftarrow \$ \mathcal{O}_{\text{fSub}. \text{Alg}_\lambda}(x, \mu')$ 7. $i \leftarrow i + 1$ 8. <b>return</b> $y$	$\mathcal{D}_2^{\mathcal{O}_{\text{PKE}. \text{Enc}}}(pk)$ <hr/> 1. $\text{SEC}^g$ with the first part of 2. $ek = (ek', \bar{k})$ , which is generated 3. by $\Gamma_2(\text{fSub}).\text{KeyGen}$ , replaced by 4. $pk$ , and $\Gamma_2(\text{fSub}).\text{Alg}_\lambda$ 5. replaced by <b>Helper</b> . <hr/> <b>Helper(x, ek, τ, μ)</b> <hr/> 1. $(pk, \bar{k}) \leftarrow ek$ 2. <b>if</b> $\tau = \perp$ <b>then</b> 3. $\mu' \leftarrow \$ \mathcal{O}_{\text{PKE}. \text{Enc}}(\mu)$ 4. $\tau' \leftarrow \perp$ 5. <b>else</b> 6. $(\mu', \tau') \leftarrow \tau$ 7. $(y, \tau') \leftarrow \$ \text{fSub}. \text{Alg}_\lambda(x, \bar{k}, \tau', \mu')$ 8. <b>return</b> $(y, (\mu', \tau'))$	$\mathcal{V}_2(\bar{k})$ <hr/> 1. $(xk, ek') \leftarrow \$ \text{PKE.KeyGen}()$ 2. $ek \leftarrow (ek', \bar{k})$ 3. $\tau \leftarrow \perp$ 4. $\mathcal{V}_1(ek)$ <hr/> $\mathcal{V}_2^{\mathcal{O}'_j, j \in J \subseteq \{1, \dots, t\}}(x)$ <hr/> 1. $y \leftarrow \$ \mathcal{V}_1^{\mathcal{O}'_j, j \in J \subseteq \{1, \dots, t\}}(x)$ 2. <b>return</b> $y$ <hr/> $\mathcal{O}_j(x), 1 \leq j \leq t_1$ <hr/> 1. <b>if</b> $\tau = \perp$ <b>then</b> 2. $\mu' \leftarrow \$ \text{PKE.Enc}(ek', 0)$ 3. <b>else</b> $\mu' \leftarrow \tau$ 4. $\tau \leftarrow \mu'$ 5. $y \leftarrow \$ \mathcal{O}'_j(x, \mu')$ 6. <b>return</b> $y$ <hr/> $\mathcal{O}_j(x), t_1 \leq j \leq t$ <hr/> 1. $y \leftarrow \$ \mathcal{O}'_j(x)$ 2. <b>return</b> $y$
--	---	--

Figure 6.4: 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  $\text{SEC}^g$  game, and any oracle query that would use  $\Gamma_2(\text{fSub}).\text{Alg}_\lambda(x, ek, \tau, \mu)$  is answered using the code in **Helper**.

future oracle invocations. All oracles not accepting a parameter  $\mu$  are provided directly to  $\mathcal{V}_1$ . This exactly simulates the  $\text{SEC}_{\Gamma_2(\text{fSub})}^0$  game. Hence  $\Pr[K_1] = \Pr[\text{SEC}_{\Gamma_2(\text{fSub})}^0(\mathcal{V}_1)] = \Pr[\text{SEC}'_{\text{fSub}}(\mathcal{V}_2)] = \Pr[K_2]$ .

Combining these results, we get

$$\begin{aligned}
\text{Adv}_{\Gamma_2(\text{fSub})}^{\text{SEC}^g}(\mathcal{V}_1) &= |\Pr[K_0] - \phi| \\
&= |\Pr[K_0] - \Pr[K_1] + \Pr[K_1] - \phi| \\
&\leq |\Pr[K_0] - \Pr[K_1]| + |\Pr[K_2] - \phi| \\
&= \text{Adv}_{\text{fSub}}^{\text{SEC}'}(\mathcal{V}_2) + 2\text{Adv}_{\text{PKE}}^{\text{IND-CPA}}(\mathcal{D}_2),
\end{aligned}$$

as desired. □

For the symmetric subversions on symmetric encryption from [BPR14] and [BJK15], we can show that the quantities  $\text{Adv}_{\text{fSub}}^{\text{SEC}'}$  and  $\text{Adv}_{\text{fSub}}^{\text{ASRDET}^{\$}}$  are small for any adversary; the proofs in Chapter 4 and Chapter 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  $\Gamma_1(\text{fSub})$  and  $\Gamma_2(\text{fSub})$  using the theorems in this chapter.

# Chapter 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 work 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 [Chapter 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 thesis.

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<sup>+</sup>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<sup>+</sup>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 De-laune, editors, *CSF 2018 Computer 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. Imprimitve 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.

- [SB88] M.E. Smid and D.K. Branstad. Data encryption standard: past and future. *Proceedings of the IEEE*, 76(5):550–559, 1988.
- [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.