ANALYSIS OF PUBLIC-KEY ENCRYPTION SCHEMES IN EXTENDED ATTACK MODELS

Dale Luke Sibborn

Royal Holloway and Bedford New College,
University of London

Thesis submitted to
The University of London
for the degree of
Doctor of Philosophy
2015.

Declaration

These doctoral studies were conducted under the supervision of Professor Kenneth G. Paterson.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Dale Luke Sibborn

June 2015

Abstract

Cryptographic models are intended to represent an adversary's capabilities when attacking encryption schemes. Models often err on the side of caution by over-estimating the power of adversaries. However, several recent attacks reported in the literature demonstrate that measuring an adversary's potential is a difficult task. This thesis will view the cryptographic landscape from the perspective of an adversary and the implementer.

The first part of this thesis will consider the view of an adversary. We study how an adversary can obtain leaked information about a private key. The particular scenario we study is the *cold boot attack* whereby an adversary can procure a noisy version of the key (i.e. the noisy copy will contain errors). Such an attack is not traditionally modelled by the standard security games. We show how the adversary can recover the original secret key, and hence compromise security, in the RSA and discrete logarithm settings. In the discrete logarithm setting our approach is general, but we mount attacks against specific elliptic curve implementations of OpenSSL and PolarSSL.

In the second part of this thesis we introduce a new type of attack, which we call the *Related Randomness Attack*. We define a security game to model these new attacks in a variety of scenarios, such as encryption schemes having access to non-uniform randomness sources, or perhaps the randomness source is under the control of an adversary. We introduce several variants of this model, and we provide generic transforms that convert traditional indistinguishability-style secure schemes into schemes that are secure with respect to our new, extended definition.

Contents

List of Tables

List of Figures

Notation

$\mathbb{N}$	The set of natural numbers
$\mathbb{Z}$	The set of integers
$\mathbb{R}$	The set of real numbers
$\lambda \in \mathbb{N}$	The security parameter
$\mathbb{Z}_n$	The set of least residues modulo $n \in \mathbb{N}$
$\{0,1\}^n$	The set of binary strings of length $n \in \mathbb{N}$
$x \oplus y$	The exclusive-or (XOR) of strings x and y
$\lvert X \rvert$	The cardinality of the set X
$\lvert r \rvert$	The absolute value of a real number r
$[n]$	The set of natural numbers $1, \ldots, n$
$\lfloor x \rfloor$	The largest integer not greater than x
$\lceil x \rceil$	The smallest integer not less than x
$s \leftarrow_{\$} S$	Selecting an element s uniformly at random from the set S
$y \leftarrow \mathcal{A}(x)$	Running an algorithm on input x and assigning the result to y
$x \leftarrow y$	Assigning the value y to the variable x
$\perp$	The error or rejection symbol
$\wedge$	Logical conjunction

$\vee$	Logical disjunction
$\mathbb{P}[A]$	The probability of event A occurring
$\log x$	The base-2 logarithm of x
$\ln x$	The natural logarithm of x
$\mathbb{G}$	Denotes a cyclic group
g	Denotes a generator of a cyclic group $\mathbb{G}$
$a \bmod b$	The remainder of a when divided by b
$\mathcal{D}(f)$	Denotes the domain of the function f
$\mathcal{R}(f)$	Denotes the range of the function f

Publications

The work in this thesis originates from the four papers listed below.

1. Kenneth G. Paterson, Antigoni Polychroniadou, and Dale L. Sibborn, *A Coding Theoretic Approach to Recovering Noisy RSA Keys*, Advances in Cryptology - Asiacrypt 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings, Lecture Notes in Computer Science, vol. 7658, Springer, 2012.

2. Kenneth G. Paterson, Jacob C. N. Schuldt, and Dale L. Sibborn, *Related Randomness Attacks for Public Key Encryption*, Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings (Hugo Krawczyk, ed.), Lecture Notes in Computer Science, vol. 8383, Springer, 2014, pp. 465-482.

3. Kenneth G. Paterson, Jacob C. N. Schuldt, Dale L. Sibborn and Hoeteck Wee, *Security Against Related Randomness Attacks via Reconstructive Extractors*, IACR Cryptology ePrint Archive Report 2015/892.

4. Bertram Poettering and Dale L. Sibborn *Cold Boot Attacks in the Discrete Logarithm Setting*, Topics in Cryptology - CT-RSA 2015, The Cryptographers Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings (Kaisa Nyberg, ed.), Lecture Notes in Computer Science, vol. 9048, Springer, 2015, pp. 449-465.

Acknowledgements

First and foremost, I would like to thank my supervisor, Kenny Paterson, for tolerating me for all these years. His support and guidance were invaluable, even though I did not utilise them as much as I should have.

I am grateful for the financial support of the EPSRC who funded the project of which I was a part, and I owe a thank you to the other members of the team: Susan and my two favourite post-docs, Bertram and Jacob.

I would also like to thank my examiners, Keith Martin and Elisabeth Oswald, for making the viva such a pleasant and enjoyable experience.

Thank you to all my friends within the department: Caroline, Christian, Dan, Dean, Eugenio, George, Gordon, James, Matteo, Pavlo, Pip, Rachel, Sam, Shahram and Wanpeng. Thank you all for the trips to the Happy Man, Crosslands and Medicine. In addition, a special thanks must go to James Alderman for his assistance with my brief foray into computer programming.

From a non-academic perspective, I would like to thank my incredible friends: Claire, Eamonn, Laura and Tim. Thank you also to my amazing housemates, the 'Vegal'antes of 59 Vegal Cresent: Ana, Andy, Hannah, Kaja and Mersiye. Thank you all for invariably putting a smile on my face. It would be remiss of me not to mention SCMKC at this point since that is where the journey really began, so thank you to all its members too.

Finally, I owe a huge debt to my family for supporting me throughout this journey, so thank you to my dad Dean, my mum Melanie, my sister Kaidee, and my grandparents Gwendoline, Leonard, Maureen and Trevor.

Chapter 1

Introduction

In this chapter we provide motivation for this thesis and we provide a roadmap of the results we shall present.

1.1 Motivation

In recent decades the study of cryptography has transformed from an art into a science, and in this time there has been a gradual divergence between the theoretical and practical cryptographic literature. A major difference occurs when attempting to evaluate the strength of an adversary. When designing cryptographic schemes we begin by modelling the capabilities of an adversary. We then attempt to show that our scheme is secure against any adversary with the modelled powers (typically by providing a reduction to a suitably-hard problem). However, an inherent problem with these models is that we do not always know the lengths an adversary will go to in order to subvert our cryptographic methods. Various schemes have been shown to be secure according to certain models, but these schemes may be trivially broken by attacks that fall outside the scope of the models. Such failures frequently occur in the real world [36, 42, 44, 2]. Similarly, the assumptions used within the

model may not always be realisable in practice (see, for example, [31, 68, 50]). Randomised primitives are almost ubiquitously proved to be secure *only* when the scheme uses fresh, uniform randomness for every (randomised) cryptographic operation. Unfortunately, obtaining such high-quality randomness is a difficult task in practice, and schemes typically forfeit all security guarantees when poor randomness is used.

In this thesis we will consider both of the above-mentioned disparities. In the first part we will study how to reconstruct private keys when an adversary is able to obtain a leakage function of the private key. Standard security models do not allow an adversary this power, but in practice it is well-known that such information can be procured, albeit with some effort on the attacker's part. We study three particular encryption schemes and show that, whilst secure according to traditional notions, these schemes may be broken when an adversary obtains sufficient partial information about the private key. In the second part of this thesis we will introduce novel security models for encryption that do *not* require randomised primitives to use fresh, uniform randomness for every operation. These models are designed to reflect the real world more accurately and hence are more applicable to practice. We provide several results with respect to these models, both negative and positive, and we show how standard encryption schemes can be modified to remain secure in our new models. The two parts are discussed in more detail below.

Part I. The first part of this thesis will study how to reconstruct a private key when given a noisy version of it. It is possible to obtain a noisy version of the key via a *cold boot attack* (whereby the contents of the memory can be extracted, but with errors). Traditional security notions for encryption do not allow an adversary to obtain such leakage, and various cryptographic schemes become insecure when an adversary is allowed access to such information. We

will concern ourselves with the reconstruction of RSA and elliptic curve private keys. In the latter case, we focus in particular on the implementation of elliptic curve primitives in OpenSSL and PolarSSL. Despite being secure according to standard security notions, for each scheme we show that we can recover private keys (and thereby completely compromise security) if an adversary can acquire a noisy version of the private key that is not too heavily degraded. Our key recovery algorithms for OpenSSL and PolarSSL are new to the literature, but key reconstruction in the RSA setting has been studied several times previously in [37], [44] and [42] (and indeed several times following the publication of the work based on this chapter [52, 51]). Crucially, however, the degradation models used in these previous reports do not accurately reflect the reality of cold boot attacks, but our models do, and we show how to recover keys that have been degraded according to this true model. We take two approaches to recovering the private keys, the first being a Maximum-Likelihood (ML) approach, and the second being a threshold-based approach (reminiscent of that in [42]). Each approach has its own advantages and disadvantages, and we therefore use different approaches for different scenarios.

Part II. In the second half of this thesis we consider how to protect against a certain class of adversarial attacks that fall outside the scope of traditional security notions for public-key encryption. Standard security games typically assume that fresh, uniform randomness is used for every randomised encryption, signature, etc. However, in practice this is frequently not true. There are well-known instances of cryptographic schemes using randomness that is highly correlated, resulting in the schemes becoming trivial to break [26, 31, 34, 35, 22, 3, 25, 60, 68]. In order to protect against such attacks, we must first introduce new security models that simulate these types of randomness failure, since there are no such models currently in the literature. We call

our attack the *Related Randomness Attack*, and we concentrate on public-key encryption and key encapsulation mechanisms. Our security game that models this type of attack extends the IND-ATK notion (for ATK=CPA or CCA) of security, and gives the adversary extra capabilities (such as requesting encryptions under correlated randomness values). As a result, any scheme that is secure in our new model is also secure according to the IND-ATK notions. We propose several variants of our strongest game and we present transforms that will convert any IND-ATK secure scheme into a scheme that is secure with respect to the Related Randomness Attack notion. Additionally, we show that it is trivial to protect against bad randomness in the symmetric setting even if an adversary can force a particular randomness value to be used for challenge encryptions.

1.2 Thesis Structure

Chapter 2. This chapter contains the preliminaries and will establish the notational conventions that will be used throughout this thesis. Standard definitions and computational assumptions can be found in this section. Any advanced or non-standard definitions will be encountered in the relevant sections.

Chapter 3. In this chapter we will consider the reconstruction of RSA private keys that have been obtained via a cold boot attack. We will briefly recall some previous work in this area, since we build upon these previous techniques. We will then provide an algorithm that is more successful and also works in a setting that is more applicable to practice. Furthermore, we are able to recover private keys for greater noise levels than are possible in the current literature.

Chapter 4. This chapter will expand on the work of Chapter 3, but in a different direction. We will study the recovery of private keys in the discrete logarithm setting; in particular we attack specific implementations of OpenSSL and PolarSSL for elliptic curve cryptography. We provide a general framework for recovering keys in each of these scenarios, and we show that a well-known statistical test can be used within this framework to achieve an arbitrary success rate when recovering keys. Furthermore, we show that this statistical test may be used within the framework of the previous chapter, thereby allowing the recovery of RSA keys with arbitrary success rate.

Chapter 5. The previous chapters considered breaking schemes using attacks that fell outside the scope of security models. In this chapter we will change perspective and consider the design of stronger security models that are designed to address such issues. Specifically, we define a new security model, which will capture our new notion of Related Randomness Attack (RRA). We also define a variant of this new model, with each variant having different strengths and security guarantees, and being applicable to different scenarios. For example, in our weaker models we may restrict either the public keys or the adaptivity of the adversary. We are able to provide concrete instantiations for these models in both the random oracle model and the standard model.

Chapter 6. In this chapter we consider another variant of our Related Randomness Attack model. In this setting the adversaries are no longer adaptive, but we require security to hold for all adversaries from a particular class. We show that this security model has interesting connections with the auxiliary-input setting for public-key encryption, in which an adversary is given a leakage function of the private key. Furthermore, we exhibit a relation between this model and auxiliary-input reconstructive extractors, showing that these

extractors may be used to instantiate secure encryption schemes in this new model.

Chapter 7. In this section we will switch our attention from public-key encryption to Key Encapsulation Mechanisms (KEMs). We adapt the security models of Chapters 5 and 6 to the KEM setting, and we show how to construct an RRA secure PKE scheme by using an RRA secure KEM as a building block. Moreover, we develop connections with Chapter 6 by showing that the main theorem of this chapter is a generalisation of a theorem that appeared in Chapter 6. Specifically, we show that a theorem of Chapter 6 implicitly used an RRA secure KEM as a building block.

Chapter 2

Preliminaries

In this chapter we will take a brief tour of the fundamental concepts in provable security, and then we shall introduce the primitives that will be used extensively throughout this thesis. Any basic definitions will be given in this section. Any new or advanced notions will be included as and when needed.

2.1 Mathematical Background

2.1.1 Computational complexity

When dealing with asymptotic results it is convenient to work with big-$\mathcal{O}$ notation. This notation gives us a succinct, but approximate, measure of the running-time of algorithms.

Definition 2.1.1 (Big-$\mathcal{O}$ notation). Let f and g be functions defined on a subset of the real numbers. We say that $f(x) = \mathcal{O}(g(x))$ as $x \to \infty$ if and only if there exists a positive constant M and a real number x_0 such that $|f(x)| \leq M|g(x)|$ for all $x \geq x_0$.

Definition 2.1.2 (Polynomial-time). An algorithm is said to be polynomial-time if it always terminates in time $\mathcal{O}(k^c)$, where k is the size of the input and

c is a constant.

Any algorithm that runs in polynomial time is deemed to be efficient. If an adversary is not polynomial-time, then it is necessarily super-polynomial time. There are various sub-categories of super-polynomial time, but we have no need to discuss these.

Definition 2.1.3 (Negligible function). A function ψ is negligible if, for every constant c, there exists a real number N_c such that $|\psi(x)| < x^{-c}$ for all $x > N_c$.

2.1.2 Computational hardness assumptions

Definition 2.1.4 (Decisional Diffie-Hellman (DDH) problem). The advantage of an algorithm $\mathcal{A}$ in solving the Decisional Diffie-Hellman problem in a cyclic group $\mathbb{G}$ of order $p = p(\lambda)$ for the security parameter λ, is defined as

$$\mathbf{Adv}^{\mathrm{ddh}}_{\mathbb{G},\mathcal{A}}(\lambda) = |\mathbb{P}[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \mathbb{P}[\mathcal{A}(g, g^a, g^b, R) = 1]|$$

where g is a random generator of $\mathbb{G}$, $a, b \leftarrow_{\$} \mathbb{Z}_p$, $R \leftarrow_{\$} \mathbb{G}$, and the probability is taken over the random coins consumed by $\mathcal{A}$.

Assumption 2.1.1 (DDH assumption for $\mathbb{G}$). No polynomial-time algorithm has a non-negligible advantage in solving the DDH problem in $\mathbb{G}$.

Definition 2.1.5 (q-Decisional Diffie-Hellman Inversion (q-DDHI) problem). The advantage of an algorithm $\mathcal{A}$ in solving the q-Decisional Diffie-Hellman Inversion problem in a cyclic group $\mathbb{G}$ of order $p = p(\lambda)$, is defined as

$$\mathbf{Adv}^{q\text{-ddhi}}_{\mathbb{G},\mathcal{A}}(\lambda) = |\mathbb{P}[\mathcal{A}(g, g^x, \ldots, g^{x^q}, g^{1/x}) = 1] - \mathbb{P}[\mathcal{A}(g, g^x, \ldots, g^{x^q}, R) = 1]|$$

where g is a random generator of $\mathbb{G}$, $x \leftarrow_{\$} \mathbb{Z}_p$, $R \leftarrow_{\$} \mathbb{G}$, and the probability is taken over the choice of $x \in \mathbb{Z}_p$, and the random coins consumed by $\mathcal{A}$.

Assumption 2.1.2 (q-DDHI assumption for $\mathbb{G}$). No polynomial-time algorithm has a non-negligible advantage in solving the q-DDHI problem in $\mathbb{G}$.

2.1.3 Code-based games

All our security definitions and proofs will utilise code-based games and the associated terminology. Code-based games were introduced by Bellare and Rogaway in [11], and the definitions in this subsection are reproduced from the cited paper.

A game consists of at least two procedures, which are **Initialise** and **Finalise**. The games always begin with the **Initialise** procedure, which assigns starting values to all variables and then returns outputs, if there are any, to the adversary. If there are procedures other than **Initialise** and **Finalise**, the adversary $\mathcal{A}$ may then submit queries to these procedures (which are typically encryption or decryption oracles), and when $\mathcal{A}$ halts (and possibly outputs a value) the **Finalise** procedure begins. The **Finalise** procedure will take the output from $\mathcal{A}$ (if there is one) as its input and will output its own value. The value output by **Finalise** is defined to be the output of the game. We write $\mathbb{P}[G^{\mathcal{A}} \Rightarrow b]$ to denote the probability that game G outputs bit b when run with $\mathcal{A}$. Occasionally, we use the shorthand $\mathbb{P}[G^{\mathcal{A}}]$ to denote the probability that game G outputs bit 1 when run with $\mathcal{A}$. Alternatively, in some proofs we will use the notation $\mathcal{A}^G \Rightarrow b$. This means that the adversary outputs b when playing game G. We will occasionally use the notation $\mathcal{A}(x) \Rightarrow b$, which denotes adversary $\mathcal{A}$ outputting b when given the input x.

2.2 Cryptographic Primitives

In this section we will introduce some standard cryptographic schemes and primitives. Throughout this thesis we let ATK denote either CPA or CCA (Chosen Plaintext Attack or Chosen Ciphertext Attack) whenever theorems or statements apply to both attacks. Any proofs or figures will refer to the

CCA setting, but may be easily modified to the CPA case by removing access to the decryption or decapsulation oracle. We begin with public-key encryption.

2.2.1 Public-key encryption

We denote a specific PKE scheme $\mathtt{PKE}$ by a triple of algorithms, which are $(\mathtt{PKE.K}, \mathtt{PKE.E}, \mathtt{PKE.D})$. All three algorithms are polynomial-time. The randomised key generation algorithm $\mathtt{PKE.K}$ takes the security parameter λ as its input and outputs a key pair (pk, sk). The encryption algorithm, on input of a message $m \in \mathcal{M}$ and a public key pk, chooses random coins from $\mathtt{Rnd}$ and uses these coins to output a ciphertext c. The decryption algorithm is deterministic. Its inputs are a private key sk and a ciphertext c. The algorithm either outputs a message m or an error symbol $\perp$. We require the scheme $\mathtt{PKE}$ to satisfy the correctness property. That is, for all $\lambda \in \mathbb{N}$, all pairs (pk, sk) output by the key generation algorithm, and all messages $m \in \mathcal{M}_\lambda^{\mathtt{PKE}}$, we require that $\mathtt{PKE.D}(sk, \mathtt{PKE.E}(pk, m)) = m$.

proc. Initialise(λ):	**proc. Dec**(c):
$b \leftarrow_\$ \{0,1\}$;	if $c \in \mathcal{S}$
$(pk, sk) \leftarrow_\$ \mathtt{PKE.K}(1^\lambda)$;	return $\perp$
$\mathcal{S} \leftarrow \emptyset$;	else
return pk.	return $\mathtt{PKE.D}(sk, c)$.
proc. LR(m_0, m_1):	**proc. Finalise**(b'):
$c \leftarrow_\$ \mathtt{PKE.E}(pk, m_b)$	if $b = b'$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$	return 1
return c.	else
	return 0.

Figure 2.1: Game IND-CCA for PKE. For the CPA version of the game, there is no access to **proc. Dec**.

The definition of security for a PKE scheme is as follows:

Definition 2.2.1. The advantage of an *IND-ATK* adversary $\mathcal{A}$ against a scheme PKE is

$$\mathbf{Adv}^{\text{ind-atk}}_{\text{PKE},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\text{IND-ATK}^{\mathcal{A}}_{\text{PKE}}(\lambda) \Rightarrow 1] - 1,$$

for game IND-ATK of Figure 2.1. A scheme PKE is *IND-ATK* secure if the advantage of any polynomial-time adversary is negligible in the security parameter λ.

Note that in Figure 2.1, the security game keeps track of the ciphertexts that have been output by the **LR** oracle, and prevents the adversary requesting the decryption of these ciphertexts. This is because any adversary can trivially win the game if we do not insist on this restriction. The adversary could simply submit (m_0, m_1) to the **LR** oracle and then submit the returned ciphertext to the **Dec** oracle. This will obviously reveal which of m_0 or m_1 was encrypted. We will insist on such a restriction in our security games for all primitives.

Perhaps it is a good idea to consider an example of this security game in action. We will take an informal look at 'textbook' RSA, which is insecure according to the previous definition. The triple of algorithms is below.

PKE.K(λ):

- choose two uniformly random λ-bit primes, p and q, and set $N = pq$.
- choose e such that $\gcd(e, \phi(N)) = 1$, where ϕ is Euler's totient function.
- compute $d = e^{-1} \bmod \phi(N)$.
- return $pk = (N, e)$ and $sk = d$.

PKE.E(pk, m):

- compute $c = m^e \bmod N$.
- return c.

PKE.D(sk, c):

- compute $m = c^d \bmod N$.
- return m.

Note that for textbook RSA both the message space and ciphertext space are restricted to being $\mathbb{Z}_N^*$. It is straightforward to see that the textbook RSA scheme is insecure according to Definition 2.2.1. Consider an adversary that plays the game in Figure 2.1. Suppose the adversary submits (m, m') to the **LR** oracle, such that $m \neq m'$ and $m, m' \in \mathbb{Z}_N^*$. The adversary will receive a ciphertext c^*. Next, the adversary computes the ciphertext $c = \mathtt{PKE.E}(pk, m)$ for himself. Then, if $c = c^*$ the adversary outputs 0, and outputs 1 otherwise. It is trivial to see that this adversary wins both the CPA and CCA game with probability 1 (since no **Dec** queries were required). Hence, this textbook scheme is insecure, since this adversary is clearly polynomial-time and has a non-negligible advantage. Note that such an attack will work against any deterministic PKE scheme. Therefore, any PKE scheme that is secure according to Definition 2.2.1 must be randomised.

2.2.2 Data encapsulation mechanisms

In this subsection we will define Data Encapsulation Mechanisms (DEMs). DEMs are more commonly referred to as Symmetric-Key Encryption (SKE) schemes in the literature but, for reasons that will become apparent, we prefer the name DEM. We define a DEM scheme $\mathtt{DEM}$ in the natural way as a triple of algorithms $(\mathtt{DEM.K}, \mathtt{DEM.E}, \mathtt{DEM.D})$. The three algorithms are polynomial-time. Key generation takes λ as an input and outputs a key K. The encryption algorithm takes a message $m \in \mathcal{M}_\lambda^{\mathtt{DEM}}$ (the message space, which is dependent on the $\mathtt{DEM}$ and λ) and a key K as its inputs, chooses random coins from $\mathtt{Rnd}_\lambda^{\mathtt{DEM}}$ (the randomness space, which is dependent on the $\mathtt{DEM}$ and λ), and outputs a ciphertext c. Decryption takes a key K and a ciphertext c. The output is either a message m or an error symbol $\perp$. Again we require the correctness property, so that for all K output by the key generation algorithm and all m, we have $\mathtt{DEM.D}(K, \mathtt{DEM.E}(K, m)) = m$.

proc. Initialise(λ): $b \leftarrow_{\$} \{0,1\}$; $K \leftarrow_{\$} \texttt{DEM.K}(1^{\lambda})$; $\mathcal{S} \leftarrow \emptyset$. **proc. LR**($m_0, m_1$): $c \leftarrow_{\$} \texttt{DEM.E}(K, m_b)$ $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$ return c.	**proc. Dec**(c): if $c \in \mathcal{S}$ return $\perp$ else return $\texttt{DEM.D}(K, c)$. **proc. Finalise**(b'): if $b = b'$, return 1 else, return 0.

Figure 2.2: Game IND-CCA for a DEM DEM. The CPA version of the game removes access to **proc. Dec**.

Definition 2.2.2. The advantage of an *IND-ATK* adversary $\mathcal{A}$ against a scheme DEM is

$$\mathbf{Adv}^{\text{ind-atk}}_{\texttt{DEM},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\text{IND-ATK}^{\mathcal{A}}_{\texttt{DEM}}(\lambda) \Rightarrow 1] - 1,$$

where game IND-ATK is in Figure 2.2. A scheme DEM is *IND-ATK* secure if the advantage of any polynomial-time adversary is negligible in the security parameter λ.

2.2.3 Key encapsulation mechanisms

A KEM scheme KEM is defined by a triple $(\texttt{KEM.K}, \texttt{KEM.E}, \texttt{KEM.D})$ of polynomial-time algorithms. The randomised key generation algorithm takes the security parameter λ as its input, then outputs a key pair (ek, dk). The encryption algorithm takes an encryption key ek as its input, generates random coins from Rnd and outputs a pair $(c, k) \in \mathcal{C}^{\texttt{KEM}}_{\lambda} \times \mathcal{K}^{\texttt{KEM}}_{\lambda}$. The deterministic decryption algorithm takes a ciphertext c and a decryption key dk as its inputs and outputs a key k, or an error symbol $\perp$. For all pairs (ek, dk) we require that if $(c, k) \leftarrow \texttt{KEM.E}(ek)$, then $k \leftarrow \texttt{KEM.D}(dk, c)$.

Definition 2.2.3. The advantage of an *IND-ATK* adversary $\mathcal{A}$ against a key encapsulation mechanism KEM is

$$\mathbf{Adv}^{\text{ind-atk}}_{\texttt{KEM},\mathcal{A}}(\lambda) = 2 \cdot \mathbb{P}[\text{IND-ATK}^{\mathcal{A}}_{\texttt{KEM}}(\lambda) \Rightarrow 1] - 1,$$

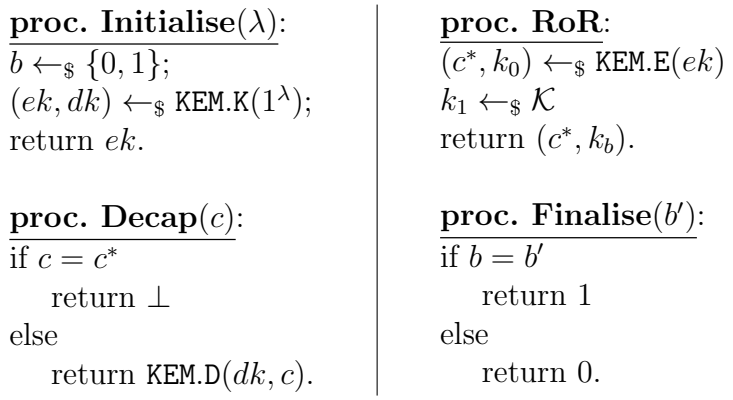

Figure 2.3: Game IND-CCA for a KEM. The CPA version removes access to **proc. Decap**.

where game IND-ATK is in Figure 2.3. We say KEM is a secure KEM if the advantage of any polynomial-time adversary is negligible in the security parameter λ.

2.2.4 KEM/DEM composition

In this section we will briefly mention the composition of KEMs and DEMs (we will study this in more detail in Chapter 7). The composition theorem was proved by Cramer and Shoup [21], and it states (informally) that a KEM and a DEM can be composed in such a way that one can obtain a secure PKE scheme. Cramer and Shoup showed that if the KEM outputs a pair (c, k), then the value k may be used as the key for a DEM. That is, if you wish to build a PKE scheme, and you want to encrypt message m with public key pk, then you can compute $(c, k) \leftarrow_{\$} \texttt{KEM.E}(pk)$, then compute $c^* \leftarrow_{\$} \texttt{DEM.E}(k, m)$. The ciphertext is then (c, c^*), which can be decrypted in the obvious way. Note that we just mentioned the use of a randomised DEM. The proof of Cramer and Shoup only requires a one-time secure DEM (i.e. the adversary can make only one **LR** query), hence the DEM need not be randomised. However, in our more general setting in Chapter 7, we will require a stronger DEM, and

therefore it is necessary for it to be randomised.

2.2.5 Pseudorandom functions

In this subsection we define Pseudorandom Functions, or PRFs. As the name suggests, a PRF is a family of functions that takes two inputs and then produces an output that should be indistinguishable from random. This notion is formalised below.

proc. Initialise(λ): $K \leftarrow_{\$} \mathtt{Keys}_\lambda$.	**proc. Initialise**(λ): FunTab $\leftarrow \emptyset$.
proc. Function(x): return $F(K, x)$.	**proc. Function**(x): if FunTab[x] $=\perp$, then FunTab[x] $\leftarrow_{\$} \mathtt{Rng}_\lambda$ return FunTab[x].
proc. Finalise(b): return b.	**proc. Finalise**(b): return b.

Figure 2.4: Games for PRF security. Game PRFReal is on the left, PRFRand on the right.

Definition 2.2.4. Let $F : \mathtt{Keys}_\lambda \times \mathtt{Dom}_\lambda \to \mathtt{Rng}_\lambda$ be a family of functions. The advantage of a PRF adversary $\mathcal{A}$ against F is

$$\mathbf{Adv}^{\mathrm{prf}}_{F,\mathcal{A}}(\lambda) := \mathbb{P}[\mathrm{PRFReal}^{\mathcal{A}}_{F}(\lambda) \Rightarrow 1] - \mathbb{P}[\mathrm{PRFRand}^{\mathcal{A}}_{\$}(\lambda) \Rightarrow 1]$$

where the games PRFReal and PRFRand are defined in Figure 2.4. We say F is a secure PRF family if the advantage of any polynomial time adversary is negligible in the security parameter λ.

In the previous security game, the adversary is required to output a bit b. We have not specified what this bit represents (since we do not need to), however intuitively we can the view this as an adversary outputting $b = 1$ if he believes the outputs are real, or $b = 0$ if he believes the outputs are random.

2.2.6 Related-key attack pseudorandom functions

Related-key attacks were introduced by Biham [14] after he noticed that the key scheduling algorithms of certain blockciphers would inherit exploitable relationships between keys. Related-key attacks have since been extended to various types of primitive, such as Identity-Based Encryption (IBE) and signatures (see, for example, [8]). In this thesis, we will only concern ourselves with related-key attacks for PRFs, which were formalised by Bellare et al. [7]. Related-Key Attack Pseudorandom Functions (RKA-PRFs) are PRFs that are secure against a much stronger class of attacks than the ones in the previous subsection. That is, a PRF must satisfy a stronger definition of security on order to be an RKA-PRF. In the previous PRF definition, the adversary was only allowed to see outputs of the PRF for a fixed key K. The RKA-PRF definition is an extension of this game that allows an adversary to submit functions ϕ to the oracle (along with an input x), and the function will be evaluated with the key $\phi(K)$. The security games and definition follow. Note that $\mathrm{FF}(X, Y, Z)$ denotes the set of all families of functions $F : X \times Y \to Z$.

proc. Initialise(λ): $K \leftarrow_{\$} \mathtt{Keys}_\lambda$. **proc. Function**$(\phi, x)$: return $F(\phi(K), x)$. **proc. Finalise**(b): return b.	**proc. Initialise**(λ): $G \leftarrow \mathrm{FF}(\mathtt{Keys}_\lambda, \mathtt{Dom}_\lambda, \mathtt{Rng}_\lambda)$ $K \leftarrow_{\$} \mathtt{Keys}_\lambda$. **proc. Function**$(\phi, x)$: return $G(\phi(K), x)$. **proc. Finalise**(b): return b.

Figure 2.5: Games for RKA-PRF security. Game RKA-PRFReal is on the left, RKA-PRFRand on the right.

Definition 2.2.5 (Φ-restricted adversary). Consider an adversary playing the game in Figure 2.5. If all functions ϕ appearing in **Function** queries are such that $\phi \in \Phi$, then we say that the adversary is Φ-restricted.

It should be clear that there exist some functions for which it will be impossible to achieve security. For example, if an adversary submits a query (ϕ, x) for some constant function ϕ (i.e. $\phi(x) = C$ for all x), then the adversary can easily compute $F(C, x)$ and compare it to the value output by the oracle. This is why we introduce the previous definition and restrict our attention to adversaries that only use certain function classes (which necessarily exclude constant functions, as we just saw).

Definition 2.2.6. Let $F : \mathtt{Keys}_\lambda \times \mathtt{Dom}_\lambda \to \mathtt{Rng}_\lambda$ be a family of functions. The advantage of an *RKA-PRF* adversary $\mathcal{A}$ against F is

$$\mathbf{Adv}^{\text{rka-prf}}_{F,\mathcal{A}}(\lambda) := \mathbb{P}[\text{RKA-PRFReal}^{\mathcal{A}}_{F}(\lambda) \Rightarrow 1] - \mathbb{P}[\text{RKA-PRFRand}^{\mathcal{A}}_{\$}(\lambda) \Rightarrow 1]$$

where the games PRFReal and PRFRand are defined in Figure 2.5. We say F is Φ-RKA-PRF secure if the advantage of any Φ-restricted, polynomial time adversary is negligible in the security parameter λ.

2.2.7 Key derivation functions

A Key Derivation Function (KDF) is a deterministic function that takes a uniformly random input and outputs a string that is indistinguishable from random, which, as the name suggests, will be used as a key for a cryptographic primitive.

Definition 2.2.7. Let λ be a security parameter. Consider a polynomial time function f that maps from some domain to $\{0,1\}^{l_1(\lambda)}$. The advantage of a *KDF* adversary $\mathcal{A}$ against f is

$$\mathbf{Adv}^{\text{kdf}}_{f,\mathcal{A}}(\lambda) := \mathbb{P}[\text{KDFReal}^{\mathcal{A}}_{f}(\lambda) \Rightarrow 1] - \mathbb{P}[\text{KDFRand}^{\mathcal{A}}_{\$}(\lambda) \Rightarrow 1]$$

where games KDFReal and KDFRand are defined in Figure 2.6. We say f is a secure Key Derivation Function (KDF) if the advantage of any polynomial time adversary is negligible in the security parameter λ.

proc. Initialise(λ): return $(\mathcal{R}(f), \mathcal{D}(f))$.	**proc. Initialise**(λ): return $(\mathcal{R}(f), \mathcal{D}(f))$.
proc. Oracle: $k \leftarrow_{\$} \mathcal{D}(f)$ return $f(k)$.	**proc. Oracle**: $r \leftarrow_{\$} \mathcal{R}(f)$ return r.
proc. Finalise(b): output b.	**proc. Finalise**(b): output b.

Figure 2.6: Games for KDF security. Game KDFReal is on the left, KDFRand is on the right.

2.3 Random Oracle Model

In this section we briefly introduce the random oracle methodology, which was first formalised by Bellare and Rogaway [9]. Security games that do no employ random oracles are said to be in the 'standard model'. In essence, a random oracle will return an independent and uniformly random output for every unique input. That is, a random oracle is a truly random function, which we will denote by H throughout this thesis. The domain of the random oracle will be bit strings of arbitrary length, and the random oracle will return bit strings of a (specified) fixed length.

In the random oracle model, the security games will include an extra procedure by which all parties may query the random oracle. That is, both the challenger and adversary may query the random oracle. Furthermore, the outputs will be consistent for all queries. Therefore if two parties both query the random oracle with the bit string x, then the random oracle will return the same output to both parties. In security games and proofs, the random oracle is usually implemented by initialising an output array to be empty. When the random oracle receives a query x, it first checks whether $H(x)$ is defined. If it is, the oracle returns $H(x)$. Otherwise, if $H(x) = \perp$, the random oracle assigns

$H(x) \leftarrow_{\$} \mathcal{R}$, where $\mathcal{R}$ is the appropriate range, and then returns $H(x)$.

2.4 Coding Theory

Coding theory is concerned with communications over unreliable channels. That is, a message transmitted over the channel may have errors when it is received. When we receive a message m' over the unreliable channel, we would like to 'decode' m' to obtain the original message m that was sent over the channel. Coding theory deals with the design and study of error-correcting codes that are intended to ensure that the message m can be reliably recovered at the receiver's end of the channel.

Definition 2.4.1. Let $A = \{a_1, \ldots, a_q\}$ be an *alphabet*. A *code* C of length n over A is a subset of A^n. A vector $c \in C$ is called a *codeword*, and we let $|C|$ denote the *size* of the code. In this report we will only concern ourselves with the alphabet $A = \{0, 1\}$. A code over $A = \{0, 1\}$ is called a *binary code*.

Definition 2.4.2. Let C be a binary code of length n and size M. Then the *rate* of C is defined to be

$$R = \frac{\log_2 M}{n}.$$

Informally, the rate of a code measures the amount of redundancy in the representation of messages. Essentially, $\log_2 M$ represents the number of 'useful' bits (since only $\log_2 M$ bits are needed to represent M codewords). Therefore, the rate is measuring the ratio between the number of useful bits and the total number of bits representing each codeword.

Definition 2.4.3. Let x and y be binary strings, and let x_i and y_i denote the ith bit of x and y, respectively. Define d as follows:

$$d(x_i, y_i) = \begin{cases} 1 & x_i \neq y_i \\ 0 & x_i = y_i. \end{cases}$$

The *Hamming distance* between x and y is defined to be the sum of the d over all bit positions. That is,

$$d(x, y) = \sum_{i=1}^{n} d(x_i, y_i).$$

Thus, the Hamming distance between two strings is the number of positions in which they differ.

Before stating the main theorem from coding theory that we require, we must first briefly discuss the *capacity* of a channel. Suppose that we send a binary message over an unreliable channel that introduces errors. Knowledge of the probabilities of these errors allows us to compute the *channel capacity*. We will not discuss the formal definition of capacity (or how to compute it), but we note that the channel capacity and code rate are closely connected, as the following well-known theorem demonstrates.

Theorem 2.4.1 (Shannon's noisy-channel coding theorem). For any $\epsilon > 0$ and code of rate R that is less than the channel capacity C, there is a decoding algorithm whose error probability is less than ϵ, for a sufficiently large code length. Also, for any rate greater than the channel capacity, the probability of error at the receiver goes to one as the code length goes to infinity.

Part I

Cold Boot Attacks

Chapter 3

Cold Boot Attacks in the RSA Setting

3.1 Introduction

Cold boot attacks [36, 37] are a class of attacks wherein memory remanence effects are exploited to extract data from a computer's memory. The idea is that modern computer memories retain data for periods of time after power is removed, so an attacker with physical access to a machine may be able to recover, for example, cryptographic key information. The time during which data is retained can be significantly increased by cooling the memory chips. For example, according to [37], in an experiment at $-50\,^{\circ}\mathrm{C}$ (obtained by spraying compressed air onto the memory chips) less than 0.1% of bits decay within sixty seconds. At temperatures of approximately $-196\,^{\circ}\mathrm{C}$ (via the use of liquid nitrogen) less than 0.17% of bits decay within one hour without power. Because the memory gradually degrades over time once power is removed, only a noisy version of the original data may be recoverable. The question then naturally arises: given a noisy version of a cryptographic key, is it possible to reconstruct the original key? In the current chapter we will attempt to answer this question for RSA keys.

This question was addressed for broad classes of cryptosystems, both symmetric and asymmetric, by Halderman et al. in [36, 37]. The schemes considered include RSA, AES, DES and various tweakable encryption modes. The question was addressed specifically for RSA private keys in [42, 44]. Similar problems arise in the context of side-channel analysis of cryptographic implementations, where noisy key information may leak through power consumption [50] or timing [19]. The question is also linked to the classical cryptanalysis problem of recovering an RSA private key when some bits of the key are known, for example the most or least significant bits, or contiguous bits spread over a number of blocks (see, for example, the surveys in [16, 59] and [45]).

Heninger and Shacham (HS) [44] considered the setting where a random fraction of the RSA private key bits is known with certainty. Their approach exploits the fact that the individual bits of an RSA private key of the form $\mathsf{sk} = (p, q, d, d_p, d_q)$ must satisfy certain algebraic relations (which we shall encounter shortly). Here, p, q and d are exactly the same variables we encountered in the 'textbook' RSA example in Section 2.2.1. The value d_p represents $d \bmod p - 1$, and d_q is similarly defined. In order to decrypt messages, it is only necessary to store the value d in the private key, but the PKCS [46] standard recommends additionally storing the parameters p, q, d_p and d_q in order to increase the efficiency of decryption via Chinese Remainder Theorem techniques. This redundancy in the key enables the recovery of the private key in a bit-by-bit fashion, starting with the least significant bits, by growing a search tree. It is easy to prune the search tree to remove partial solutions that do not match with the known key bits. The resulting algorithm will always succeed in recovering the private key, since the pruning process will never remove a partial version of the correct solution. On the other hand, when only few bits are known, the search tree may grow very large, and the HS algorithm will blow up. It was proved in [44] that, under reasonable

assumptions concerning the bit-distributions of incorrect solutions, the HS algorithm will efficiently recover an n-bit RSA private key in time $\mathcal{O}(n^2)$ with probability $1 - 1/n^2$ when a random fraction of at least 0.27 of the private key bits is known with certainty. These theoretical results are well-matched by experiments reported in [44]. These experiments also confirm that the HS algorithm has good performance when the known fraction is as small as 0.24 even for keys as large as 8192 bits, and the analysis of [44] extends to cases where the RSA private key sk is of the form (p, q, d) or (p, q). When sk is of the form (p, q, d), the algorithm theoretically requires at least a fraction of 0.42 known bits. When sk is of the form (p, q), the fraction required rises to 0.57.

Henecka, May and Meurer (HMM) [42] took the ideas of [44] and developed them further to address the situation where no RSA private key bits are known with certainty. They consider the *symmetric* case where the two possible bit flips ($0 \to 1$ and $1 \to 0$) have equal probability δ. Their main idea was to consider t bit-slices at a time of possible solutions to the equations relating the bits of sk, instead of single bits at a time as in the HS algorithm. In the formulation where $\mathsf{sk} = (p, q, d, d_p, d_q)$, this yields 2^t candidate solutions on $5t$ new private key bits for each starting candidate at each stage of the algorithm. The HMM algorithm then computes the Hamming distance between the candidate solutions and the noisy key, keeping all candidates for which this metric is less than some carefully chosen threshold C.[1] This replaces the procedure of looking for exact matches used in the HS algorithm. Of course, now the correct solution may fail this statistical test and be rejected; moreover the number of candidate solutions retained may explode if C is set too loosely. Nevertheless, it was shown in [42] that the HMM algorithm runs in

[1]HMM actually computed the number of bit matches between the noisy version and the candidates, and the candidates with more than C matches were kept. However, this is equivalent to using Hamming distance, albeit with a different threshold C and keeping candidates with a Hamming distance less than C.

polynomial time and has reasonable success in outputting the correct solution provided that $\delta < 0.237$. Again, the analysis depends on assumptions concerning the random behaviour of incorrect solutions. To support the analysis, [42] reports the results of experiments for different noise levels and algorithmic parameters. For example, with $\mathsf{sk} = (p, q, d, d_p, d_q)$, the algorithm can cope with $\delta = 0.20$, having an experimental success rate of 21% and a running time of three minutes at this noise level. The HMM algorithm also generalises to the cases when $\mathsf{sk} = (p, q, d)$ or $\mathsf{sk} = (p, q)$. When $\mathsf{sk} = (p, q, d)$, the HMM algorithm can handle noise rates up to 0.16 theoretically, and managed 0.14 in experimental results. When $\mathsf{sk} = (p, q)$, their theoretical limit is 0.084, and in practice they achieved results for $\delta \leq 0.08$.

3.1.1 Limitations of previous work and open questions

Although inspired by cold boot attacks, it transpires that neither the HS algorithm nor the HMM algorithm actually solves the motivating cold boot problem. Let us see why.

One observation made in [36, 37] is that for a given region of memory, the decay of memory bits is overwhelmingly either $0 \to 1$ or $1 \to 0$. The type of decay depends on the so-called 'ground state' of the particular memory region. Each region has the ground state set to either 0 or 1. In a particular ground state, all bits are hard-wired to either 1 or 0. Typically, bits will degrade to the ground state, but in rare cases it is possible for a bit to decay in the opposite direction. The ground state of a particular region can usually be inferred from the distribution of 0 and 1 bits. For an uncorrupted private key, we expect the number of 1s and 0s to be approximately equal. Therefore, if we observe many more 0s than 1s, we can assume that we are in a $1 \to 0$ region, and vice versa. As a result of these decay patterns, in a $1 \to 0$ region, a 1 bit in the

noisy version of the key is known (with high probability) to correspond to a 1 bit in the original key.

In the case of [44], the assumption is made that a certain fraction of the RSA private key bits – both 0s and 1s – is known with certainty. But, in the cold boot scenario, only 1 (or 0) bits are known (in a particular region), and not a mixture of both. Fortunately, the authors of [44] have informed us that their algorithm does still work when only 0 or only 1 bits are known, but this is not the case it was designed for, and, formally, the performance guarantees obtained in [44] do not apply in this case. Furthermore, in a real cold boot attack, bits are never known with *absolute* certainty, because even in a $1 \to 0$ region, say, bit flips in the reverse direction can occur. Halderman et al. [36] report rates of 0.05% to 0.1% for this event. Such an event will completely derail the HS algorithm, as it will result in the correct solution being eliminated from the search tree. Based on an occurrence rate of 0.1%, this kind of fatal event can be expected to arise around 2.5 to five times in a real key recovery attack for 1024-bit RSA moduli with $\mathsf{sk} = (p, q, d, d_p, d_q)$. Naturally, one could correct for this by re-running the HS algorithm many times with a small subset of bits being flipped in sk each time. However, this would be highly inefficient, as well as inelegant. Thus, the HS algorithm really only applies to an 'idealised' cold boot setting, where some bits are known for sure.

The HMM algorithm is designed to work for the symmetric case where the two possible bit flips have equal probability δ. Yet, in a cold boot attack, in a $1 \to 0$ region say, $\alpha := \mathbb{P}(0 \to 1)$ will be very small (though non-zero), while $\beta := \mathbb{P}(1 \to 0)$ may be relatively large, and perhaps even greater than 0.5 in a very degraded case. The use of Hamming distance as a metric for comparison and the setting of the threshold C are closely tied to the symmetric case, and it is not immediately clear how one can generalise the HMM approach to

handle the type of errors occurring in real cold boot attacks. So, while the HMM algorithm may be appropriate for RSA private key recovery in some side-channel settings (such as power analysis attacks), it does not solve the cold boot problem for RSA keys.

Intriguing features of the work in [44, 42] are the constants 0.27 and 0.237, which bound the fraction of known bits/noise rate the HS and HMM algorithms can handle. One can trace through the relevant analysis to see how these numbers emerge, but it would be more satisfying to have a deeper, unifying explanation. One might also wonder if these bounds are the best possible or whether significant improvements might yet be forthcoming. Is there any ultimate limit to the noise level with which these kinds of algorithms can cope? And can we design an algorithm that works in the true cold boot setting, or for fully general noise models that might be expected to occur in other types of side channel attack?

3.1.2 Our contributions

In this chapter we show how to recast the problem of noisy RSA key recovery as a problem in coding theory. That such a connection exists should be no surprise: after all, we are in a situation where bits have an associated probability distribution and we wish to recover the true bits. However, this connection opens up the opportunity to apply to our problem the full gamut of sophisticated tools that have been developed by coding theorists over the last sixty years. We sketch this connection and its main consequences next, with the details to come in the later sections of this chapter.

Recall that in the HMM algorithm, we generate from each solution so far a set of 2^t candidate solutions on $5t$ new bits. We now view the set of 2^t candidates as being a *code*, with one codeword $\mathtt{s}$ (representing bits of the true

private key) being selected and transmitted over a noisy channel, resulting in a received word r (representing $5t$ bits of the noisy version of the key). In the HMM case, the noise is realised via bit-flipping with probability δ. The HS algorithm can be seen as arising from the special case $t = 1$, where the noise now corresponds to erasing a fraction of key bits instead of flipping them. Alternatively, we can consider a generalisation of the HS algorithm which considers $5t$ bits at a time, generated just as in the HMM algorithm, and which then filters the resulting 2^t candidates based on matching with known key bits. Because filtering is based on exact matching, this algorithm has the same output as the original HS algorithm.[2] This brings the two algorithms under a single umbrella.

In general, in coding theory, the way in which s is transformed into r depends on the *channel model*, which in its full generality defines the probabilities $\mathbb{P}(\mathsf{r}|\mathsf{s})$ over all possible pairs (s, r). In the case of [44], the assumption is that particular bits are known with certainty and others are not known at all, with the bits all being treated independently. The appropriate channel model is then an *erasure* channel, meaning that bits are independently either erased or transmitted correctly over the channel, with the receiver knowing the positions of the erasures. In the case of [42], the appropriate channel model is the binary symmetric channel with cross-over probability δ. It also emerges that the appropriate channel model for the true cold boot setting is a binary *non-symmetric* channel with cross-over probabilities (α, β). In general, the problem we are faced with is to decode r, with the aim being to reproduce s with as high a probability as possible.

When formulated in this language, it becomes obvious that the HS and HMM algorithms do not solve the original cold boot problem – simply put,

[2]However, in practice the running-time will be slightly greater because incorrect candidates will not be discarded at the earliest possible moment.

these algorithms use inappropriate channel models for that specific problem. We can also use this viewpoint to derive limits on the performance of *any* procedure for selecting which candidate solutions to keep in an HMM-style algorithm. To see why, we recall that the converse to Shannon's noisy-channel coding theorem (see Section 2.4 or [74]) states that *no* combination of code and decoding procedure can jointly achieve arbitrarily reliable decoding when the code rate exceeds the (Shannon) capacity of the channel. There is a subtle technicality here: the converse to Shannon's theorem applies only to decoding algorithms that output a single codeword s, while both the HS and HMM algorithms are permitted to output many candidates at each stage, with the final output list only being required to contain the correct private key. The correct key can then be determined by, for example, using a trial encryption and decryption. It is then plausible that such list-outputting algorithms might surpass the bounds imposed by the converse to Shannon's theorem. However, this is not the case for the erasure channel and the binary symmetric channel: there are analogues of the converse of Shannon's theorem for so-called *list decoding* that essentially show that channel capacity is also the barrier to any efficient algorithm outputting lists of candidates for these channels, as the HS and HMM algorithms do.

When sk is of the form (p, q, d, d_p, d_q), for example, the code rate is fixed at $1/5$ (we have 2^t codewords of length $5t$, so the rate is $\log 2^t/5t$). The channel capacity can be calculated as a function of the channel model and its parameters. For example, for the erasure channel with erasure probability ρ (meaning that a fraction $1 - \rho$ of the bits are known with certainty), the capacity is simply $1 - \rho$. Then we see that the code rate exceeds capacity whenever we have $\rho \geq 0.8$, meaning that the fraction of known bits must be *at least* 0.2 to achieve arbitrarily reliable, efficient decoding. The analysis in [44] needs that fraction to be at least 0.27, though a fraction as low as 0.24 could

be handled in practice. Thus a capacity analysis suggests that there should be room to improve the HS algorithm further, but capacity shows that it is impossible to go below a fraction 0.2 of known bits with an efficient algorithm. Similar remarks apply to the HMM algorithm: here the relevant cross-over probability δ at which a capacity of $1/5$ is reached is $\delta = 0.243$, which is in remarkable agreement with the maximum value of 0.237 for δ arising in the analysis of [42], but still a little above the figure of 0.20 achieved experimentally in [42]. Again, the capacity analysis indicates that going above a noise level of 0.243 is likely to be very difficult, if not impossible, for an efficient HMM-style algorithm. See Section 3.3 for further details on list decoding and its application to the analysis of the HS and HMM algorithms.

Informed by our coding-theoretic viewpoint, we derive a new key recovery algorithm that works for any (memoryless) binary channel and therefore *is* applicable to the cold boot setting (and more), in contrast to the HS and HMM algorithms. In essence, we modify the HMM algorithm to use a likelihood statistic in place of the Hamming metric when selecting from the candidate codewords. We keep the L codewords having the highest values of this likelihood statistic and reject the others; from a coding perspective, our algorithm uses maximum likelihood list decoding, where L is the size of the list, which is a user-specified parameter. An important consequence of this algorithmic choice is that our algorithm has *deterministic* running time $\mathcal{O}(L2^t n/t)$ and, when implemented using a stack, *deterministic* memory consumption $\mathcal{O}(L+t)$. This stands in contrast to the running time and memory usage of the HS and HMM algorithms, which may blow up when the erasure/error rates are high. We note that private RSA keys are big enough that they may cross regions when stored in memory. We can handle this by changing the likelihood statistic used in our algorithm at the appropriate transition points, requiring only a simple modification to our approach.

Also using coding-theoretic tools, we are able to give an analysis of the success probability of our new algorithm. We show that, as $t \to \infty$, its success probability tends to 1 provided the code rate (1/5 when $\mathsf{sk} = (p, q, d, d_p, d_q)$) remains below the channel capacity. Moreover, from the converse to Shannon's theorem, we are unlikely to be able to improve this result if reliable key recovery is required. Our analysis is very simple but non-rigorous: it assumes the code behaves like a random code and follows from a direct application of Shannon's noisy-channel coding theorem. We note that it seems unlikely that a rigorous proof for the full case will be easy to obtain: such a proof would likely yield a Shannon-style random coding bound for list decoding on non-symmetric channels, and such bounds are (to the best of our knowledge) not currently known, despite list decoding having been the subject of many decades of study in the information theory community.

As a complement to our theoretical analysis, and as validation of it, we include the results of extensive experiments using our new algorithm. These demonstrate that our approach matches or outperforms the HS and HMM algorithms in the cases they are designed for, and achieves results close to the limits imposed by our capacity analysis more generally. For example, in the symmetric case with $\delta = 0.20$, we can achieve a 19% success rate in recovering keys for $t = 18$ and $L = 32$. This is comparable to the results of [42]. Furthermore, for the same t and L we achieve a 3% success rate for $\delta = 0.22$, whilst [42] does not report any experiments for an error rate this high. As another example, our algorithm can handle the idealised cold boot scenario by setting $\alpha = 0$ (in which case all the 1 bits in r are known with certainty, i.e. we are in a $1 \to 0$ region). Here, our capacity analysis puts a bound of 0.666 on β for reliable key recovery. Using our algorithm, we can recover keys for $\beta = 0.6$ with a 14% success rate using $t = 18$ and $L = 32$, whereas the HS algorithm can only reach $\beta = 0.52$ (and this under the assumption that the

experimental results reported in [44] for a mixture of known 0 and 1 bits do translate to the same performance for the case where only 1 bits are known). In the same setting, we can even recover keys up to $\beta = 0.63$ with a non-zero success rate. We also have similar experimental results for the 'true' cold boot setting where both α and β are non-zero, and for the situation where sk is of the form (p, q, d) or (p, q).

3.1.3 Further related work

In recent work that is independent of ours, Sarkar and Maitra [73] revisited the work of [42], applying the HMM algorithm to break Chinese Remainder implementations of RSA with low weight decryption exponents and giving ad hoc heuristics to improve the algorithm. Also in recent independent work, Kunihiro et al. [52] generalised the work of [44] and [42] to consider a combined erasure and symmetric error setting, where each bit of the private key is either erased or flipped. The practical motivation for considering this type of channel is unclear.

Key recovery for various types of symmetric key have been considered in [76, 47, 2].

3.2 The HS and HMM algorithms

In this section we describe the work of [44, 42], maintaining the notation of these papers as far as is possible.

Let (N, e) be the RSA public key, where $N = pq$ is an n-bit RSA modulus, and p, q are balanced primes. As with [44, 42], we assume throughout that e is small, say $e = 3$ or $e = 2^{16}+1$; for empirical justification of this assumption, see

[82]. We start by assuming that private keys sk follow the PKCS#1 standard and so are of the form $(N, p, q, e, d, d_p, d_q, q_p^{-1})$, where d is the decryption key, $d_p = d \mod p-1$, $d_q = d \mod q-1$ and $q_p = q^{-1} \mod p$. However, neither the algorithms of [44, 42] nor ours make use of q_p^{-1}, so we henceforth omit this information. Furthermore, we assume N and e are publicly known, so we work only with the tuple $\mathsf{sk} = (p, q, d, d_p, d_q)$. We will also consider attacks where the private key contains less information – either $\mathsf{sk} = (p, q, d)$ or $\mathsf{sk} = (p, q)$.

Now assume we are given a degraded version of the key $\widetilde{\mathsf{sk}} = (\tilde{p}, \tilde{q}, \tilde{d}, \tilde{d}_p, \tilde{d}_q)$. We start with the four RSA equations:

$$\begin{aligned} N &= pq && (3.2.1)\\ ed &= k(N-p-q+1)+1 && (3.2.2)\\ ed_p &= k_p(p-1)+1 && (3.2.3)\\ ed_q &= k_q(q-1)+1, && (3.2.4) \end{aligned}$$

where k, k_p and k_q are integers to be determined. A method for doing so is given in [44]: first it is shown that $0 < k < e$; then, since e is small, we may enumerate

$$d(k') := \left\lfloor \frac{k'(N+1)+1}{e} \right\rfloor$$

for all $0 < k' < e$. We then find the k' such that $d(k')$ is 'closest' to $\tilde{d}$ in the most significant half of the bits. Simple procedures for doing this are given in [44, 42]. In the more general setting where bit flips can occur in both directions and with different probabilities, we proceed as follows. First, we estimate parameters $\alpha = \mathbb{P}(0 \to 1)$ and $\beta = \mathbb{P}(1 \to 0)$ from known bits. This may be done, for example, by comparing the public key with its noisy version from the degraded memory [42]. Second, we compute for each k' an approximate log-likelihood using the expression

$$n_{01} \log \alpha + n_{00} \log(1-\alpha) + n_{10} \log \beta + n_{11} \log(1-\beta)$$

where n_{01} is the number of positions in the most significant half where a 0 appears in $d(k')$ and a 1 appears in $\tilde{d}$, etc (see page 63 for an explanation of this expression). Finally, we select the k' that provides the highest log-likelihood.

At the end of this procedure, with high probability we will have $k' = k$ and we will have recovered the most significant half of the bits of d. Now we wish to find k_p and k_q. By manipulating the above equations we see that

$$k_p^2 - (k(N-1)+1)k_p - k \equiv 0 \mod e.$$

If e is prime (as in the most common case $e = 2^{16}+1$) there will only be two solutions to this equation. One will be k_p and the other k_q. If e is not prime we will have to try all possible pairs of solutions in the remainder of the algorithm.

Now, for integers x, we define $\tau(x) := \max\{i \in \mathbb{N} : 2^i \mid x\}$. Then it is easy to see that $2^{\tau(k_p)+1}$ divides $k_p(p-1)$, $2^{\tau(k_q)+1}$ divides $k_q(q-1)$ and $2^{\tau(k)+2}$ divides $k\phi(N)$. These facts, along with relations (3.2.2) – (3.2.4), allow us to see that

$$\begin{aligned} d_p &\equiv e^{-1} \mod 2^{\tau(k_p)+1} \\ d_q &\equiv e^{-1} \mod 2^{\tau(k_q)+1} \\ d &\equiv e^{-1} \mod 2^{\tau(k)+2}. \end{aligned}$$

This allows us to correct the least significant bits of d, d_p and d_q. Furthermore we can calculate $\mathsf{slice}(0)$, where we define

$$\mathsf{slice}(i) := (p[i], q[i], d[i+\tau(k)], d_p[i+\tau(k_p)], d_q[i+\tau(k_q)]),$$

with $x[i]$ denoting the ith bit of the string x.

Now we are ready to explain the main idea behind the algorithm of [44]. Suppose we have a solution (p', q', d', d'_p, d'_q) from $\mathsf{slice}(0)$ to $\mathsf{slice}(i-1)$. Then

[44] uses a multivariate version of Hensel's Lemma to show that the bits involved in $\mathsf{slice}(i)$ must satisfy the following congruences:

$$\begin{aligned}
p[i] + q[i] &= (N - p'q')[i] \mod 2 \\
d[i + \tau(k)] + p[i] + q[i] &= (k(N+1) + 1 - k(p' + q') - ed')[i + \tau(k)] \mod 2 \\
d_p[i + \tau(k_p)] + p[i] &= (k_p(p' - 1) + 1 - ed'_p)[i + \tau(k_p)] \mod 2 \\
d_q[i + \tau(k_q)] + q[i] &= (k_q(q' - 1) + 1 - ed'_q)[i + \tau(k_q)] \mod 2.
\end{aligned}$$

Because we have four constraints on five unknowns, there are exactly two possible solutions for $\mathsf{slice}(i)$, rather than thirty-two. This is then used in [44] as the basis of building a search tree for the unknown private key bits. At each node in the tree, representing a partial solution up to $\mathsf{slice}(i-1)$, at most two successor nodes are added by the above procedure. Moreover, since a random fraction of the bits is assumed to be known with certainty, the tree can be pruned of any partial solutions that are not consistent with these known bits. Clearly, if the fraction of known bits is large enough, then the tree will be highly pruned and the number of nodes in the tree will be small. The analysis of [44] shows that if the fraction of known bits is at least 0.27, then the tree's size remains close to linear in n, the size of the RSA modulus, meaning that an efficient algorithm results. A similar algorithm and analysis can be given for the case where sk is of the form (p, q, d) or (p, q); in each case, there are exactly two possible solutions for each $\mathsf{slice}(i)$.

Instead of doing Hensel lifting bit-by-bit and pruning on each bit, the HMM algorithm performs t Hensel lifts for some parameter t, yielding, for each surviving candidate solution on $\mathsf{slice}(0)$ to $\mathsf{slice}(i-1)$, a tree of depth t whose 2^t leaf nodes represent candidate solutions on slices $\mathsf{slice}(0)$ to $\mathsf{slice}(i+t-1)$, involving $5t$ new bits (in $\mathsf{slice}(i)$ to $\mathsf{slice}(i+t-1)$). A solution is kept for the next iteration if the Hamming distance between the $5t$ new bits and the corresponding vector of noisy bits is less than some threshold C. Clearly the

HS algorithm could also be modified in this way, lifting t times and then doing pruning based on matching known key bits. Alternatively, one can view the HS algorithm as being the special case $t = 1$ of the HMM algorithm (with a different pruning procedure). The HMM algorithm can also be adapted to work with sk of the form (p, q, d) or (p, q). Henecka et al. [42] showed how to select C and t so as to guarantee that their algorithm is efficient and produces the correct solution with a reasonable success rate. In particular, they were able to show that this is the case provided the probability of a bit flip δ is at most 0.237.

Some additional remarks on these algorithms follow. Firstly, the HMM algorithm is iterative: at each stage in the algorithm, candidate solutions on t new slices are constructed. Then roughly $n/2t$ iterations or stages of the algorithm are needed, since all the quantities being recovered contain at most $n/2$ bits. As pointed out in [42], only half this number of stages is required since once we have the least significant half of the bits of the private key, the entire private key can be recovered using a result of Coppersmith [20]. Secondly, the analysis in [42] is based on the heuristic that every candidate solution on bit slices i to $i + t - 1$ 'is an ensemble of t randomly chosen bit slices' when the starting solution on bit slices 0 to $i - 1$ is an incorrect solution. Equivalently, it is assumed that each of the 2^t vectors of $5t$ bits $\mathsf{s} = (\mathsf{slice}(i), \ldots, \mathsf{slice}(i + t - 1))$ representing all candidate solutions generated from a single incorrect partial solution is uniformly distributed. This seems to be well-supported by experimental evidence [44, 42]. Note also that, in the analysis of [42], these 2^t vectors do not need to be independent of one another, though independence of the bits of any given candidate is needed at one point in the analysis of [42] (in order to be able to apply Hoeffding's bound). Thirdly, at their conclusion, the HS and HMM algorithms output lists of candidate solutions rather than a single solution, but it is easy to verify the

correctness of each candidate by using a trial encryption and decryption, say. Thus the success rate of the algorithms is defined to be the probability that the correct solution is on the output list. We adopt the same measure of success in the remainder of this chapter. However, if the output list becomes too large, then it is clear that the algorithm will be inefficient. Ultimately, we seek a trade-off between the success and efficiency.

3.3 The Coding-Theoretic Viewpoint

In this section, we develop our coding-theoretic viewpoint on the HS and HMM algorithms, using it to derive limits on the performance of these and similar algorithms. In particular, we will explain how channel capacity plays a crucial role in setting these limits.

We begin by defining the parameter m. We set $m = 5$ when $\mathsf{sk} = (p, q, d, d_p, d_q)$, $m = 3$ when $\mathsf{sk} = (p, q, d)$, and $m = 2$ when $\mathsf{sk} = (p, q)$. Consider a stage of the HMM algorithm, commencing with M partial solutions that have survived the previous stage's pruning step. The HMM algorithm produces a total of $M2^t$ candidate solutions on mt bits, prior to pruning. We label these $\mathsf{s}_1, \ldots, \mathsf{s}_{M2^t}$, let $\mathcal{C}$ denote the set of all $M2^t$ candidates, and use r to denote the corresponding vector of mt noisy bits in sk.

Now we think of $\mathcal{C}$ as being a code. This code has rate $R \geq 1/m$, but its other standard parameters such as its minimum distance are unknown (and immaterial to our analysis). The problem of recovering the correct candidate s_c given r is clearly just the problem of decoding this code. Now both the HS and HMM algorithms have pruning steps that output lists of candidates for the correct solution, with the list size being dynamic in both cases and depending on the number of candidates surviving the relevant filtering process (based

either on exact matches for the HS algorithm or on Hamming distance for the HMM algorithm). In this sense, the HS and HMM algorithms are performing types of *list decoding*, an alternative to the usual unique decoding of codes that was originally proposed by Elias [27].

To complete the picture, we need to discuss what error and channel models are used in [44, 42], and what models are appropriate to the cold boot setting. As noted in the introduction, [44] assumes that some bits of r are known exactly, while no information at all is known about the other bits. This corresponds to an *erasure* model for errors, and an *erasure* channel. Usually, this is defined in terms of a parameter ρ representing the fraction of erasures. So $1-\rho$ represents the fraction of known bits, a parameter denoted δ in [44]. On the other hand, [42] assumes that all bits of r are obtained from the correct s_c by independent bit flipping with probability δ. In standard coding terminology, we have a (memoryless) binary symmetric channel with crossover probability δ. From the experimental data reported in [36, 37], an appropriate model for the cold boot setting would be a binary non-symmetric channel with crossover probabilities (α, β), with α being small and β being significantly larger in a $1 \to 0$ region (and vice-versa in a $0 \to 1$ region). In an idealised cold boot case, we could assume $\alpha = 0$, meaning that a $0 \to 1$ bit flip can never occur, so that all 1 bits in r are known with certainty. This is better known as a Z-channel in the coding-theoretic literature.

This viewpoint highlights the exact differences between the settings considered in [44, 42] and the true cold boot setting. It also reveals that, while the HS algorithm can be applied for the Z-channel seen in the idealised cold boot setting, there is no guarantee that the performance proven for it in [44] for the erasure channel will transfer to the Z-channel. Moreover, one might hope for substantial improvements to the HS algorithm if one could somehow take into account the (partial) information known about 0 bits as well as the

exact information known about 1 bits.

3.3.1 The link to channel capacity

We can use this coding viewpoint to derive limits on the performance of any procedure for selecting which candidate solutions to keep in the HS and HMM algorithms. To see why, we recall that the converse to Shannon's noisy-channel coding theorem [74] states that no combination of code and decoding procedure can jointly achieve arbitrarily reliable decoding when the code rate exceeds the capacity of the channel. Our code rate is at least $1/m$ where $m = 2, 3$ or 5, and the channel capacity can be calculated as a function of the channel model and its parameters.

Two caveats must be made here. Firstly, capacity only puts limits on *reliable* decoding, and even decoding with low success probability is of interest in cryptanalysis. Secondly, Shannon's result applies only to decoding algorithms that output a single codeword s, while both the HS and HMM algorithms are permitted to output many candidates at each stage, with the final output list only being required to contain the correct private key. Perhaps such list-outputting algorithms can surpass the bounds imposed by Shannon's theorem? Indeed, the HS algorithm is guaranteed to output the correct key provided the algorithm terminates. Similarly, the threshold C in the HMM algorithm can always be set to a value that ensures that every candidate passes the test and is kept for the next stage, thus guaranteeing that the algorithm is always successful. However, neither of these variants would be *efficient* and in fact there are analogues of the converse of Shannon's noisy-channel coding theorem that essentially show that capacity is the barrier for efficient list decoding too.

For the binary symmetric channel, it is shown in [33, Theorem 3.4] that if $\mathcal{C}$ is *any* code of length $n = mt$ and rate $1 - H_2(\delta) + \epsilon$ (for some $\epsilon > 0$), then

there exists a word r is such that the Hamming sphere of radius δn around r contains at least $2^{\epsilon n/2}$ codewords. Here $H_2(\cdot)$ is the binary entropy function:

$$H_2(x) = -x \log_2(x) - (1-x) \log_2(1-x),$$

and $1 - H_2(\delta)$ is just the capacity of the channel. The proof also shows that, over a random choice of r, the average number of codewords in a sphere of radius δn around r is $2^{\epsilon n/2}$. Since the expected number of errors in r is δn, we expect the correct codeword to be in this sphere, along with $2^{\epsilon n/2}$ other codewords. This implies that, if the rate of the code exceeds the channel capacity $1-H_2(\delta)$ by a constant amount ϵ, then $\mathcal{C}$ cannot be list decoded using a polynomial-sized list, either in the worst case or on average, as $n \to \infty$.

An analogous result can be proved for the erasure channel, based on a similarly simple counting argument as was used in the proof of [33, Theorem 3.4]: if ρ is the erasure probability and $\mathcal{C}$ is any code of rate $1 - \rho + \epsilon$ (i.e. ϵ above the erasure channel's capacity), then it can be shown that on average there will be $2^{\epsilon n}$ codewords that differ from r in its erasure positions, assuming r contains ρn erasure symbols. Hence, reliable list decoding for $\mathcal{C}$ cannot be achieved using a polynomial-sized list. The justification for this is as follows. Suppose $\mathcal{C}$ is a code of rate $1 - \rho + \epsilon$, where ρ is the erasure probability. Furthermore, suppose that we receive a codeword r that contains ρn errors (the expected number when degrading an n-bit string). There are $2^{\rho n}$ bit strings of length n that could have degraded to r, since there are two options (either a 1 or 0 bit) for each of the erasure symbols, of which there are ρn. Denote this set of strings as $\mathcal{B}$. We now need to calculate the expected size of $\mathcal{B} \cap \mathcal{C}$, the intersection of $\mathcal{B}$ and $\mathcal{C}$. We have that

$$\mathbb{E}(|\mathcal{B} \cap \mathcal{C}|) = \frac{|\mathcal{B}||\mathcal{C}|}{2^n}.$$

Now, we know that the rate of the code $\mathcal{C}$ is $(\log_2|\mathcal{C}|)/n$, but we also know

that the rate is $1 - \rho + \epsilon$, by assumption. Hence,

$$|\mathcal{C}| = 2^{n(1-\rho+\epsilon)}.$$

Therefore, it follows that

$$\begin{aligned} \mathbb{E}(|\mathcal{B} \cap \mathcal{C}|) &= \frac{|\mathcal{B}||\mathcal{C}|}{2^n} \\ &= \frac{2^{\rho n} \cdot 2^{n(1-\rho+\epsilon)}}{2^n} \\ &= 2^{\epsilon n}, \end{aligned}$$

which verifies that there will be an exponential number of codewords to consider when the code rate exceeds the channel capacity.

In the next sub-section, we will examine in more detail the implications of these results on list decoding for the HS and HMM algorithms.

3.3.2 Implications of the capacity analysis

3.3.2.1 The binary symmetric channel and the HMM algorithm:

If the HMM algorithm is to have reasonable success probability in recovering the key, then at each stage, it must set the threshold C in such a way that all words $\mathsf{s}_i \in \mathcal{C}$ with $d_H(\mathsf{s}_i, \mathsf{r}) \approx \delta mt$ are accepted by the algorithm. This is because mt bits are considered at each stage of the algorithm, and each bit has a probability δ of flipping. Hence, δmt is the expected number of errors occurring in r, and if the threshold is set below this value, then the correct codeword is highly likely to be rejected by the algorithm. Recall that we have rate $R \geq 1/m$. Now suppose δ is such that $R = 1 - H_2(\delta) + \epsilon$, for some $\epsilon > 0$, i.e. δ is chosen so that the code rate is just above capacity. Then the argument above shows that there will be on average at least $2^{\epsilon mt/2}$ codewords on the output list at each stage. Thus, as soon as δ is such that R exceeds capacity by a constant amount ϵ, then there must be a blow-up in

the algorithm's output size at each stage, and the algorithm will be inefficient asymptotically.

We write $C_{\mathrm{BSC}}(\delta) = 1 - H_2(\delta)$ for the capacity of the binary symmetric channel. Table 3.1 shows that $C_{\mathrm{BSC}}(\delta) = 0.2$ when $\delta = 0.243$. Thus our capacity analysis shows that the best error rate one could hope to deal with in the HMM algorithm when $m = 5$ is $\delta = 0.243$. Notice that this value is rather close to, but slightly higher than, the corresponding value of 0.237 arising from the analysis in [42]. The same is true for the other entries in this table. This means that significantly improving the theoretical performance of the HMM algorithm whilst keeping the algorithm efficient will not be possible. The experimental work in [42] gives results up to a maximum δ of 0.20; compared to the capacity bound of 0.243, it appears that there is some room for practical improvement in the symmetric case. Similarly, when $m = 2$ and 3 the capacity analysis suggests that the theoretical limit for δ is 0.11 and 0.174, respectively. However, [42] only achieved experimental results for δ up to 0.08 and 0.14, respectively.

3.3.2.2 The erasure channel and the HS algorithm:

As noted above, for the erasure channel, the capacity is $1 - \rho$, where ρ is the probability that a bit is erased by the channel. Note that the list output by the HS algorithm is independent of whether pruning is done after each lift or in one pass at the end (but obviously doing so on a lift-by-lift basis is more efficient in terms of the total number of candidates examined). Then considering the HS algorithm in its entirety (i.e. over $n/2$ Hensel lifts), we see that it acts as nothing more than a list decoder for the erasure channel, with the code $\mathcal{C}$ being the set of all $2^{n/2}$ words on $mn/2$ bits generated by doing $n/2$ Hensel lifts without any pruning, and the received word r being the noisy

version of the entire private key sk.

Then our analysis above applies to show that the HS algorithm will produce an exponentially large output list, and will therefore be inefficient, when the rate (which in this case is exactly $1/m$) exceeds the capacity $1-\rho$. For $m = 5$, we have rate 0.2 and so our analysis shows that the HS algorithm will produce an exponentially large output list whenever ρ exceeds 0.8. Now [44] reports good results (in the sense of having a reasonable running time) for ρ as high as 0.76 (corresponding to Heninger and Shacham's parameter δ, the fraction of known bits in the private key, being equal to 0.24 and assuming that 1 and 0 bits are equiprobable in the private key), leaving a gap between the experimental performance and the theoretical bound. Similar remarks apply for the cases $m = 2$ and 3: for $m = 2$, the HS algorithm should be successful for $\rho = 0.43$ ($\delta = 0.57$), while the bound from capacity is 0.50; for $m = 3$, we have $\rho = 0.58$ ($\delta = 0.42$) and the capacity bound is 0.67. Hence, further improvements for $m = 2$ and 3 are not ruled out by the capacity analysis.

3.3.2.3 The Z-channel:

We may also apply the above capacity analysis to the idealised cold boot setting, where the crossover probabilities are of the form $(0, \beta)$, assuming we are in a $1 \to 0$ region. Here we have a Z-channel, whose capacity can be written as:

$$C_{\mathrm{Z}}(\beta) = \log_2(1 + (1-\beta)\beta^{\frac{\beta}{1-\beta}}).$$

Solving the equation $C_{\mathrm{Z}}(\beta) = R$ for $R = 1/5,\ 1/3,\ 1/2$ gives us the entries in Table 3.2. We point out the large gap between these figures and what we would expect to obtain both theoretically and experimentally if we were to directly apply the HS algorithm to the idealised cold boot setting. For example, when $m = 5$, the analysis of [44] suggests that key recovery should be successful

provided that β does not exceed 0.46 (the value of $\delta = 0.27$ translates into a β value of 0.46 via the formula $\delta = (1-\beta)/2$ given in [44]), whereas the capacity analysis suggests a maximum β value of 0.666. This illustrates that the HS algorithm is not well-matched to the Z-channel: an algorithm to recover the key should incorporate information from both 1 bits and possibly incorrect 0 bits, while the HS algorithm exploits only information obtained from the 1 bits known with certainty. Our new algorithm will substantially close the gap between the HS algorithm and the capacity analysis.

3.3.2.4 The true cold boot setting:

For the true cold boot setting, we must consider the general case of a memoryless, binary channel with crossover probabilities (α, β). We can calculate the capacity $C(\alpha, \beta)$ of this channel and obtain the regions for which $C(\alpha, \beta) > R$ for $R = 1/5,\ 1/3,\ 1/2$. The results are shown in Figure 3.1. Notice that these plots include as special cases the data from Tables 3.1 and 3.2. If we set $\alpha = 0.001$, say, we see that the maximum achievable β is quite close to that in the idealised cold boot setting. Note also that the plots are symmetric about the lines $y = x$ and $y = 1 - x$, reflecting the fact that capacity is preserved under the transformations $(\alpha, \beta) \to (\beta, \alpha)$ and $(\alpha, \beta) \to (1 - \alpha, 1 - \beta)$.

However, we must caution that capacity-based bounds for list decoding for the general binary non-symmetric channel (including the Z-channel) are not known in the coding-theoretic literature. Strictly speaking, then, our capacity analysis for this case does not bound the performance of key recovery algorithms that are allowed to output many key candidates, but only the limited class of algorithms that output a *single* key candidate. This said, our capacity analysis sets a target for our new algorithm, which follows.

sk	R	δ
(p, q, d, d_p, d_q)	$1/5$	0.243
(p, q, d)	$1/3$	0.174
(p, q)	$1/2$	0.110

Table 3.1: Private key-type, equivalent rate R, and maximum crossover probability δ allowing reliable key recovery, symmetric channel case.

sk	R	β
(p, q, d, d_p, d_q)	$1/5$	0.666
(p, q, d)	$1/3$	0.486
(p, q)	$1/2$	0.304

Table 3.2: Private key-type, equivalent rate R, and maximum error probability β allowing reliable key recovery, Z-channel case.

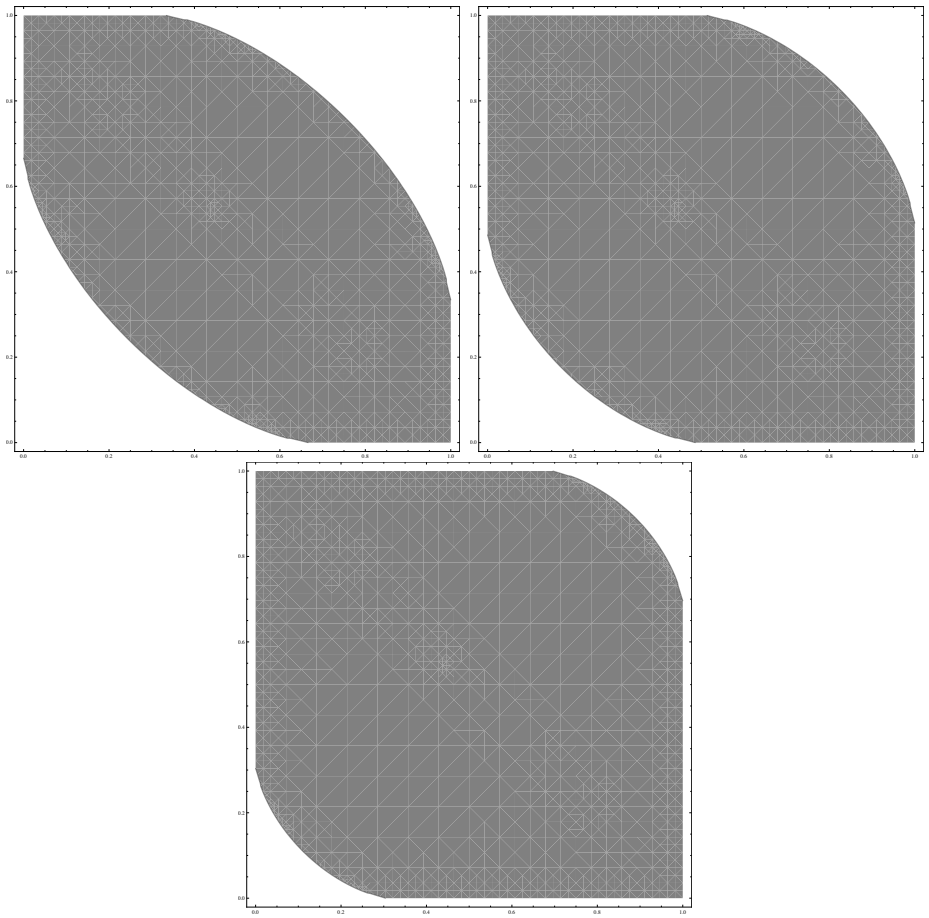

Figure 3.1: Plots showing achievable (α, β) pairs for private keys containing 5 components (top left), 3 components (top right) and 2 components (bottom). The vertical axis is β, the horizontal axis is α. Axes range from 0 to 1 at intervals of 0.2. The shaded area in each case represents the unachievable region.

3.4 The New Algorithm and Its Analysis

In this section we give our new algorithm for noisy RSA key recovery that works for any memoryless, binary channel, as characterised by the cross-over probabilities (α, β). Our algorithm has the same basic structure as the HMM algorithm, but uses a different procedure to decide which candidate solutions to retain and which to reject. Specifically, we use a likelihood measure in place of Hamming distance.

Recall that we label the $M2^t$ candidate solutions on mt bits arising at some stage in the HMM algorithm $\mathsf{s}_1, \ldots, \mathsf{s}_{M2^t}$, and let us name the corresponding vector of mt noisy bits in the RSA private key r. Then the Maximum Likelihood (ML) estimate for the correct candidate solution is simply:

$$\arg \max_{1 \le i \le M2^t} \mathbb{P}(\mathsf{s}_i | \mathsf{r}).$$

That is, the choice of i that maximises the conditional probability $\mathbb{P}(\mathsf{s}_i|\mathsf{r})$. Using Bayes' theorem, this can be rewritten as:

$$\arg \max_{1 \le i \le M2^t} \frac{\mathbb{P}(\mathsf{r}|\mathsf{s}_i)\mathbb{P}(\mathsf{s}_i)}{\mathbb{P}(\mathsf{r})}.$$

Here, $\mathbb{P}(\mathsf{r})$ is a constant for a given set of bits r. Let us make the further mild assumption that $\mathbb{P}(\mathsf{s}_i)$ is also a constant, independent of i. Then the ML estimate is obtained from

$$\arg \max_{1 \le i \le M2^t} (\mathbb{P}(\mathsf{r}|\mathsf{s}_i)) = \arg \max_{1 \le i \le M2^t} \left((1-\alpha)^{n_{00}^i} \alpha^{n_{01}^i} (1-\beta)^{n_{11}^i} \beta^{n_{10}^i} \right),$$

where $\alpha = \mathbb{P}(0 \to 1)$ and $\beta = \mathbb{P}(1 \to 0)$ are the crossover probabilities, n_{00}^i denotes the number of positions where s_i and r both have 0 bits, n_{01}^i denotes the number of positions where s_i has a 0 and r has a 1, and so on. Note that the previous equation used the fact that bit flips are assumed to be independent, so the expression is merely the product of the individual probabilities for each bit of the sent codeword.

Algorithm 1 Pseudo-code for the maximum likelihood list decoding algorithm for reconstructing RSA private keys.

Input: $\mathsf{list} \leftarrow \mathsf{slice}(0)$
Output: list
for $\mathsf{stage} = 1$ to $n/2t$ **do**
– Replace each entry in list with a set of 2^t candidate solutions obtained by Hensel lifting
– Calculate the log-likelihood $\log \mathbb{P}(\mathsf{r}|\mathsf{s}_i)$ for each entry s_i on list
– Keep the L entries in list having the highest log-likelihoods and delete the remainder.

Equivalently, we may maximise the log of these probabilities, and so we seek:

$$\begin{aligned}&\arg \max_{1 \le i \le M2^t} (\log \mathbb{P}(\mathsf{r}|\mathsf{s}_i)) \\ &\quad = \arg \max_{1 \le i \le M2^t} \left(n_{00}^i \log(1-\alpha) + n_{01}^i \log \alpha + n_{11}^i \log(1-\beta) + n_{10}^i \log \beta \right),\end{aligned}$$

which provides us with a simpler form for computational purposes.

Then our proposed algorithm is simply this: select at each stage from the candidates generated by Hensel lifting those L candidates s_i that produce the highest values of the log-likelihood $\log \mathbb{P}(\mathsf{r}|\mathsf{s}_i)$ as in the equation above. These candidates are then passed to the next stage. So at each stage, except the first, we will generate a total of $L2^t$ candidates and keep the best L. We may then test each entry in the final list by trial encryption and decryption to recover a single candidate for the private key. Pseudo-code for this algorithm is shown in Algorithm 1. Note that here we assume there are $n/2t$ stages; this number can be halved as in the HS and HMM algorithms by using the previously-mentioned methods of Coppersmith.

Our algorithm has fixed running time $O(L2^t)$ for each of the $n/2t$ stages, and fixed memory consumption $O(L2^t)$. This is a consequence of choosing to keep the L best candidates at each stage in place of all candidates surpassing some threshold as in the HMM algorithm. In practice, we do not generate all

$L2^t$ candidates and then filter. Instead, we generate the candidates using a depth-first search approach, implemented using a stack. The stack is initialised with the L starting candidates at each stage. We then filter each candidate as it is produced, adding it to the list only if its log-likelihood is superior to the worst entry currently on the list (expunging the worst entry if this is the case). This radically reduces the memory consumption of the algorithm for large t, from $O(L2^t)$ down to $O(L + t)$. The main overhead is then the Hensel lifting to generate candidate solutions; the subsequent computation of log-likelihoods for each candidate is relatively cheap. Notice that if $\alpha = 0$ (as in the Z-channel for an idealised cold boot setting), then any instance of a $0 \to 1$ bit flip is very heavily penalised by the log-likelihood statistic – it adds a $-\infty$ term to $\log \mathbb{P}(\mathsf{r}|\mathsf{s}_i)$. In practice, for $\alpha = 0$, we just reject any solution containing a $0 \to 1$ transition since we know that such an occurrence is impossible. For the erasure channel, we reject any candidate solution that does not match r in the known bits.

A special case of our algorithm arises when $L = 1$ and corresponds to just keeping the single ML candidate at each stage. This algorithm then corresponds to Maximum Likelihood (ML) decoding. However, at a given stage, it is likely that the correct solution will be rejected because an incorrect solution happens to have the highest likelihood. This is especially so in view of how similar some candidates will be to the correct solution. Therefore, ML decoding is likely to go awry at some stage of the algorithm.

This approach is broadly comparable to that taken in [42]: whereas Henecka et al. use a threshold to accept or reject solutions and thus also keep lists of 'likely' partial solutions, we avoid having to set a threshold and instead simply keep a list of the best L solutions according to our likelihood statistic. We reiterate that the approach of [42] is formulated only in a case equivalent to that of a binary symmetric channel, where the crossover probabilities are

equal, while our modified algorithm can handle this case, the Z-channel case implicit in [44], and a realistic model of the cold boot setting where bit flips in both directions are possible but may have substantially different probabilities.

3.4.1 Asymptotic analysis of our algorithm

We next give two analyses of our algorithm, using tools from coding theory to assist us. The first analysis uses a strong randomness assumption, that the $L2^t$ candidates s_i generated at each stage of Algorithm 1 are independent and uniformly random mt-bit vectors. It shows that, asymptotically, our algorithm will be successful in recovering the RSA private key provided $1/m$ is less than the capacity of the memoryless, binary channel with crossover probabilities (α, β). Unfortunately, it is easy to see that our strong randomness assumption is in fact not true for the codes $\mathcal{C}$ generated in our algorithm, because of the iterative nature of the Hensel lifting. The second analysis attempts to prove a similar result for the symmetric case under weaker (and therefore more realistic) randomness assumptions, but is unfortunately flawed.

3.4.1.1 First analysis:

The first analysis is very simple and is based on the following:

Strong randomness assumption: The $L2^t$ candidates s_c generated at each stage of Algorithm 1 are independent and uniformly random mt-bit vectors.

Our first analysis then proceeds as follows. As before we consider the set $\mathcal{C} = \{\mathsf{s}_i : 1 \leq i \leq L2^t\}$ of candidate solutions at a given stage as a code of length mt and rate $(t + \log_2 L)/mt$. One codeword s_c (corresponding to the correct private key bits) is selected and sent over a memoryless, binary

channel with crossover probabilities (α, β) to produce the received vector r. The problem of recovering the correct sent codeword, given r, is the problem of decoding this code. Under our strong randomness assumption, the set of $L2^t$ candidates at each stage is a random code. Now Shannon's noisy-channel coding theorem [74] states that, as $mt \to \infty$, the use of random codes in combination with Maximum Likelihood (ML) decoding achieves arbitrarily small decoding error probability, provided that the code rate stays below the capacity of the channel.[3] For fixed L and m, for our code, this holds provided $1/m$ is strictly less than the capacity as $t \to \infty$. Our algorithm does not quite implement ML decoding at each stage, but it always includes the ML candidate in its output list at each stage, since it selects the L most likely candidates. Hence, at each stage, it will be successful in including the correct candidate s_c in its output list provided that ML decoding is also successful. Finally the arbitrarily small error probability per stage guaranteed by Shannon's theorem translates into a success rate at each stage that can be made arbitrarily close to 1 as $t \to \infty$. Since the algorithm runs over $n/2t$ stages, by setting t appropriately as a function of n, we can achieve an overall success rate that is also arbitrarily close to 1.

Summarising, the above analysis shows that, asymptotically, our algorithm will be successful in recovering the RSA private key provided $1/m$ is less than the capacity of the memoryless, binary channel with crossover probabilities (α, β). This concludes the first analysis.

Unfortunately, it is easy to see that our strong randomness assumption is in fact not true for the codes $\mathcal{C}$ generated in our algorithm: because of the iterative nature of the Hensel lifting, the 2^t candidates arising from one starting point are arranged in a tree of depth t, with adjacent leaves in the

[3]The usual proof of Shannon's theorem does not employ ML decoding but instead uses a less powerful decoding algorithm based on *jointly typical* sequences. However ML decoding will do at least as well as any alternative procedure.

tree agreeing in the first $m(t-1)$ bits, groups of 4 leaves agreeing in the first $m(t-2)$ bits, and so on. Nevertheless, this strong assumption allows a very simple analysis of our algorithm, and the bounds that it yields are well-supported by our experiments to follow.

3.4.1.2 Second analysis:

Now we attempt to give a rigorous analysis of our algorithm under reasonable assumptions in the symmetric case (where $\alpha = \beta = \delta$).

At each stage the algorithm generates a forest of L trees (one for each of the L surviving candidates), which we denote $T_0, \ldots, T_{L-1}$, with each tree T_i being of depth t and having 2^t leaves (see Figure 3.2 for a visualisation). The internal nodes in each tree represent partial candidate solutions generated during the Hensel lifting and each leaf represents a candidate solution on mt bits.

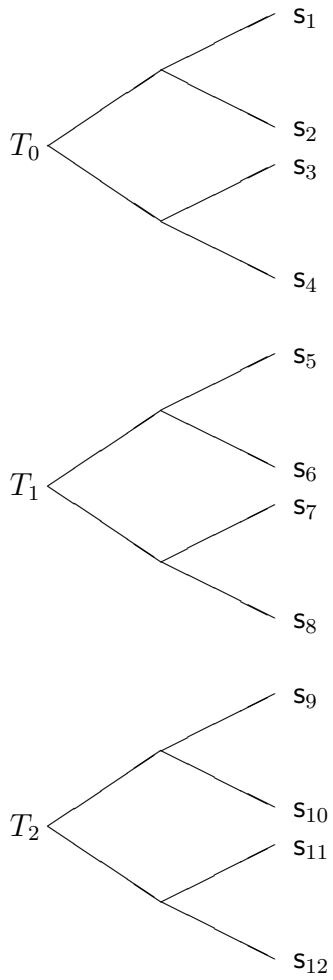

Figure 3.2: Example of the forest of candidate solutions with $L = 3$ and $t = 2$.

Now consider two leaves in some T_i that have a common ancestor node at depth ℓ, and no common ancestor at any depth $\ell' > \ell$ (e.g. s_1 and s_2 of Figure 3.2 have a common ancestor at depth $\ell = 1$, whilst s_1 and s_3 have a common ancestor at depth $\ell = 0$). These two leaves are equal in their first $m\ell$ bits, with the next m bits being generated as two distinct solutions to a system of $m - 1$ linear equations. On the other hand, the last $m(t - \ell - 1)$ bits of two such candidates are generated starting from distinct solutions on

m bits at level $\ell + 1$ of the tree. Informally, the closer together in a tree two leaves are, the more correlated their bits are. We now make the following assumptions about the candidate solution bits at the leaves of these trees:

Weak randomness assumptions: At any stage in the algorithm:

- The bits of all candidate solutions are uniformly distributed over $\{0, 1\}$.
- For some $k \geq 1$, leaves in the same tree (T_i) are independent on the last $m(t-\ell-k)$ bits, provided the leaves have no common ancestor at depth greater than ℓ.
- The bits at leaves in distinct trees are independent of each other across all mt bits.

In the above assumptions, $k \geq 1$ is a parameter which determines the number of fully independent bits in leaves of the tree. From Equations (3.2.2) to (3.2.4) it is clear that the assumption certainly does not hold for $k = 1$ for any RSA moduli; but experimental evidence indicates that $k = 5$ usually suffices to make the bits approximately independent (for smaller k, there are often certain pairs of bit slices that never appear). The question of whether it is possible to prove independence (or some approximation to it) for sufficiently large k is open.

To attempt to bound the success rate of our algorithm under these assumptions, we adapt a standard result on list decoding of random codes, see for example [28, 33]. Let $\mathcal{C}$ denote the code of $L2^t$ candidate solutions on mt bits at a given stage and let r denote the noisy RSA private key bits. In the symmetric case, the log likelihood expression for $\log \mathbb{P}(\mathsf{r}|\mathsf{s}_i)$ simplifies to

$$(n_{00}^i + n_{11}^i) \log(1 - \delta) + (n_{01}^i + n_{10}^i) \log \delta$$

where now $n_{01}^i + n_{10}^i = d_H(\mathsf{r}, \mathsf{s}_i)$ and $n_{00}^i + n_{11}^i = mt - d_H(\mathsf{r}, \mathsf{s}_i)$. Thus maximising likelihood is equivalent to minimising Hamming distance and we may assume that our algorithm outputs the L codewords from $\mathcal{C}$ that are closest to r in the Hamming metric.

We proposed the following theorem in [64], but we have since discovered that it is inapplicable to our algorithm. We will first see the proof, and then we will explain why we cannot use this theorem to bound the success of our algorithm.

Theorem 3.4.1. Let $c > 0$ be a constant, and write $R = 1/m$. Suppose δ is such that

$$R \leq 1 - (1-c)H_2\left(\frac{\delta}{1-c}\right) - \left(c + \frac{1}{L}\right)$$

Then with notation as above, and under our weak randomness assumptions, we have:

$$\mathbb{P}(\mathsf{r}, S \subset \mathcal{C}, |S| = L+1 : d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt \ \forall \mathsf{s} \in S) \leq L^{L+1} 2^{-mt/L} + \frac{L+1}{2^{\lfloor ct \rfloor - k + 1}}.$$

In particular, for fixed $c > 0$, k and L, the above probability tends to 0 as $t \to \infty$.

The theorem merits some interpretation. The standard bound on list decoding for random codes over the symmetric channel [28, 33] states that, provided $R \leq 1 - H_2(\delta) - 1/L$, then a rate R random code of length n is such that the probability

$$\mathbb{P}(\mathsf{r}, S \subset \mathcal{C}, |S| = L+1 : d_H(\mathsf{r}, \mathsf{s}) \leq \delta n \ \forall \mathsf{s} \in S)$$

tends to zero as $n \to \infty$. Here, $1 - H_2(\delta)$ is just the classical Shannon capacity of the channel. One interprets this result as saying that list decoding for a list of length L will be successful provided the rate is not too close to capacity. Our theorem shows that, for a random subset S of size $L+1$, the same is true

for our code $\mathcal{C}$, up to a small defect induced by c. Since $c > 0$ is arbitrary, we obtain, qualitatively, a bound of the same strength in the asymptotic setting, i.e. as $t \to \infty$.

Finally, we note that the argument we use in the proof of Theorem 3.4.1 is specific to the symmetric case.

Proof. (of Theorem 3.4.1) For $S \subset \mathcal{C}$ with $|S| = L + 1$, we say that S is *bad* if any 2 words in S correspond to leaves from the same tree having a common ancestor at depth greater than or equal to $\ell_c := \lfloor ct \rfloor - k$. Otherwise, we say that S is *good*. Note that if S is bad, then some pair of words in S will have their first $m\ell_c$ bits in common, while if S is good, then all pairs of words in S will have (at least) their last $m(t - \ell_c - k)$ bits independent and uniformly random.

Now, over a uniformly random choice of $S' \subset \mathcal{C}$ of size 2, we have

$$\mathbb{P}(S' \text{ bad}) \leq \frac{2^{t-\ell_c} - 1}{L2^t} \leq \frac{1}{L2^{l_c}}$$

since there are $L2^t$ leaves in total and, given a choice of a first leaf, there are $2^{t-\ell_c} - 1$ choices of other leaves below the first leaf's ancestor node at depth ℓ_c. Applying a union bound over pairs of words from S of size $L + 1$, we get:

$$\mathbb{P}(S \text{ bad}) \leq \binom{L+1}{2} \cdot \frac{1}{L2^{\ell_c}} \leq \frac{L+1}{2^{\ell_c+1}}.$$

Notice that, for our choice of ℓ_c, $\mathbb{P}(S \text{ bad})$ becomes arbitrarily small as $t \to \infty$ for fixed L.

Let $E(\mathsf{r}, S)$ denote the event that $d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt$ for all $\mathsf{s} \in S$. Now, to bound the probability of $E(\mathsf{r}, S)$ over random choices of r, S, we condition on S being good or bad:

$$\begin{aligned} \mathbb{P}(E(\mathsf{r}, S)) &= \mathbb{P}(E(\mathsf{r}, S)|S \text{ good}) \cdot \mathbb{P}(S \text{ good}) + \mathbb{P}(E(\mathsf{r}, S)|S \text{ bad}) \cdot \mathbb{P}(S \text{ bad}) \\ &\leq \mathbb{P}(E(\mathsf{r}, S)|S \text{ good}) + \mathbb{P}(S \text{ bad}). \end{aligned}$$

Here, we already have a bound for the second term which tends to 0 for fixed L as $t \to \infty$. We proceed to bound the first term. Now for fixed r and good S, we know that all words in S are uniformly and independently distributed on their last $m(t-\ell_c-k)$ bits. Moreover, r is randomly distributed. We will focus the remainder of the proof on these last $m(t-\ell_c-k)$ bits. If $d_H(\mathsf{r},\mathsf{s}) \leq \delta mt$ then the Hamming distance is certainly less than or equal to δmt for the last $m(t-\ell_c-k)$ bits. Hence, when S is good, for each $\mathsf{s} \in S$, we have:

$$\mathbb{P}(d_H(\mathsf{r},\mathsf{s}) \leq \delta mt) \leq \frac{\mathrm{Vol}(\delta mt, m(t-\ell_c-k))}{2^{m(t-\ell_c-k)}}$$

where $\mathrm{Vol}(a,b)$ denotes the volume of a Hamming ball of radius a in b-dimensional Hamming space. Approximating ℓ_c by $ct-k$, and using the standard volume bound

$$\mathrm{Vol}(\delta b, b) \leq 2^{bH_2(\delta)},$$

we obtain, after some simplification:

$$\mathbb{P}(d_H(\mathsf{r},\mathsf{s}) \leq \delta mt) \leq \frac{2^{(mt(1-c))H_2(\delta/(1-c))}}{2^{mt(1-c)}}.$$

Now we use a union bound, noting that there are 2^{mt} choices for r, at most $(L2^t)^{L+1}$ choices for S of size $L+1$, and everything is independent on the last $m(t-\ell_c-k)$ bits, to obtain:

$$\begin{aligned}\mathbb{P}(E(\mathsf{r},S)|S \text{ good}) &\leq 2^{mt} \cdot (L2^t)^{L+1} \cdot \left(\frac{2^{(mt(1-c))H_2(\delta/(1-c))}}{2^{mt(1-c)}}\right)^{L+1} \\ &= L^{L+1}(2^{mt})^{1+(1/m)(L+1)+(1-c)(L+1)H_2(\delta/(1-c))-(L+1)(1-c)}.\end{aligned}$$

Now we select parameters so that the exponent of 2^{mt} here simplifies. This exponent is

$$1 + \frac{1}{m}(L+1) + (1-c)(L+1)H_2\left(\frac{\delta}{1-c}\right) - (L+1)(1-c) \quad (3.4.1)$$

$$= 1 + (L+1)\left[\frac{1}{m} + (1-c)H_2\left(\frac{\delta}{1-c}\right) - (1-c)\right]. \quad (3.4.2)$$

Suppose we set δ such that

$$\frac{1}{m} = 1 - (1-c)H_2\left(\frac{\delta}{1-c}\right) - \left(c + \frac{1}{L}\right),$$

(such a value δ will always exist provided c is sufficiently small and $m = 2, 3$ or 5). After some manipulation, we find that for this carefully chosen value of δ, the exponent of 2^{mt} in (3.4.2) simplifies to $-1/L$, and hence, for this value of δ, we have:

$$\mathbb{P}(E(\mathsf{r}, S)|S \text{ good}) \leq L^{L+1}(2^{mt})^{-1/L}.$$

Moreover, for sufficiently small c, the above bound also holds for every δ such that

$$\frac{1}{m} \leq 1 - (1-c)H_2\left(\frac{\delta}{1-c}\right) - (c + \frac{1}{L}). \tag{3.4.3}$$

This follows easily from noting that $1-(1-c)H_2\left(\frac{\delta}{1-c}\right)$ is a decreasing function of δ for $0 \leq \delta \leq (1-c)/2$.

Putting everything together, we have that, provided δ satisfies (3.4.3),

$$\mathbb{P}(E(\mathsf{r}, S)) \leq L^{L+1}2^{-mt/L} + \frac{L+1}{2^{\lfloor ct \rfloor - k + 1}}$$

which concludes the proof. □

We originally believed that a sufficient condition for our algorithm to be successful on average is that

$$\mathbb{P}(\mathsf{r}, S \subset \mathcal{C}, |S| = L+1 : d_H(\mathsf{r}, \mathsf{s}) \leq \delta mt \ \forall \mathsf{s} \in S)$$

should be sufficiently small over random choices of r and subset S of $\mathcal{C}$ with $|S| = L+1$. For, if this is the case, then, given that r will have on average δmt bits in error compared to the correct codeword s_c, this bound on probability dictates that there is a small chance that L wrong codewords will be close enough to r that they are all preferred over s_c in the list output by the algorithm. Hence the algorithm is highly likely to output the correct s_c in its list when this probability is small. However, we have since realised that this line of reasoning is incorrect. Standard list-decoding results usually assume the existence of a random code, and in such scenarios *every* set of solutions has

an equal distribution. Hence, to bound the probability of a set of size $L+1$ being in a particular Hamming ball, it is sufficient to bound the probability of a random set of size $L+1$ being in this Hamming ball, and then use a union bound over all possible sets of size $L+1$. Unfortunately, in our scenario the code is not random, and hence such an approach does not bound the success of our algorithm. In particular, whilst the probability may be small over a random choice of set S, our algorithm considers *all* possible sets S of size $L+1$, which is where the reasoning breaks down. Consequently, we are unable to provide a rigorous success analysis of our algorithm even in the case of symmetric errors. In addition, based on our analysis of the literature, the state-of-the-art appears to be that no random coding bounds are known for list decoding for asymmetric channels. However, we note that the motivation for this chapter is the study of practical cold boot attacks, and we will shortly see that our algorithm is very successful in practice.

3.5 Experimental Results

For our experiments using our algorithm, we used a multi-threaded 'C' implementation and Shoup's NTL library for large integer arithmetic. The stack-based, depth-first approach to generating trees of candidates greatly reduced the amount of memory consumed by the algorithm. Because of certain thread-safety issues in NTL, we had to introduce some artificial delays into our code, making all of our running times higher than they could otherwise have been. We ran our code on an 8x virtual CPU hosted on a 2x Intel Xeon X5650, clocked at 2.67 GHz (IBM BladeCenter HS22V). Except where noted below, our experiments were run for 100 trials using a randomly-generated RSA key for each trial. Also except where noted, our results refer to private keys of the form $\mathsf{sk} = (p, q, d, d_p, d_q)$ and are all for 1024-bit RSA moduli.

δ	0.08	0.10	0.12	0.14	0.16	0.18	0.2	0.22
t	6	8	10	12	16	18	18	18
L	4	4	8	32	32	32	32	64
Suc.	0.982	0.991	0.985	0.978	0.82	0.61	0.19	0.03
T (ms)	95	172	211	1852	35062	161512	160325	325193

Table 3.3: Success probabilities for the symmetric case ($(\alpha, \beta) = (\delta, \delta)$). Experiments with $\delta \leq 0.14$ are based on 500 trials. Capacity bound on δ is 0.243.

3.5.1 The symmetric and cold boot channels

We have conducted extensive experiments using our algorithm for the symmetric case considered in [42]. Our results are shown in Table 3.3. For small values of δ, we achieve a success rate of 1 or very close to 1 using only moderate amounts of computation. By contrast the HMM algorithm does not achieve such high success rate for small δ. This cannot be solved by increasing t in the HMM algorithm because this leads to a blow-up in running time. For larger δ, the success rate of our algorithm is comparable to that of [42] for similar values of t. We were able to obtain a non-zero success rate for $\delta = 0.22$, while [42] only reached $\delta = 0.20$. The bound from capacity is 0.243.

For the idealised cold boot setting where $\alpha = 0$, our experimental results are shown in Table 3.4. Recall that the HS algorithm can also be applied to this case. Translating the fraction of known bits $(1 - \rho)$ to the idealised cold boot setting, and assuming the HS algorithm works just as well when only 1 bits are known (instead of a mixture of 0 and 1 bits), the maximum value of β that could be handled by the HS algorithm theoretically would be 0.46 (though results reported in [44] would allow β as high as 0.52). Our algorithm still has a reasonable success rate for β as high as 0.6 and non-zero success rate even for $\beta = 0.63$, beating the HS algorithm by some margin. Our capacity

ρ	0.2	0.3	0.4	0.5	0.55	0.6	0.62	0.63
t	6	8	12	18	18	18	18	18
L	4	8	8	16	16	16	64	64
Suc.	0.99	0.95	0.94	0.80	0.40	0.14	0.05	0.01
T (ms)	90	129	1135	154562	155328	156515	288531	282957

Table 3.4: Success probabilities for the idealised cold boot case ($\alpha = 0$). Capacity bound on β is 0.666.

β	0.1	0.2	0.3	0.4	0.5	0.55	0.6	0.61
t	6	6	8	12	16	18	18	18
L	4	4	8	8	16	32	64	64
Suc.	1	0.99	0.99	0.98	0.61	0.33	0.11	0.04
T (ms)	71	73	281	2385	22316	171356	333614	339198

Table 3.5: Success probabilities for the true cold boot case with $\alpha = 0.001$. Capacity bound on β is 0.658.

analysis for this case suggests that the maximum value of β will be 0.666. Thus our algorithm is operating within 5% of capacity here.

We present experimental results for the true cold boot setting in Table 3.5. Given $\alpha = 0.001$, it follows from our asymptotic analysis that the theoretical maximum value of β which can be handled by our algorithms is 0.658. Our

β	0.1	0.15	0.20	0.25	0.30	0.35	0.40	0.43
t	6	10	14	16	18	18	18	18
L	4	16	16	16	16	16	32	64
Suc.	1	0.99	0.97	0.95	0.60	0.58	0.11	0.05
T (ms)	62	402	3056	12135	62537	62139	121358	120985

Table 3.6: Success probabilities for the true cold boot case with $\alpha = 0.001$ and $\mathsf{sk} = (p, q, d)$. Capacity bound on β is 0.479.

β	0.05	0.1	0.15	0.20	0.26
t	10	12	16	18	18
L	8	8	16	32	64
Suc.	0.96	0.80	0.65	0.31	0.08
T (ms)	318	629	6451	61213	131600

Table 3.7: Success probabilities for the true cold boot case with $\alpha = 0.001$ and $\mathsf{sk} = (p, q)$. Capacity bound on β is 0.298.

algorithm still has a non-zero success rate for β as high as 0.61. We reiterate that this true cold boot setting is not handled by any of the algorithms previously reported in the literature.

Furthermore, for private keys of the form $\mathsf{sk} = (p, q, d)$ and $\mathsf{sk} = (p, q)$, our algorithm performs very well in the true cold boot setting. For $\mathsf{sk} = (p, q, d)$, the maximum value of β suggested by our capacity analysis is 0.479. With $\beta = 0.4$, $t = 20$ and $L = 16$ our success rate is 0.11 and we have non-zero success rate even with $\beta = 0.43$. Similarly, when $\mathsf{sk} = (p, q)$ our capacity analysis shows that the maximum β is 0.298. When $\beta = 0.2$, $t = 18$ and $L = 16$ we still have a success rate of 0.31, but we can even recover keys with non-zero success rate for β as high as 0.26. Tables 3.6 and 3.7 show our results for these cases.

3.6 Conclusions

In this chapter we have introduced a coding-theoretic viewpoint to the problem of recovering an RSA private key from a noisy version of the key. This provides new insights on the HS [44] and HMM [42] algorithms and leads to a new algorithm which is efficiently implementable and enjoys good performance at high error rates. In particular, it is the first algorithm that works for the

true cold boot case, where both $\mathbb{P}(0 \to 1)$ and $\mathbb{P}(1 \to 0)$ are non-zero. Open problems include developing a rigorous asymptotic analysis of our algorithm. Another open problem is to provide a threshold-based approach (à la [42]) for the case of non-symmetric errors ($\mathbb{P}(0 \to 1) \neq \mathbb{P}(1 \to 0)$). This is a problem we will address in the next chapter.

Chapter 4

Cold Boot Attacks in the Discrete Logarithm Setting

In this chapter we will discuss cold boot attacks in the discrete logarithm setting. For an introduction to the nature of cold boot attacks, please refer to Section 3.1. We begin by discussing prior work in this area (the LKBC algorithm [55]), then we proceed to discuss a statistical test that we will use within our key-recovery algorithms. Once armed with this statistical test we will study certain elliptic curve discrete logarithm implementations, and we will study how these implementations may be vulnerable to cold boot attacks. Finally, we provide details of some experimental results to support the theoretical analysis of our algorithms.

4.1 The LKBC Algorithm and Its Limitations

Published cold boot analyses almost ubiquitously assume that attackers can obtain a (noisy) copy of a private key that has some form of redundancy. For instance, in the previous chapter we discussed how the PKCS#1 standard advises storing the additional values p, q, d_p, d_q, and q_p^{-1} as part of the RSA private key to increase the efficiency of decryption and signing operations. It

is this redundancy that was exploited by previous authors to recover private keys even when they were subjected to very high noise levels (since more redundancy will allow the attacker to deal with higher noise levels). As far as we are aware, there has only been one paper (by Lee et al. [55]) that discussed the possibility of key recovery in the discrete logarithm setting. Their attack model assumes that an attacker only has access to the public key g^x and a decayed version of the private key x. Consequently, given that there is no further redundancy, their proposed algorithm would be unable to efficiently recover keys that were affected by particularly high noise levels. There are several more drawbacks to the analysis of [55], which we will discuss after giving a brief summary of the LKBC algorithm.

At the heart of the LKBC is a technique courtesy of Heiman [41]. Suppose we have an (unknown) n-bit private key denoted x, and the public key is $y = g^x$. Furthermore, suppose we know the Hamming weight of x is t (i.e. the number of 1 bits in the binary representation of x is t). Heiman proposed splitting the exponent as $x = x_1 + x_2$, where x_1 has Hamming weight t_1, and x_2 has Hamming weight t_2 (with $t = t_1 + t_2$). Heiman then proposed cycling through pairs of possible x_1 and x_2, and computing g^{x_1} and yg^{-x_2} for each pair. When a pair (x_1, x_2) is found such that $g^{x_1} = yg^{-x_2}$, then we must have $x = x_1 + x_2$, which reveals the private key. The LKBC makes use of this algorithm (with several slight modifications that are irrelevant for our discussion). The authors of the LKBC algorithm assume that an adversary obtains the noisy private key (denoted x') and the adversary knows an upper bound for the number of bits that have flipped, say k. For each value of $k' \leq k$, the LKBC algorithm uses the modified Heiman algorithm and cycles through possible values x_1 and x_2 such that the Hamming distance between $x_1 + x_2$ and x' is k'. When a pair (x_1, x_2) is found such that $g^{x_1} = yg^{-x_2}$, the algorithm outputs $x_1 + x_2$. When pairs x_1 and x_2 have been exhausted,

the LKBC algorithm increments k' by one. If the algorithm does not output anything for $k' \leq k$, then the algorithm fails. From this brief discussion, it is clear to see that the LKBC algorithm is not very sophisticated and various improvements should be possible.

A particular drawback of the LKBC algorithm is the assumption that an adversary knows an upper bound for the number of errors in the private key. Such an assumption is far too strong. If δ is the probability that an individual bit flips, then an adversary may compute an upper bound for the noise rate $\delta_{\max}$ (the computation of the upper bound was considered in [42][1]), but this is not the same as computing an upper bound for the number of bits that have flipped. The only upper bound that can be known is the trivial upper bound: the number of bits in the key. Lee et al. used the naïve method of assuming that the upper bound on the number of flipped bits is $\lfloor n \cdot \delta_{\max} \rfloor$, where n is the number of bits in the private key x. Whilst this may suffice in the majority of cases, there will clearly be occasions when the total number of errors exceeds this value. If this happens, the LKBC algorithm will fail, but there is no probability analysis discussing how often this happens. Furthermore, the algorithm of LKBC does not explicitly consider the true cold boot scenario. In a one-to-zero region, the probability of a $1 \rightarrow 0$ flip is much higher than a $0 \rightarrow 1$ flip. The LKBC algorithm implicitly works in this scenario since they assume an upper bound on the noise rate. However, if the probabilities of bit flips are different for zeros and ones, then perhaps the algorithm could be enhanced by employing an algorithm that takes this discrepancy into account. LKBC do attempt to model the cold boot scenario by providing an analysis in the case of unidirectional errors. That is, in a $1 \rightarrow 0$ region, they assume that 0 bits do not flip. Unfortunately, this is not a true reflection of what occurs

[1]This computation was specifically for RSA keys, but a similar result will hold for any algorithm in which an adversary has access to the noisy and original versions of the public key.

in a cold boot attack. In a $1 \to 0$ region, a 0 bit will flip with an approximate probability of 0.001. If just a single 0 bit flips to a 1, the unidirectional LKBC algorithm will fail. If a 160-bit discrete logarithm key is chosen uniformly and zeros degrade with probability 0.001, then at least one 0 will flip (and the LKBC algorithm will instantly fail) approximately 8% of the time. Furthermore, for larger key sizes this probability of failure obviously increases, so the LKBC analysis certainly does not naturally extend to these cases.

4.2 Our Contributions

Given the above discussion on the results of [55], it is natural to ask whether, in practical discrete logarithm-based software implementations, there are any private key representations that contain redundancy that can be used to improve cold boot key-recovery algorithms. It turns out that such cases are common, and they will form the basis of this chapter. The scenarios we consider are taken from two wide-spread ECC implementations found in TLS libraries: the windowed non-adjacent form (wNAF) representation used in OpenSSL, and the PolarSSL comb-based approach. By exploiting redundancies in the respective in-memory representations of private keys we are able to improve upon the results from [55] by providing an algorithm that is applicable to the true cold boot scenario; something that cannot be said for the LKBC algorithm.

Our techniques are based on a novel statistical test that allows a trade-off between success rate and execution speed. We stress that this test is not only applicable to the discrete logarithm setting, but is applicable to all types of key. In particular, it complements the framework of the previous chapter for the RSA setting.

Recall that in the previous chapter we proposed using an ML approach

(see Algorithm 1), which has a bounded running-time, but no lower bound on the success rate of the algorithm is provided. In contrast, for our discrete logarithm algorithm we succeed in lower-bounding the success rate. Furthermore, the statistical test used within our algorithms may also be applied to the RSA setting to obtain a threshold-based approach (à la [42]), which comes with a theoretical lower bound on success probability. Although we provide no bound on the running-time of our primary algorithm, we note that various modifications allow an attacker to seek his own compromise between a preferred success rate and a desired running-time.

4.3 Multinomial Distributions and the Multinomial Test

The general strategy behind key-recovery procedures for cold boot attacks is to only consider small parts of the targeted key at a time. For instance, RSA-based reconstruction procedures usually start with the least significant bits (LSBs) such as we did in the previous chapter, along with the following works: [42, 44, 51, 52, 55, 73]. However, it is also possible to begin with the most significant bits (MSB) [72]. It is typical to use an iterative process to guess a couple of bits of the key and assess the plausibility of the guess on the basis of both a model of the decay process and the available redundancy in the encoding. Previous cold boot papers have proposed various methods by which the plausibility of the guess is ascertained. In Chapter 3 we used a maximum-likelihood estimate when deciding whether to discard a candidate solution, and in [42] a threshold-based approach was used. The theoretical success rate of the algorithm is usually based on assumptions that are typically only true for a specific type of key being considered (i.e. RSA or AES), and are possibly not easy to generalise. In this section we propose a general statistical test that can

be used in various scenarios. The test is based on multinomial distributions and works well for scenarios where the distribution of private key bits is known (such as RSA, where it is generally assumed key bits are uniformly random), but can also be modified to work even when the attacker knows nothing about the distribution of the private key.

We will now study the multinomial distribution and its associated test. Multinomial distributions are a generalisation of the binomial distribution. The distribution has k mutually exclusive events with probabilities $p = (p_1, \ldots, p_k)$, where $\sum_{i=1}^{k} p_i = 1$ and for all i we have $p_i \neq 0$. If there are n trials, we let the random variables $X_1, \ldots, X_k$ denote the number of times the ith event occurs and say that $X = (X_1, \ldots, X_k)$ follows a multinomial distribution with parameters n and p. Given a set of observed values, $x = (x_1, \ldots, x_k)$, we can use a multinomial test to see if these values are consistent with the probability vector p (which is the null hypothesis, denoted H_0). The alternative hypothesis (denoted H_1) for the probability vector is $\pi = (x_1/n, \ldots, x_k/n)$, where each component is the maximum-likelihood estimate for each probability. The two hypotheses can be compared via the calculation $-2\sum_{i=1}^{k} x_i \ln(p_i/\pi_i)$, which is called the *multinomial test statistic*. When the null hypothesis is true, the distribution of this statistic converges to the chi-squared distribution with $k-1$ degrees of freedom as $n \to \infty$ [54].

We will now see how the multinomial test statistic may be applied in cold boot key recovery algorithms. Let s_i denote a (partial) candidate solution for the private key (including the redundant representation) across a section of bits. When comparing a partial candidate solution s_i to the noisy information r we define n_{01}^i to be the number of positions at which there is a 0 in the candidate solution and a 1 in the corresponding position in the noisy information r. We define n_{00}^i, n_{10}^i, and n_{11}^i correspondingly, so $n = n_{00} + n_{01} + n_{11} + n_{10}$. Crucially, this count only considers the newly-guessed bits generated at the

relevant phase of the algorithm, while all previous bits are ignored. It is clear that these counts follow a multinomial distribution. Let $\alpha := \mathbb{P}(0 \to 1)$ denote the probability that a 0 bit flips to a 1 in the execution of the cold boot attack, and let $\beta := \mathbb{P}(1 \to 0)$ denote the probability that a 1 flips to a 0. For the correct candidate solution, s_c, the we know that $(n^c_{00}, n^c_{01}, n^c_{11}, n^c_{10})$ follows a multinomial distribution with $p = (p_0(1-\alpha), p_0\alpha, p_1(1-\beta), p_1\beta)$, where p_b, $b \in \{0, 1\}$, is the probability of a b-bit appearing in the correct candidate solution. We therefore set this to be the null hypothesis (denoted H_0). Notice that we require $\alpha, \beta \neq 0$ since each component of the probability vector must be non-zero. The test may be modified to cover the case when α or β is zero, but we defer this discussion to the Section 4.7. For each candidate solution we could use the previous set of probabilities as the null hypothesis of the multinomial test. We would like to test whether our guessed candidate solution is consistent with this probability vector. The alternative hypothesis is that the set of probabilities for the four bit-pairs is equal to the maximum-likelihood estimates for each category. That is, $H_1 : p = (n^i_{00}/n, n^i_{01}/n, n^i_{11}/n, n^i_{10}/n)$ for each candidate i. We define our first statistical test, which we call Correlate$'$, to be

$$\begin{aligned} \text{Correlate}'(s_i, r) \;:=\; & -2n^i_{00} \ln\left(\frac{np_0(1-\alpha)}{n^i_{00}}\right) - 2n^i_{01} \ln\left(\frac{np_0\alpha}{n^i_{01}}\right) \\ & -2n^i_{11} \ln\left(\frac{np_1(1-\beta)}{n^i_{11}}\right) - 2n^i_{10} \ln\left(\frac{np_1\beta}{n^i_{10}}\right), \end{aligned} \quad (4.3.1)$$

where the values in brackets are the null hypothesis values divided by the alternative hypothesis values, and we define $n \log(1/n) = 0$ if $n = 0$. Correlate$'$ outputs a numerical value (≥ 0) for each candidate. We now need to discuss when we consider this test to pass or fail. It is well known that when the null hypothesis is correct the distribution of the right-hand side of Equation (4.3.1) converges to a chi-squared distribution with $k-1$ degrees of freedom as $n \to \infty$. In our analysis we have $k = 4$, hence the test statistic converges to a chi-squared distribution with three degrees of freedom. We can therefore set a

threshold C such that any candidate whose test statistic is less than C is retained, otherwise the candidate is discarded. We therefore define

$$\text{Correlate}_C(s_i, r) = \text{pass} \quad \Leftrightarrow \quad \text{Correlate}'(s_i, r) < C \ ,$$

where C would be an additional (user-chosen) input to the algorithm. The chi-squared distribution can tell us how to set the threshold C to achieve any desired success rate. If we set the threshold C such that $\int_0^C \chi_3^2(x)\mathrm{d}x = \gamma$ (where $\chi_3^2(x)$ is the probability of the chi-squared distribution equalling x when there are three degrees of freedom), we know that, asymptotically, the probability that the correct candidate's correlation value $\text{Correlate}'(s_i, r)$ is less than C is equal to γ. Recall that the Correlate$'$ test only considers the newly generated bits at each stage of the algorithm, and all previous bits are ignored. This eases the success rate analysis of the algorithm since the probability of passing each Correlate test is independent in this case. Therefore, if the private key has been parsed into m distinct parts, and the attacker applies a Correlate test to each of the m parts, the probability that the correct private key is recovered is γ^m, assuming the same threshold C was used for each Correlate test.

When analysing the success, the only issue yet to be addressed is specifying the values that p_0 and p_1 should take. If the distribution of the private key is known, then it is easy to assign values to these parameters. For example, in the RSA setting of the previous chapter, we assumed that the entire private key would have approximately an equal number of zeros and ones. Therefore, if we were to use the Correlate$'$ test (Equation (4.3.1)) in the RSA setting we would set $p_0 = p_1 = 1/2$. Notice that this immediately gives us a threshold-based approach for recovering noisy RSA private keys that have been degraded according to an asymmetric binary channel (i.e. $\alpha \neq \beta$). Such an approach is currently lacking in the literature. Specifically, Algorithm 1 would take an extra input parameter C, and we would replace the last two lines by the

following:

- Compute Correlate$'$ for each candidate s_i.
- Keep all s_i having Correlate$'$ value less than C and delete all other s_i.

In other settings (such as the key types we consider in this chapter) it may not be possible to accurately assign values to these parameters. The approach we use to overcome this issue is to conduct two separate multinomial tests: one for the 0 bits and another for the 1 bits. The advantage of using two separate tests is that we do not need to estimate the values of p_0 and p_1, and hence our algorithm's success rate will not be harmed by a poor estimation of these parameters. For the correct solution, each 0 can flip to a 1 with probability α or it can remain a 0 with probability $1-\alpha$. Hence, if there are n_0 zeros in the correct solution, then (n_{01}, n_{00}) follows a multinomial distribution with parameters n_0 and $p = (\alpha, 1-\alpha)$. Similarly, if there are n_1 ones in the correct solution, then (n_{10}, n_{11}) follows a multinomial distribution with parameters n_1 and $p = (\beta, 1-\beta)$. We may now use the multinomial test to examine each candidate solution without having to estimate p_0 and p_1. Specifically, we define

$$\text{Correlate}^0(s_i, r) := -2n_{00}^i \ln\left(\frac{n_0(1-\alpha)}{n_{00}^i}\right) - 2n_{01}^i \ln\left(\frac{n_0\alpha}{n_{01}^i}\right) \tag{4.3.2}$$

and

$$\text{Correlate}^1(s_i, r) := -2n_{11}^i \ln\left(\frac{n_1(1-\beta)}{n_{11}^i}\right) - 2n_{10}^i \ln\left(\frac{n_1\beta}{n_{10}^i}\right). \tag{4.3.3}$$

Then we define

$$\text{Correlate}_C(s_i, r) := \text{pass} \quad \Leftrightarrow \quad \text{Correlate}^0(s_i, r) < C \wedge \text{Correlate}^1(s_i, r) < C. \tag{4.3.4}$$

Notice that Correlate^0 and Correlate^1 are functions with one degree of freedom. Therefore the probability that Correlate$^b < C$ is $\gamma = \int_0^C \chi_1^2(x)\mathrm{d}x$, for

$b \in \{0, 1\}$, where $\chi^2_1(x)$ is the probability density function of the chi-squared variable with only one degree of freedom.

To complete the picture of this statistical test, we would like to bound the probability of a Type II error (i.e. not rejecting the null hypothesis when it is false). This would allow us to bound the running-time of an algorithm that implements this statistical test. Such an analysis is currently lacking from the literature, and appears difficult to obtain. We will discuss this issue in more detail in Section 4.8.

4.3.1 Convergence of the multinomial test

One issue with the approach proposed in the previous section is that the Correlate function will only consider a small number of bits at a time, but the convergence of the test to the chi-squared distribution is an asymptotic result. Hence, given the small sample sizes, there will be some discrepancy between what we observe in practice and what is predicted by the chi-squared statistic. There are various modifications that can be made to the multinomial test in order to force a better agreement with the chi-squared test [75, 80], which we will briefly discuss now.

The first modification was due to Williams [80]. Williams' idea was to multiply the threshold C by a factor q, where

$$q = 1 + \frac{-1 + \sum_{i=1}^{k} p_i^{-1}}{6n(k-1)}.$$

An alternative result by Smith, Rae, Manderscheid and Silbergeld [75] suggests that the correcting-factor q should be

$$q = 1 + \frac{1}{6n(k-1)} \cdot \left(-1 + \left(\sum_{i=1}^{k} p_i^{-1} \right) + n^{-1} \sum_{i=1}^{k} (p_i^{-1} - p_i^{-2}) \right).$$

The Williams method always produces a factor $q > 1$. However, the method of Smith et al. will sometimes produce values of q that are less than zero. This

results in a negative threshold C for the multinomial test, which will mean a success rate of zero (because the multinomial test outputs values greater than or equal to zero, so no candidate will have a value less than the threshold). A closer inspection of the result of Smith et al. reveals why the threshold will sometimes be negative. The range of the chi-squared distribution is $[0, \infty)$ whereas the range of the multinomial test statistic is $[0, -2n \ln p_{\min}]$, where $p_{\min}$ is the minimum value in the probability vector of the null hypothesis. At first sight, it may appear that the result of [75] is intended to decrease the discrepancy between the theoretical and practical results for small values of n. However, the intention of their work is to reduce the discrepancy for *finite* values of n. That is, their result attempts to address the difference between the finite range of the multinomial test statistic and the infinite range of the chi-squared distribution. Consequently, their result is still an asymptotic result. They show that their modified test statistic converges to the chi-squared distribution more quickly than the standard multinomial test statistic. However, since their result is asymptotic in nature, it will produce some impractical (i.e. negative) thresholds for small values of n, and is therefore inappropriate for our algorithms.

4.4 Exponentiation Algorithms

We now turn to a discussion of the discrete logarithm setting. Specifically, we will consider textbook methods for point multiplication in the discrete logarithm setting before proceeding to the specialised algorithms of OpenSSL and PolarSSL that will form the basis of this chapter.

A core part of any DLP-based cryptosystem realised in the elliptic curve setting is a point multiplication routine. Here, a curve point P, also referred to as a base point, is multiplied with a scalar $a \in \mathbb{N}$ to obtain another curve

point $Q = aP$. The overall performance of this operation depends on various factors, including the representation of field elements (e.g. 'pseudo-Mersenne' vs. 'Montgomery-friendly' moduli for prime fields), the availability of optimised formulas for basic group operations like point addition and doubling (e.g. 'Weierstrass' vs. 'Edwards' curves), the representation of curve points (e.g. 'affine' vs. 'projective'), and the scheduler that specifies how the basic group operations are combined to achieve a full point multiplication algorithm (see [18] for a recent survey on available options and tradeoffs in all these categories). In the context of cold boot attacks particularly, the scheduler seems to be an interesting target to analyse: in ECC-based cryptosystems, secret keys typically correspond with scalars, i.e. with precisely the information with which the scheduler works. In the following we give a brief overview of the most relevant of such algorithms [38]. We analyse their resilience against cold book attacks in later sections of this chapter.

4.4.1 (Windowed) double-and-add

The textbook method for performing point multiplication is the *double-and-add* algorithm.[2] Given scalar $a \in \mathbb{N}$ and an appropriate length parameter $\ell \in \mathbb{N}$, it requires that a is represented by its *binary expansion* $[a]_1 = (a_\ell, \ldots, a_0)$, where $a = \sum_{i=0}^{\ell} a_i 2^i$ and $a_\ell, \ldots, a_0 \in \{0, 1\}$. Given $[a]_1$, and denoting right-shifting a by k positions with $a \gg k$, we observe

$$aP = \left(\sum_{i=0}^{\ell} a_i 2^i \right) P = \left(\sum_{i=0}^{\ell-1} a_{i+1} 2^{i+1} \right) P + a_0 P = 2(a \gg 1)P + a_0 P.$$

This recursion can be unrolled to

$$aP = 2(2(2(\ldots + a_3 P) + a_2 P) + a_1 P) + a_0 P. \tag{4.4.1}$$

[2] This is known as the square-and-multiply algorithm if the group is written multiplicatively.

The double-and-add algorithm for computing $Q = aP$ is now immediate: it initialises Q with $\mathcal{O}$ and iteratively updates $Q \leftarrow 2Q + a_i P$, where the a_i are considered 'left-to-right' (i.e. i counts backwards from ℓ down to 0). The whole procedure takes approximately $\ell/2$ additions and ℓ doublings per point multiplication, if uniform exponents are assumed.

A common approach to improve the efficiency of this algorithm is to decrease the number of required additions by applying a window technique. More precisely, for fixed *window size* w, we define the notion of *windowed binary expansion* as above, this time relaxing the requirement on the a_i to $a_\ell, \ldots, a_0 \in [0 \,..\, 2^w - 1]$ and using notation $[a]_w = (a_\ell, \ldots, a_0)$. While such an encoding is in general not unique, it can be shown to uniquely exist [38] if one additionally requires either that $a_i = 0$ for all $i \not\equiv 0 \pmod{w}$ (*fixed window*), or that all non-zero a_i be odd and that all w-length subsequences of $[a]_w$ contain at most one such element (*sliding window*).

Observe that, if we assume the fixed-window case and that points $P, 2P, \ldots, (2^w - 1)P$ are precomputed, then $w - 1$ out of w additions vanish from Equation (4.4.1). Even more additions potentially vanish in the sliding-window case; moreover, as here all non-zero coefficients a_i are odd, fewer precomputed points have to be tabulated.

Example 1. For $a = 30$ and $\ell = 6$ we have $[a]_1 = (0, 0, 1, 1, 1, 1, 0)$. Windowed binary expansions for $w = 2$ are $(0, 0, 1, 0, 3, 0, 2)$ (fixed window) and $(0, 0, 0, 3, 0, 3, 0)$ (sliding window).

4.4.2 (Windowed) signed-digit representations

Many different ways to represent elliptic curve points have been proposed [38, 18]; a common property of all these encodings is that group negation is a cheap operation. For instance, for curves in Weierstrass form that are defined over

prime fields, e.g. the five 'prime curves' standardised by NIST, the negative of a point (x, y) is simply $(x, -y)$. A general consequence of this is that point subtraction performs as efficiently as point addition. This is exploited in point multiplication algorithms that are based on the *signed-digit representation* of scalars.

Formally, for fixed window size w, we denote by $[a]_{\pm w} = (a_\ell, \ldots, a_0)$ any decomposition of $a \in \mathbb{N}$ such that $a = \sum_{i=0}^{\ell} a_i 2^i$ and $a_i \in [-2^{w-1} .. 2^{w-1} - 1]$. As Equation (4.4.1) still holds if some of the coefficients a_i are negative, a 'double-and-add-or-subtract' algorithm that operates on such signed-digit representations is readily derived. The key idea is that the extra freedom obtained by allowing coefficients to be negative will make it possible to find particularly sparse scalar representations, i.e. representations for which only a minimum number of group additions/subtractions is required.

We describe three common signed-digit normal forms for representing scalars $a \in \mathbb{N}$. The first one, *non-adjacent form* (NAF), limits the digit set to $\{0, \pm 1\}$ and requires that no two consecutive coefficients be non-zero. The second and third are defined with respect to a window size w. Specifically, while the *fixed-window NAF* is an encoding of the form $[a]_{\pm w}$ that requires $a_i = 0$ for all $i \not\equiv 0 \pmod{w}$, the *sliding-window NAF* (wNAF) ensures that all non-zero a_i are odd and all w-length subsequences of $[a]_{\pm w}$ contain at most one such element. All three types of encoding are unique. Note that in the $w = 1$ case the notions of NAF and wNAF coincide. Observe also that storing a NAF or wNAF might require one extra digit over the plain binary expansion. For an example of the latter, consider that the binary expansion of the decimal number 15 is the sequence $(1, 1, 1, 1)$, while its NAF is $(1, 0, 0, 0, \bar{1})$, where we write $\bar{1}$ for -1.

Example 2. The NAF of $a = 30$ is $(0, 1, 0, 0, 0, \bar{1}, 0)$. For window size $w = 2$ the fixed-window NAF is $(1, 0, \bar{2}, 0, 0, 0, \bar{2})$ and the wNAF is $(0, 1, 0, 0, 0, \bar{1}, 0)$.

Algorithm 2 Textbook wNAF encoding. Operator 'smod' computes signed remainders of integer divisions by powers of two. Precisely, for integers a, b we have $b = a \operatorname{smod} 2^w$ iff $\exists k : a = k2^w + b \ \wedge \ b \in [-2^{w-1} .. 2^{w-1} - 1]$.

Input: scalar a, length parameter ℓ, window size w
Output: wNAF $(b_\ell, \ldots, b_0)$

```
1: for i ← 0 to ℓ do
2:     if a is odd then
3:         b_i ← a smod 2^w
4:     else
5:         b_i ← 0
6:     a ← (a − b_i) ≫ 1
7: return (b_ℓ, ..., b_0)
```

Algorithm 2 gives instructions on how to derive the wNAF of a scalar $a \in \mathbb{N}$. Observe that the computation is conducted in a greedy right-to-left fashion, with a $(w-1)$-look-ahead. As the latter property will become relevant in our later analyses, we state it formally.

Fact 1 (Suffix property of wNAF). Fix a scalar $a \in \mathbb{N}$ and a window size w. Denote a's binary expansion with $(a_\ell, \ldots, a_0)$ and its wNAF with $(b_\ell, \ldots, b_0)$, for an appropriate length parameter ℓ. Then for all $0 \leq t \leq \ell - w + 1$ it holds that $(b_t, \ldots, b_0)$ is fully determined by $(a_{t+w-1}, \ldots, a_0)$.

4.4.3 Point multiplication in OpenSSL

We give details about the elliptic curve point multiplication routine used in OpenSSL. Specifically, we studied the code from file `crypto/ec/ec_mult.c` of OpenSSL version 1.0.1h from March 2012, which is the latest stable release. Particularly relevant for this work is the function `compute_wNAF` defined in line 193, which computes a so-called *modified wNAF*. In brief, while a regular wNAF requires every w-length subsequence of digits to contain at most one non-zero element, in modified wNAFs [61] this requirement is relaxed for the most significant non-zero position. For instance, in case $w = 2$

Algorithm 3 OpenSSL's wNAF encoding

Input: scalar a, length parameter ℓ, window size w
Output: modified wNAF $(b_\ell, \ldots, b_0)$
1: compute $b = (b_\ell, \ldots, b_0)$ using Algorithm 2
2: **if** b has prefix $0^*10^{w-1}\beta$ with $\beta < 0$ **then**
3: $\quad \bar{\beta} \leftarrow 2^{w-1} + \beta$
4: $\quad$ in b, replace substring $10^{w-1}\beta$ by $010^{w-2}\bar{\beta}$
5: **return** b

this allows to encode decimal number 11 as $(1, 1, 0, \bar{1})$, while the corresponding (strict) wNAF is $(1, 0, \bar{1}, 0, \bar{1})$ and hence involves one more doubling operation. OpenSSL's `compute_wNAF` function computes the modified wNAF following Algorithm 3, with default window size $w = 4$ (see line 816). The resulting coefficients $a_i \in [-2^{w-1} .. 2^{w-1} - 1]$ are encoded into an array of octets (data type '`signed char`'), using a standard two-complement in-memory representation. For instance, we have

$$\begin{array}{rcl} -3 & \mapsto & \texttt{11111101} \\ -1 & \mapsto & \texttt{11111111} \\ 0 & \mapsto & \texttt{00000000} \\ +1 & \mapsto & \texttt{00000001} \\ +3 & \mapsto & \texttt{00000011} \quad . \end{array}$$

We confirmed that the OpenSSL forks LibreSSL[3] (version 2.0.3 from July 2014) and BoringSSL[4] (version from July 2014) use precisely the same exponent encoding as described above.

4.4.4 Comb-based methods

The various methods for point multiplication that we studied in the preceding sections aimed at requiring less point additions than the basic double-and-add technique; the number of doubling operations, however, was left untouched

[3] `http://www.libressl.org/`, see file `crypto/ec/ec_mult.c`
[4] `https://boringssl.googlesource.com/`, see file `crypto/ec/wnaf.c`

Algorithm 4 Textbook comb encoding

Input: scalar a, parameters w, d
Output: coefficients $K^{d-1}, \ldots, K^0$
1: **for** $i \leftarrow 0$ **to** $d-1$ **do**
2: $\quad K^i \leftarrow (a_{i+(w-1)d}, \ldots, a_{i+d}, a_i)$
3: **return** $K^{d-1}, \ldots, K^0$

(or was even increased). In contrast, comb-based methods [57] manage with significantly fewer doublings, at the expense of some pre-computation dependent on the base point. In the following we give a rudimentary introduction to comb-based multiplication techniques. See [38] for further details.

Fix a base point P and parameters $w, d \in \mathbb{N}$. For any scalar $a \in \mathbb{N}$ with $0 \leq a < 2^{wd}$ let $[a]_1 = (a_{wd-1}, \ldots, a_0)$ denote its binary expansion. For all $i \in [0\,..\,d-1]$ let $K^i = (K^i_{w-1}, \ldots, K^i_0)$ where $K^i_j = a_{i+jd}$, as formalised by Algorithm 4 and illustrated in Figure 4.1. That is, since values $K^i_j \in \{0, 1\}$ are assigned such that

$$a = \sum_{i=0}^{wd-1} 2^i a_i = \sum_{i=0}^{d-1} \sum_{j=0}^{w-1} 2^{i+jd} K^i_j = \sum_{i=0}^{d-1} 2^i \sum_{j=0}^{w-1} 2^{jd} K^i_j$$

we have that

$$aP = \sum_{i=0}^{d-1} 2^i T(K^i_{w-1}, \ldots, K^i_0) \qquad \text{where} \qquad T\colon (k_{w-1}, \ldots, k_0) \mapsto \sum_{j=0}^{w-1} 2^{jd} k_j P.$$

The fundamental idea behind comb-based point multiplication is to precompute table T; as we have seen, the remaining part of the computation of aP can then be conducted with not more than d additions and doublings.

As first observed by Hedabou et al. [39], implementations of the described point multiplication method might offer only limited resilience against side-channel attacks based on simple power analysis (SPA). This comes from the fact that any vector $K^i = (K^i_{w-1}, \ldots, K^i_0)$ is equal to $(0, \ldots, 0)$ with probability 2^{-w} and that, in the multiplication process, this condition implies adding

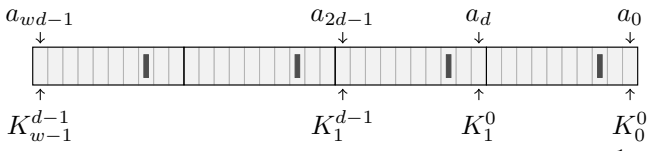

Figure 4.1: Visualisation of the comb method, for parameters $(w, d) = (4, 10)$. The cells represent the bits of the scalar, the bold rectangles mark the prongs of a comb positioned at offset $i = 2$.

neutral element $T(0, \ldots, 0) = \mathcal{O}$ to the current accumulator: an event that is likely to be detectable by analysing power traces.

To mitigate the threat, [39] proposes a comb-based scheduler where the situation $K^i = (0, \ldots, 0)$ does not occur. In a nutshell, it (a) considers only odd scalars (this guarantees $K^0 \neq (0, \ldots, 0)$), (b) introduces for each $i \in [0\,..\,d-1]$ a flag $\sigma^i \in \{\pm 1\}$ that defaults to $\sigma^i = +1$ and indicates whether the corresponding K^i should be considered 'positive' or 'negative', and (c) examines vectors $K^1, \ldots, K^{d-1}$ (in that order) and for each particular K^i that is equal to $(0, \ldots, 0)$ it updates $K^i \leftarrow K^{i-1}$ and $\sigma^{i-1} \leftarrow -1$. Observe that restriction (a) does not impose a real limitation in groups of prime order q because $aP = -(-aP) = -(q - a)P$ and either a or $q - a$ is odd. Observe also that the steps introduced in (c) do not affect the overall outcome of the point multiplication as for all integers x we have $x = 2 \cdot x + (-1) \cdot x$.

In [40], the same authors improve on their proposal by first encoding (odd) scalars $a = \sum a_i 2^i$ using only signed binary digits $a_i \in \{\pm 1\}$, and then computing vectors K^i from these coefficients. This not only avoids the $K^i = (0, \ldots, 0)$ situation but also reduces the size of the precomputed table by a factor of two.

4.4.5 Point multiplication in PolarSSL

We analysed the source code of the point multiplication routine deployed in PolarSSL version 1.3.8, published on July 11 2014.[5] The scheduler (function `ecp_comb_fixed` in file `library/ecp.c`) is comb-based, and comments around the code give explicit credit to the results of [39]. However, as a matter of fact the actually implemented algorithm significantly improves on the referred-to work, as we detail below. We believe that this is the first description of this point multiplication method in the academic literature.

PolarSSL borrows from [39] both the restriction to handle only odd scalars and the introduction of flags $\sigma^i \in \{\pm 1\}$ that indicate whether corresponding K^i are considered 'positive' or 'negative'. The novelty here is that the iteration over $K^1, \ldots, K^{d-1}$, that before was solely concerned with fixing the $K^i = (0, \ldots, 0)$ condition, is now replaced by an iteration over the same values where action is taken roughly every second time, namely whenever $K^i_0 = 0$. Concretely, in this case the algorithm sets $\sigma^{i-1} \leftarrow -1$ (similarly to [39]) and replaces K^i by $K^i \boxplus K^{i-1}$, where addition '$\boxplus$' is understood position-wise, carrying over into K^{i+1}. This method ensures that all K^i have $K^i_0 = 1$ (as is easily shown by an inductive argument), and effectively makes the pre-computed table T half the size. On the downside, for recording the carries of the final '$\boxplus$' step, vector $K^{d-1}, \ldots, K^0$ has to be extended by an auxiliary component K^d. More details on the procedure are given in Algorithm 5. Note that in Algorithm 5, for same-size bit-vectors $\alpha, \beta, \gamma, \delta, \epsilon$ we write $(\alpha, \beta) = \gamma \boxplus \delta$ iff $2\alpha_i + \beta_i = \gamma_i + \delta_i$ for all i. Correspondingly we write $(\alpha, \beta) = \gamma \boxplus \delta \boxplus \epsilon$ iff $2\alpha_i + \beta_i = \gamma_i + \delta_i + \epsilon_i$. That is, the addition is bit-wise and the sum is stored in β_i, with α_i taking the carry.

We conclude by describing how resulting sequence $(\sigma^d, K^d), \ldots, (\sigma^0, K^0)$ is

[5] Available at `https://polarssl.org`.

Algorithm 5 PolarSSL's comb encoding

Input: odd scalar a, parameters w, d
Output: coefficients $K^d, (\sigma^{d-1}, K^{d-1}), \ldots, (\sigma^0, K^0)$

1: compute $(\bar{K}^{d-1}, \ldots, \bar{K}^0)$ using Algorithm 4
2: $K^0 \leftarrow \bar{K}^0$
3: $c \leftarrow (0, \ldots, 0)$
4: **for** $i \leftarrow 1$ **to** $d-1$ **do**
5: **if** $\bar{K}^i_0 = 0$ **then**
6: $(c, K^i) \leftarrow \bar{K}^i \boxplus K^{i-1} \boxplus c$
7: $\sigma^{i-1} \leftarrow -1$
8: **else**
9: $(c, K^i) \leftarrow \bar{K}^i \boxplus c$
10: $\sigma^{i-1} \leftarrow +1$
11: **return** $c, (+1, K^{d-1}), (\sigma^{d-2}, K^{d-2},) \ldots, (\sigma^0, K^0)$

encoded in computer memory. PolarSSL imposes the requirement $w \in [2\,..\,7]$ (in practice $w \in \{4, 5\}$ is used, see line 1382 of `ecp.c`) and can hence store each K^i in a separate octet (data type '`unsigned char`'). The remaining eighth bit is used to store the corresponding sign indicator; precisely, $\sigma^i = +1$ and $\sigma^i = -1$ are encoded as `0` and `1`, respectively. For example, if $w = 3$ and $\sigma^i = -1$ and $K^i = (1, 0, 1)$, the in-memory representation is `10000101`.

Similarly to Fact 1 we can state a suffix property for this encoding.

Fact 2 (Suffix property of PolarSSL's comb encoding). Fix a scalar $a \in \mathbb{N}$ and parameters w, d. Denote a's binary expansion with $(a_{wd-1}, \ldots, a_0)$, its (textbook) comb encoding with $(\bar{K}^{d-1}, \ldots, \bar{K}^0)$ where $\bar{K}^i_j = a_{i+jd}$, and its PolarSSL comb encoding with $(K^d, \sigma^{d-1}, K^{d-1}, \ldots, \sigma^0, K^0)$. Then it holds for all $1 \leq t \leq d$ that $(K^{t-1}, \sigma^{t-2}, K^{t-2}, \ldots, \sigma^0, K^0)$ is fully determined by $(\bar{K}^{t-1}, \ldots, \bar{K}^0)$.

4.5 General Procedures for Recovering Noisy Keys

Next we present our algorithms that recover the private keys of DL-based cryptosystems from noisy memory images. Separate algorithms are proposed for OpenSSL and PolarSSL and, thus, each will have its own analysis of success probability. We start with specifying the attack model.

4.5.1 Attack model

In both OpenSSL and PolarSSL, discrete logarithm secret keys and their NAF or comb encodings reside in computer memory simultaneously, at least for a short period of time. Our cold boot attack model hence assumes that the adversary can obtain noisy versions of the original private key and its encoding, and aims at recovering the private key. We assume that a 0 bit will flip with probability $\alpha = \mathbb{P}(0 \to 1)$ and a 1 bit will flip with probability $\beta = \mathbb{P}(1 \to 0)$. Furthermore, we assume that the attacker knows the values of α and β. Such an assumption is possible because an adversary can easily estimate them using an analysis similar to [42]. We refer the reader to that paper for the details.

4.5.2 NAF encodings

Algorithm 6 attempts to recover a key that has been encoded with either the textbook wNAF or the modified NAF of OpenSSL (from Algorithms 2 and 3, respectively). It takes several inputs: the public key, $Q = aP$; the noisy memory image, $\mathcal{M}^*$; the length of the private key, ℓ; the window size, w; a variable parameter, t; and a constant k.

Algorithm 6 Generic key-recovery algorithm for textbook and OpenSSL wNAF.

Input: noisy memory image $\mathcal{M}^*$, reference public key $Q = aP$, parameters ℓ, w, t, k; use $k = 0$ for textbook wNAF recovery, and $k > 0$ otherwise.

Output: secret key a or $\perp$

1: $\mathcal{C}and\mathcal{S}et \leftarrow \emptyset$
2: **for all** $x \in \{0,1\}^{t+w-1}$ **do**
3: calculate partial representation $\mathcal{M}_x$ of x
4: **if** Correlate$(\mathcal{M}_x, \mathcal{M}^*) =$ pass **then**
5: add x to $\mathcal{C}and\mathcal{S}et$
6: **for** $i \leftarrow 2$ **to** $\lfloor (\ell - k + 1 - w)/t \rfloor$ **do**
7: $\mathcal{C}and\mathcal{S}et' \leftarrow \emptyset$
8: **for all** $x \in \{0,1\}^t \times \mathcal{C}and\mathcal{S}et$ **do**
9: calculate partial representation $\mathcal{M}_x$ of x
10: **if** Correlate$(\mathcal{M}_x, \mathcal{M}^*) =$ pass **then**
11: add x to $\mathcal{C}and\mathcal{S}et'$
12: $\mathcal{C}and\mathcal{S}et \leftarrow \mathcal{C}and\mathcal{S}et'$
13: **for all** $x \in \{0,1\}^{k+(\ell-k-w+1 \bmod t)} \times \mathcal{C}and\mathcal{S}et$ **do**
14: $a \leftarrow \sum_{i=0}^{\ell-1} 2^i x_i$
15: **if** $Q = aP$ **then**
16: **return** a
17: **return** $\perp$

We first discuss the textbook NAF, for which $k = 0$. The algorithm will output either a (the private key) or $\perp$, which represents failure. The recovery procedure begins by initialising a set $\mathcal{C}and\mathcal{S}et$ to be empty. The set $\mathcal{C}and\mathcal{S}et$ will store (partial) candidate solutions for the private key a. At each stage of the algorithm we wish to compute t new wNAF digits for each candidate solution. To be certain of outputting the first t signed digits of the wNAF, the algorithm requires knowledge of the least $t + w - 1$ bits of the binary representation (cf. Fact 1). Hence, the first stage of the algorithm (cf. lines 1–5) takes all bit strings of length $t+w-1$ (giving us the ability to calculate the least t signed digits of the wNAF), converts them to integers, then calculates their corresponding wNAFs for positions $b_{t-1}, \ldots, b_0$ (prepending zeros if necessary,

and ignoring any b_j for $j \geq t$ if they exist). The algorithm then compares each bit string and its corresponding wNAF against $\mathcal{M}^*$ via the Correlate function (see Section 4.3). If the candidate passes the Correlate test, then the candidate solution is added to the set $\mathcal{C}and\mathcal{S}et$, otherwise it is discarded. Once all bit strings of length $t+w-1$ have been checked, we move on to the second stage of the algorithm (cf. lines 6–12). We first initialise a set $\mathcal{C}and\mathcal{S}et'$ to be empty. For each string x in $\mathcal{C}and\mathcal{S}et$, we prepend all bit strings of length t to x (giving us the ability to compute the next t signed digits of the wNAF). We then calculate the wNAFs of (the integer conversions of) all the strings. Again, we prepend zeros to the wNAF if necessary, and we ignore any b_j for $j \geq 2t$. Then the algorithm compares each bit string and its corresponding wNAF against $\mathcal{M}^*$ via the Correlate function. If the candidate solution passes the test it is added to $\mathcal{C}and\mathcal{S}et'$. When all appropriate strings have been checked, we overwrite $\mathcal{C}and\mathcal{S}et \leftarrow \mathcal{C}and\mathcal{S}et'$. If we let ℓ' denote the length of the partial candidates, then we repeat the previous stage of the algorithm until $\ell' > \ell - t$ (because, at this point, prepending t bits to the candidate solutions would result in them having a greater length than the private key a). At this juncture the algorithm will prepend all bit-strings of length $\ell - \ell'$ to all the strings in $\mathcal{C}and\mathcal{S}et$ (cf. lines 13–16). Each of these new strings x is then compared against the public key $Q = aP$, via the calculation xP. If there is a match with $Q = aP$, then the algorithm outputs x, otherwise the algorithm outputs $\perp$.

We will now discuss the modifications that we make for the OpenSSL implementation of the wNAF encoding. From Algorithm 3 it is clear that the OpenSSL wNAF only modifies the textbook wNAF in (at most) the most significant $w+1$ digits (excluding leading zeros). Algorithm 6 relies on the fact that textbook wNAFs can be built up in a bit-by-bit fashion from the least significant bit (cf. Fact 1), but this is no longer possible with the modified

wNAF. Therefore, when dealing with the OpenSSL NAF, we include an extra parameter $k > 0$ in Algorithm 6, where $k \in \mathbb{N}^{>0}$. The only difference is that instead of entering the final stage of the algorithm when $\ell' > \ell - t$, we now enter the final stage when $\ell' > \ell - t - k$. That is, we stop k bits earlier than we normally would for the textbook wNAF, and then the final stage appends $\ell - \ell'$ bits to each string in $\mathcal{C}and\mathcal{S}et$ and checks whether any of these new strings matches the private key, a. The reasoning behind this is that if the bit representation of an integer has a leading 1 in position i, then the standard wNAF will only be affected in positions $i+1$ to $i-w+1$. In Algorithm 6, at most we compute $\ell - k - w + 1$ signed digits for each candidate solution. For a uniformly random private key a, the higher we set k, the more likely it is that the textbook wNAF and modified wNAF of a agree in the positions our algorithm computes (since a uniformly random key is more likely to have a leading 1 in bit positions $\ell-1$ to $\ell-k-1$, meaning the first $\ell-k-w+1$ signed digits remain unaffected). This will be discussed in more detail in Section 4.5.4. However, there is a trade-off between running-time and success. A higher k results in a higher success, but the last stage of Algorithm 6 appends bit-strings of at least length k to all surviving candidates. Hence, the greater k is, the longer the running-time of this final phase. A typical value for k would be below 10.

4.5.3 Comb encodings

In this section we consider key-recovery for comb-based methods. The textbook comb encoding together with the original key merely represents a repetition code, and there are standard techniques to recover the key for such a code. Hence, we shall proceed straight to the discussion of PolarSSL combs. To prevent side-channel attacks (cf. Section 4.4.4), the PolarSSL comb uses a lookahead algorithm, so we will need a more sophisticated algorithm than

the standard techniques used for repetition codes. The pseudocode for our algorithm can be found in Algorithm 7. The inputs are: the noisy memory image, $\mathcal{M}^*$; the public key, $Q = aP$; the length of the comb (i.e. the number of prongs), w; the number of comb positions, d; and a variable parameter t. To calculate component K^0 of the comb requires knowledge of bits $a_{(w-1)d}, a_d, \ldots, a_0$ (and only these bits). If we want to calculate K^1 and σ^0, we additionally need bits $a_{1+(w-1)d}, a_{1+d}, \ldots, a_1$, and so on (cf. Fact 2). Our algorithm considers t-many comb components at each stage. During the first stage (cf. lines 1–9) we wish to compute $K^{t-1}, (\sigma^{t-2}, K^{t-2}), \ldots, (\sigma^0, K^0)$ for each candidate solution. To calculate these components only requires knowledge of tw bits (in the appropriate positions of the key). Since PolarSSL only handles odd scalars, there are 2^{tw-1} candidate solutions across these tw bits. For each of these candidate strings, we compare the bits of the string x and its comb with the noisy versions via the Correlate function. If the candidate passes the Correlate test, the string is added to $\mathcal{C}and\mathcal{S}et$ (which we initialize to empty), otherwise it is discarded. We then (cf. lines 10–20) repeat the procedure by combining each surviving candidate with all possible bit combinations in the tw positions that will allow us to compute the next t comb components, which are $K^{2t-1}, (\sigma^{2t-2}, K^{2t-2}), \ldots, (\sigma^t, K^t)$. If ℓ' denotes the length of the current candidates, the algorithm exits this particular For loop when $dw - \ell' \leq tw$ (i.e. when adding t more $\bar{K}^j$ would result in there being more $\bar{K}^j$ than exist for the private key). At this point, the algorithm fills in all the missing bits with all possible combinations (cf. lines 21–26). Then the algorithm checks whether any of the strings is a match for the private key (by using the public information $Q = aP$). If there is a match, the algorithm outputs the string, otherwise it outputs $\perp$.

Algorithm 7 Generic key-recovery algorithm for PolarSSL comb method.

Input: noisy memory image $\mathcal{M}^*$, reference public key $Q = aP$, parameters d, w, t

Output: secret key a or $\perp$

1: $\mathcal{C}and\mathcal{S}et \leftarrow \emptyset$
2: **for all** $x \in \{0,1\}^{tw-1} \times \{1\}$ **do**
3: **for** $j \leftarrow 0$ **to** $t-1$ **do**
4: $\bar{K}^j \leftarrow (x_{(j+1)w-1}, \ldots, x_{jw})$
5: compute $K^{t-1}, (\sigma^{t-2}, K^{t-2}), \ldots, (\sigma^0, K^0)$
6: using lines 2–10 of Algorithm 5
7: calculate partial representation $\mathcal{M}_x$
8: **if** Correlate$(\mathcal{M}_x, \mathcal{M}^*) =$ pass **then**
9: add x to $\mathcal{C}and\mathcal{S}et$
10: **for** $i \leftarrow 2$ **to** $\lceil d/t \rceil - 1$ **do**
11: $\mathcal{C}and\mathcal{S}et' \leftarrow \emptyset$
12: **for all** $x \in \{0,1\}^{tw} \times \mathcal{C}and\mathcal{S}et$ **do**
13: **for** $j \leftarrow 0$ **to** $it-1$ **do**
14: $\bar{K}^j \leftarrow (x_{(j+1)w-1}, \ldots, x_{jw})$
15: compute $K^{it-1}, (\sigma^{it-2}, K^{it-2}), \ldots, (\sigma^0, K^0)$
16: using lines 2–10 of Algorithm 5
17: calculate partial representation $\mathcal{M}_x$
18: **if** Correlate$(\mathcal{M}_x, \mathcal{M}^*) =$ pass **then**
19: add x to $\mathcal{C}and\mathcal{S}et'$
20: $\mathcal{C}and\mathcal{S}et \leftarrow \mathcal{C}and\mathcal{S}et'$
21: **for all** $x \in \{0,1\}^{wd-(\lceil d/t \rceil - 1)tw} \times \mathcal{C}and\mathcal{S}et$ **do**
22: **for** $j \leftarrow 0$ **to** $d-1$ **do**
23: $\bar{K}^j \leftarrow (x_{(j+1)w-1}, \ldots, x_{jw})$
24: $a \leftarrow \sum_{j=0}^{d-1} \sum_{i=0}^{w-1} 2^{j+id} \bar{K}_i^j$
25: **if** $Q = aP$ **then**
26: **return** a
27: **return** $\perp$

Remark 1. We note that in some cases there is a simple way to slightly increase the efficiency of Algorithm 7. If ℓ is the length of the private key, but $\ell \neq wd$, then the private key will have to be prepended with $wd - \ell$ zero bits. Algorithm 7 can be improved by utilising this information and only considering candidate solutions with zeros in these particular positions. However, as in

practice $w = 4$ or $w = 5$ is used and we consider $\ell = 160$ in our simulations, there will be no need for prepended zeros and our algorithm will run exactly as presented in Algorithm 7.

Remark 2 (Optimality of Algorithms 6 and 7). We do not claim that Algorithms 6 or 7 are the optimal procedures for recovering keys in their respective scenarios. However, these algorithms are appealing because we are able to provide a theoretical analysis of the success rate (cf. Section 4.3). Furthermore, the experimental results we obtain from these algorithms are good in practice, as we shall see in the coming sections.

4.5.4 Success analysis of OpenSSL implementation

We now analyse the success probability of Algorithm 6 when combined with the Correlate test from Section 4.3. The success probability is relatively straightforward to calculate if the input is an image of a textbook wNAF: the correct candidate will pass the Correlate test (Equation (4.3.4)) with probability γ^2, where $\gamma = \int_0^C \chi_1^2(x)\mathrm{d}x$. Hence, the probability of recovering the correct key would be $\gamma^{2\cdot\lfloor(\ell+1-w)/t\rfloor}$ because there are $\lfloor(\ell+1-w)/t\rfloor$-many Correlate tests to pass and the probability of passing each test is independent (because Correlate considers only the newly computed bits at each stage).

However, since a modified NAF is used in OpenSSL, the corresponding analysis of success will differ slightly. Fortunately, the difference between textbook NAF and modified NAF is only in the most significant $w + 1$ bits (and sometimes there is no difference at all): if the leading 1 bit of the discrete logarithm key is in position i then, at most, only signed digits $i - w + 1$ to $i + 1$ of the standard wNAF will be affected by the transformation to the modified wNAF. Therefore the standard and modified wNAFs will agree up to position $i - w$. Algorithm 6 only computes the least significant $j =$

$\ell - k - w + 1 - (\ell - k - w + 1 \bmod t)$ digits of the wNAF, i.e., $b_{j-1}, \ldots, b_0$. Therefore, we must now bound the probability that a randomly chosen private key's standard wNAF is equal to its modified NAF up to digit b_{j-1}. If the private key has a 1 bit anywhere between positions $j + w - 1$ and $\ell - 1$ then the computed NAF digits will be identical to the modified wNAF digits up to position $j - 1$, and then the multinomial test will behave exactly as expected (having probability γ of passing each test). The probability of a 1 bit appearing in any of these positions is precisely

$$1 - 2^{-k-(\ell-k-w+1 \bmod t)} \quad .$$

If we let M-NAF denote the modified wNAF, and wNAF_{j-1} (resp. M-NAF_{j-1}) denote digits 0 to $j - 1$ of wNAF (resp. M-NAF), then it follows that

$$\begin{aligned} \mathbb{P}(\text{success}) &\geq \mathbb{P}(\text{success}|\text{wNAF}_{j-1} = \text{M-NAF}_{j-1})\mathbb{P}(\text{wNAF}_{j-1} = \text{M-NAF}_{j-1}) \\ &= \left(1 - 2^{-k-(\ell-k-w+1 \bmod t)}\right) \cdot \gamma^{2 \cdot \lfloor (\ell-k+1-w)/t \rfloor} \quad . \end{aligned}$$

Thus, by setting the thresholds k and C (and, hence, γ) appropriately, we can achieve any desired success rate (potentially at the expense of a long running time since larger k and C will result in a longer running time).

If either $\alpha = 0$ or $\beta = 0$ our algorithm has a slightly different analysis. Since neither α nor β will be zero in practice, we have relegated this analysis to Section 4.7.

4.5.5 Success analysis of PolarSSL implementation

Given the previous discussion regarding the success of recovering keys of the NAF algorithms, it is now very easy to analyse the success of Algorithm 7. It is clear from the algorithm that there are $\lceil d/t \rceil - 1$ Correlate tests to pass. The correlate function is described in Equation (4.3.4), and the correct candidate has probability γ^2 of passing the test, where $\gamma = \int_0^C \chi_1^2(x)\mathrm{d}x$. Since

each Correlate test only considers the newly calculated bits, the probability of passing each Correlate test is independent, so we have $\mathbb{P}(\text{success}) = \gamma^{2\cdot(\lceil d/t \rceil - 1)}$.

4.6 Implemented Simulations of Key Recovery

We present the results of some simulations of Algorithms 6 and 7 using the Correlate test from Equation (4.3.4). Unless otherwise stated, we ran 100 tests for each given set of parameters. The results for OpenSSL can be seen in Table 4.1 and those for PolarSSL in Table 4.2. The values displayed in these tables are merely to support the validity of our theoretical analysis, and they do not represent the practical limits of our algorithms. However, it is clear that any algorithm attempting key recovery in the PolarSSL and OpenSSL settings will not be able to match the performance of the RSA algorithm of Chapter 3. We discuss the reasons why in Section 4.10. For each set of parameters, the table shows the predicted theoretical success of the algorithms and the success rate we achieved with our 100 simulations. Note that as the noise rate increases the success rate will slowly decline. However, for OpenSSL, the success rate for $\beta = 0.15$ was higher than for $\beta = 0.10$, despite all other parameters being the same. This is merely an outlier, and a result of the limited number of simulations we ran. If we were to perform a much larger number of simulations, we expect this outlier to disappear. All values we have used for α and β are typical values that might arise in a real cold boot attack, but the higher values of β are much rarer in practice. For small values of β (which are most common) our algorithms have a good success rate. For example, for OpenSSL we have a success rate of 45% when $\beta = 0.05$. Furthermore, for small values such as this we could significantly improve the success rate by increasing the threshold C. For such small values

w	α	β	t	C	k	χ^2 est.	prac. suc.
2	0.001	0.01	7	6	3	0.51	0.92
2	0.001	0.05	10	3.5	3	0.15	0.45
2	0.001	0.10	10	3.5	3	0.15	0.17
2	0.001	0.15	10	3.5	3	0.15	0.20
2	0.001	0.20	14	2	3	0.02	0.07
2	0.001	0.25	12	2	3	0.01	0.06
2	0.001	0.30	12	2	3	0.01	0.04
2	0.001	0.35	14	0.75	3	0	0.02

Table 4.1: Results of simulations of cold boot attacks against the point multipliers of OpenSSL. All simulations used 160-bit keys, and we ran 100 simulations for each row in the table. The theoretically estimated success probabilities, based on the convergence to the chi-squared distribution, are in the columns labelled 'χ^2 est'. Note that the effective success rates of our implemented attacks, in columns 'prac. suc.', are generally much larger.

of β this would not greatly affect the running time. Note that the majority of the experiments were conducted in a $1 \to 0$ region (where $\alpha \ll \beta$). This choice will not affect the theoretical success rate of the algorithm, but is likely to have an impact on the running-time. In the PolarSSL setting, the difference in performance of our algorithm between a $1 \to 0$ and $0 \to 1$ region will be very small, due to the approximately equal distribution of 1 and 0 bits in the private key. However, for OpenSSL there will be a noticeable difference, which is explored further in Section 4.9.

4.7 Analysis of Success for the Z-Channel

Throughout this chapter we have assumed that both $\alpha := \mathbb{P}(0 \to 1)$ and $\beta := \mathbb{P}(1 \to 0)$ are non-zero (as they are likely to be in practice), and the analyses of Sections 4.5.4 and 4.5.5 were dependent on this fact. Here we briefly discuss how to handle the case when either α or β is zero (if both are zero, then no bits will flip, hence the noisy key will be identical to the original key). We will first discuss the OpenSSL case (see Section 4.5.2). We assume

w	d	α	β	t	C	χ^2 est.	prac. suc.
4	40	0.001	0.01	2	7	0.73	0.81
4	40	0.001	0.02	2	5	0.38	0.65
4	40	0.001	0.03	2	4	0.17	0.60
4	40	0.001	0.05	2	3.5	0.09	0.58
4	40	0.001	0.06	2	3	0.04	0.55
4	40	0.001	0.07	2	3	0.04	0.52
4	40	0.001	0.08	2	2.5	0.01	0.37
4	40	0.001	0.10	2	2.5	0.01	0.08

Table 4.2: Results of simulations of cold boot attacks against the point multipliers of PolarSSL. All simulations used 160-bit keys, and we ran 100 simulations for each row in the table.. The theoretically estimated success probabilities, based on the convergence to the chi-squared distribution, are in the columns labelled 'χ^2 est'. Note that the effective success rates of our implemented attacks, in columns 'prac. suc.', are generally much larger.

that $\alpha = 0$ (the case $\beta = 0$ follows easily), so there will be no 0 to 1 bit flips from the original key to the noisy key. We cannot perform a multinomial test on the 0s of the candidate solutions (because this requires non-zero α), so instead we merely discard any candidate solution in which a 0 must have flipped to a 1. Notice that it is impossible to reject the correct solution via this test (because $\mathbb{P}(n_{01} = 0) = 1$ for the correct candidate). The Correlate test is then

$$\text{Correlate}_C(s_i, r) = \text{pass} \quad \Leftrightarrow \quad \text{Correlate}^1(s_i, r) < C \wedge n_{01}^i = 0. \qquad (4.7.1)$$

Recall that the pruning phase used by the HS algorithm [44] only considered whether $n_{01}^i = 0$. Hence, in the Z-channel, our Correlate test is an extension of the HS algorithm because we also consider the extra information concerning $1 \to 0$ flips, whereas this information was ignored by the HS algorithm. The theoretical success rate of Algorithm 6 with $k = 0$ is $\gamma^{\lfloor(\ell+1-w)/t\rfloor}$, and its success rate is at least $\left(1 - 2^{-k-(\ell-k-w+1 \bmod t)}\right) \cdot \gamma^{\lfloor(\ell-k+1-w)/t\rfloor}$ if $k > 0$, where γ is the probability of passing a single multinomial test. If instead we have

$\beta = 0$, then

$$\text{Correlate}_C(s_i, r) = \text{pass} \quad \Leftrightarrow \quad \text{Correlate}^0(s_i, r) < C \wedge n_{10}^i = 0. \qquad (4.7.2)$$

The success of the PolarSSL algorithm (see Section 4.5.3) now follows easily. The Correlate functions are those in Equations (4.7.1) and (4.7.2) (for $\alpha = 0$ and $\beta = 0$ respectively) and the success rate can be estimated by $\gamma^{\lceil d/t \rceil - 1}$.

4.8 Running-Time Analysis

So far we have provided a theoretical analysis for the success rate of our key-recovery procedures from Section 4.5. To complete the picture, we would also like to be able to provide an analysis of the running-time of the algorithms. Unfortunately, such an analysis appears to be very difficult to obtain in our setting. If we were able to bound the probability of a 'Type II error' (not rejecting an incorrect solution) in the multinomial test, this would allow us to bound the running-time. In the RSA setting, a typical assumption is that incorrect candidate solutions are uniformly random and independent of all previous key bits.[6] In the two scenarios we consider, such assumptions clearly do not hold. The bits of a particular candidate wNAF or comb solution are entirely dependent on *all* the previous key bits. Given that we cannot employ any independence assumptions, we are unable to provide a theoretical analysis of the running-time of our algorithms.

Notice, however, that the Correlate function can be modified to produce a test that has a bounded running-time. We have previously suggested setting a threshold and discarding any solutions that do not have a Correlate value that

[6]The analyses of [42, 44] used such an assumption, whereas Chapter 3 made use of a slightly weaker assumption.

falls below this threshold. This meant that the algorithm would output lists of a variable size. Instead, we can modify our algorithms to output shorter (or even fixed-sized) lists, thereby allowing the running-time to be easily bounded. The Correlate test could be modified so that it computes Correlate^0 and Correlate^1 (Equations (4.3.2) and (4.3.3) respectively), and then sums these two values. Then the algorithm would output the L-many candidates with the lowest values of Correlate^0 + Correlate^1, where L is a parameter chosen by the attacker. This algorithm clearly has a bounded running-time, but a theoretical analysis of success rate is lacking for this particular approach. Note that this approach would be broadly comparable to the technique we used in Chapter 3 for the RSA setting, except there we used a maximum-likelihood test, as opposed to the multinomial test we used in this chapter. We could also use a maximum-likelihood approach as our Correlate function in this chapter, but due to the way we generate candidate solutions it is unlikely that such an algorithm would be successful since the algorithm would have difficulty distinguishing the good and bad solutions (see Section 4.10 for an expanded exposition of this point). Fortunately, the multinomial test does not need to distinguish the good and bad solutions. It only requires the correct solution to be 'sufficiently close' to the noisy information, and therefore the success rate is independent of the number (and distributions) of incorrect candidates. Indeed, preliminary experiments were conducted using the ML approach, but non-zero success rates proved elusive for our limited number of trials.

4.9 Comparison of Ground States

In this section we study the impact of the ground states on our cold boot recovery algorithms. Recall that there are two ground states: 0 and 1. In a 0 region, 1 bits have a reasonably high chance of flipping to a 0, but 0 bits will

α	β	C	suc.	min.	LQ	med.	UQ	max.
0.001	0.01	4	0.338	1	2	2	4	120
0.01	0.001	4	0.524	1	48	144	483	82944
0.01	0.001	2.5	0.195	2	16	48	144	2880

Table 4.3: Quartile data for the number of candidate solutions that passed the final Correlate test of Algorithm 6 (i.e., the size of $\mathcal{C}and\mathcal{S}et$ at line 13). For each set of parameters we ran 1000 tests with 160-bit keys, and we set $t = 7$.

have a very low probability (typically 0.001) of flipping to a 1. In a 1 region the opposite is true. Typically, RSA cold boot analyses assume that private keys consist of approximately an equal number of 0 and 1 bits. Indeed, we made this exact assumption in Chapter 3. Evidently then, the recovery algorithms would run equally well in both types of region. In contrast, in an OpenSSL wNAF there are significantly more 0 bits than 1 bits, particularly for larger w. As a result our algorithm may have different performance depending on the type of decay region. The theoretical success rate is obviously unaffected, but the running-time varies significantly, as Table 4.3 shows. Note that results were implemented with the textbook NAF, rather than the modified version.

For all tests we set the key size to be 160 bits and we conducted 1000 tests for each set of parameters. We chose to run 1000 tests in order to eradicate any statistical anomalies that might arise from using small sample sizes. For the first two sets of parameters we set w to be 2, the threshold C was 4, and t was 7. For the first set of parameters, we set $\alpha = \mathbb{P}(0 \to 1) = 0.001$ and $\beta = \mathbb{P}(1 \to 0) = 0.01$, to represent a 1-to-0 region. For the second set, we reversed these values, so $\alpha = \mathbb{P}(0 \to 1) = 0.01$ and $\beta = \mathbb{P}(1 \to 0) = 0.001$, which represents a 0-to-1 region. For each test in which we successfully recovered the private key we kept a record of the number of solutions that passed the final Correlate test. Table 4.3 displays the quartiles of this data (the minimum, lower quartile, median, upper quartile and maximum). It is clear from the table that the

algorithm had to consider many more solutions in the 0-to-1 region, which results in a much greater running time. However, this could be partially explained by the much higher success rate in the 0-to-1 region. In the 1-to-0 region, the success rate was 0.338, compared to 0.524 in the 0-to-1 region. This elevated success rate will obviously result in more candidates being checked, but this is not the only reason. The convergence of the multinomial test to the chi-squared distribution is dependent on the expected values of n_{10} and n_{01} (the higher they are, the better the convergence), where we recall that n_{ij} is the number of i bits in the candidate solution that map to j bits in the noisy version of the key. By changing the values of α and β, we change the expected values of n_{ij}, which results in varying success probabilities, despite having the same threshold. To counteract this problem, we ran another set of experiments. For $\alpha = 0.01$ and $\beta = 0.001$ we re-ran the simulations, but with the threshold now set to $C = 2.5$. The success probability for this new set of tests was 0.195, which is much lower than the success for the 1-to-0 region. Despite this, however, the quartiles were still much higher for the 0-to-1 region. The explanation for this phenomenon appears to be simple. In a wNAF, the 1 bits are sparse. In a 1-to-0 region, if we observe a 1 bit in the noisy version of the key then, with high probability, the private key has a 1 bit in that particular position. Since 1 bits are infrequent, there will be very few candidate solutions that have a 1 bit in the necessary positions. Conversely, in a 0-to-1 region, if we observe a 0 bit in the noisy key then, with high probability, the private key has a 0 bit in the corresponding position. Unfortunately, since 0 bits are abundant in a wNAF, there are typically many candidates that have 0 bits in these positions. Hence, whilst the success is unaffected by the ground state, the running-time will be much greater in a 0-to-1 region because there will be many more incorrect candidates that pass the Correlate test.

For the PolarSSL comb the distribution of bits is not uniform (since some bits are always set to be 1), which will result in slightly different performances in the two regions. The running-time will not vary considerably, but for particularly small values of α and β we might see a slight difference in the success rates of simulations.

4.10 Comparison with the RSA Setting

At first glance our experimental results for both OpenSSL's wNAF and PolarSSL's comb multiplier (Tables 4.1 and 4.2) appear to be inferior to corresponding results in the RSA setting. Whilst this is true, it should not come as a surprise. Analyses in the RSA setting enjoy several benefits. The major advantage they had was the relationship between key bits via equations such as $N = pq$, where N is the public exponent, and p and q are the private primes. There are four such equations relating the five components of the RSA private key. Heninger et al. [44] showed that if there is a partial solution for the private RSA key and an adversary wishes to discover the next bit of each of the five private key components, then the RSA equations give only two possible solutions for the string of five bits, rather than the thirty-two solutions that would need to be checked in a brute-force search. Hence, when the RSA algorithms calculate possible solutions across a new set of $5t$ bits, if there are M surviving candidates from the previous pruning phase, there will be $M \cdot 2^t$ possible solutions to check at the next stage. These solutions may then be tested to discard unlikely candidates. In the NAF and comb settings, such strong structure in the private key does not exist. Hence, when we consider solutions in a string of $5t$ bits, if M candidates pass the previous pruning phase, then we have to consider $M \cdot 2^{5t}$ solutions (compared to $M \cdot 2^t$ for the RSA setting), and then discard unlikely candidates. Furthermore, when the

RSA algorithms calculate the two possible solutions for a particular set of five bits, the two solutions have a Hamming distance of four. Consequently, if the correct key has high likelihood of coming from the noisy information (as expected), then the second possible solution will have a low likelihood. This allows the RSA algorithms to easily discard incorrect candidates with high probability. Unfortunately, in our settings we will have to consider many solutions that have Hamming distance less than four from the correct solution. The solutions with low Hamming distance from the correct key will have a high probability of passing the threshold test. Bearing these facts in mind, it is quite clear that any algorithm in the NAF or comb settings will not be able to compete with the RSA algorithms in terms of the cross-over probabilities that can be handled for the asymmetric channel. Furthermore, the discussion above highlights why a maximum-likelihood approach is unlikely to be successful in this setting: there are too many incorrect candidates to consider, so there is a high probability that some of these will have a higher ML value than the correct candidate.

4.11 Conclusions

In this chapter we have proposed key-recovery algorithms for various discrete logarithm cryptosystems, with particular emphasis on the widely deployed PolarSSL and OpenSSL implementations. These algorithms represent a large improvement over previous key-recovery algorithms for discrete logarithm cold boot attacks. We provided a theoretical analysis that lower-bounds the success of our algorithms. Furthermore, the statistical test we use in our framework provides an avenue to derive an algorithm with arbitrarily high success rates in the RSA setting when the errors are asymmetric. Such results were only previously available in the symmetric setting. We provided results of several

key-recovery simulations, both for PolarSSL and OpenSSL, that fully support our theoretical analyses and show that our attacks are practical. An open problem is to bound the running-time of our algorithms, but as previously discussed, this appears to be a difficult problem. It would be possible to apply the coding-theoretic techniques of Chapter 3 to our algorithms, but given the nature of the OpenSSL and PolarSSL encodings (such as the high proportion of 0 bits for OpenSSL), the bounds obtained will be very far from what is achievable in practice. For this reason, we omit such an analysis from this chapter.

Part II

Related Randomness Attacks

Chapter 5

Related Randomness Attacks for Public-Key Encryption

In this chapter we will look at cryptography from the perspective of the implementer. Previously we studied how to subvert traditional security notions to compromise security of cryptographic schemes. We will now study how to strengthen security models in order that they capture attacks not currently modelled in the literature. Not only will we present new, more applicable models, but we will show how to construct schemes that are secure in these new models.

5.1 Introduction

Modern cryptographic primitives are heavy consumers of randomness. Unfortunately, random number generators (RNGs) used to provide this randomness often fail in practice [26, 31, 34, 35, 22, 3, 25, 60]. This is due to issues including poor algorithmic design, software bugs, insufficient or poor estimation of system entropy, and the handling of randomness across virtual machine resets [68]. The results of randomness failures can be catastrophic and newsworthy in practice – DSA, ECDSA and Schnorr private signing keys can be

exposed [12, 68]; plaintext recovery for low entropy plaintext becomes possible in the the public-key encryption setting; key generation processes can be severely weakened [22, 56, 43, 13]; ephemeral Diffie-Hellman keys can become predictable leading to compromise of session keys [31]; and electronic wallet security can be compromised [15].

Evidently, randomness failures are a major problem in practice. The cryptography research community has begun to address this problem only relatively recently [69, 70, 48, 4, 81, 68]. Accepting that randomness failures are endemic and unlikely to be eliminated in totality, a basic approach is to try to *hedge* against randomness failures, that is, to design cryptographic primitives that still offer a degree of security in the face of randomness failures.

Work in this direction can be summarised as follows:

- For signatures, there is a folklore de-randomisation technique which neatly sidesteps security issues arising from randomness failures: simply augment the signature scheme's private key with a key for a pseudo-random function (PRF), and derive any randomness needed during signing by applying this PRF to the message to be signed; meanwhile verification proceeds as normal. This does require a small modification to the original signing scheme (the inclusion of an additional private key component), but renders trivial the problem of dealing with bad randomness for signatures. We believe the first formalisation of this technique appeared in 1998, courtesy of M'Raïhi et al. [62], but more recently this technique has been formalised in RFC 6979 [66] with regards to implementations of DSA and ECDSA.

- In the private-key (symmetric) encryption setting, Rogaway [69] argued for the use of nonce-based encryption, thus reducing reliance on randomness. Rogaway and Shrimpton [70] initiated the study of misuse-resistant

authenticated encryption (MRAE), considering the residual security of AE schemes when nonces are repeated. Katz and Kamara [48] considered the security of symmetric encryption in a chosen-randomness setting, wherein the adversary has complete control over the randomness used for encryption (except for the challenge encryption which uses fresh randomness). We will briefly encounter the model of Kamara and Katz in Section 5.7.

- In the public-key encryption (PKE) setting, Bellare et al. [4] considered security under *chosen distribution attack*, wherein the joint distribution of message and randomness is specified by the adversary, subject to containing a reasonable amount of min entropy. The PKE scheme designer's challenge is to find a way of 'extracting' this entropy in a secure way. Bellare et al. gave several designs for PKE schemes achieving this notion in the Random Oracle Model (ROM) and in the standard model. This is a powerful and general approach, but does have its limitations: under extreme failure conditions, the joint message-randomness distribution may simply *fail* to contain sufficient entropy, at which point all security guarantees may be lost; moreover, for technical reasons, the model in [4] requires the target public key to be hidden from the adversary until all encryption queries have been made. This is impractical in real world applications.

- Also in the PKE setting, Yilek [81], inspired by virtual machine reset attacks in [68], considered the scenario where the adversary does not know the randomness (in contrast to the chosen-randomness setting of [48]), but can instead force the reuse of random values that are otherwise well-distributed and unknown to the adversary. This is referred to in [81] as the *Reset Attack* (RA) setting. To fully reflect the reality of randomness failures in this setting, Yilek provides the adversary with

the ability to encrypt chosen messages under adversarially generated public keys using the unknown but repeated random values. This makes his model very powerful, to the extent that certain trivial attacks must be excluded by assuming the adversary is *equality-pattern respecting*. In [81], Yilek also gave a general construction in which the random coins of the encryption algorithm are used as a key to a PRF, the input to the PRF is the public key concatenated with the message to be encrypted, and the output of the PRF is then used as the 'randomness' for the encryption algorithm. This is sufficient to achieve security in his RA setting. Note that the RA security model is incomparable with the CDA model of [4]. Neither notion of security implies the other.

- Ristenpart and Yilek [68] studied the use of 'hedging' as a general technique for protecting against broad classes of randomness failures in already-deployed systems, and implemented and benchmarked this technique in OpenSSL. Hedging in the sense of [68] involves replacing the random value r required in some cryptographic scheme with a hash of r together with other contextual information, such as a message, algorithm or unique operation identifier, etc. Their results, while applying to a variety of different randomness failure types (see in particular [68, Figure 3]), all have their security analyses restricted to the ROM.

5.1.1 Motivation

Inspired by the challenge of preserving security under randomness failures, we initiate the study of security for PKE in what we call the *Related Randomness Attack* (RRA) setting. Our RRA setting builds on the RA setting from [81] and brings the theory of hedging PKE against randomness failures closer to practice. As we shall see, it also has interesting connections with related-key

attacks for PRFs and PKE, as developed in [7, 5, 6, 8, 79], and leakage resilient cryptography (and in particular, the techniques developed in [23] to provide security for PKE in the auxiliary input setting).

In our RRA setting, the adversary can now not only force the reuse of existing random values as in the RA setting, but can also force the use of *functions of* those random values. This power is analogous to the power granted to the adversary in the Related-Key Attack (RKA) setting, wherein an adversary is able to tamper with private (or secret) keys used during cryptographic operations ([14, 5, 49]). The RA setting arises as the special case of our RRA setting where only the identity function is allowed. The extra adversarial power in the RRA setting allows the modelling of reset attacks in which the adversary does not have an exact reset capability, but where the randomness used after a reset is in some way related to that used on previous resets. Such behaviours were observed in the experimental work in [68]. Furthermore, our RRA setting allows modelling of situations where the randomness used in a scheme comes from a PRNG which is not regularly refreshed with new entropy, but which steps forward under some deterministic state evolution function $\texttt{Next}$ and output function $\texttt{Out}$; here the appropriate functions in our RRA setting would be the compositions $\texttt{Out}(\texttt{Next}^i(\cdot))$.

More generally, RRA security has a strong theoretical motivation as being a stepping stone towards giving the adversary enhanced control over the inputs to cryptographic algorithms – messages (in the standard PKE setting), keys (in the RKA setting), and now randomness (in our new RRA setting).

5.1.2 Bad randomness in practice

In this section we will consider an example of a public-key encryption scheme that is secure according to traditional notions, but is trivially insecure when

Algorithm $\texttt{ElG.K}(1^\lambda)$:	Algorithm $\texttt{ElG.E}(pk, m)$:	Algorithm $\texttt{ElG.D}(sk, c)$:
$(\mathbb{G}, q, g) \leftarrow_\$ \texttt{GenGrp}(1^\lambda)$	$r \leftarrow_\$ \{0, \ldots, q-1\}$	$m \leftarrow (c_1^{sk})^{-1} \cdot c_2$.
$x \leftarrow_\$ \{0, \ldots, q-1\}$	$c_1 \leftarrow g^r$	
$pk = g^x$	$c_2 \leftarrow m \cdot pk^r$	
$sk = x$	return (c_1, c_2).	
return (pk, sk).		

Figure 5.1: The ElGamal public-key encryption scheme.

an adversary can obtain encryptions under related randomness. Let us consider the ElGamal encryption scheme [30], which can be seen in Figure 5.1. Suppose an adversary submits the query (m_0, m_1) to his challenge oracle. The challenger will return the encryption $(c_1, c_2) = (g^r, y^r \cdot m_b)$, where r was the randomness chosen for encryption, y is the public key and b is the bit chosen in the CPA/CCA game. Now suppose that the adversary somehow manages to obtain an encryption of message m under randomness $r/2$. The ciphertext will be of the form $(g^{r/2}, y^{r/2} \cdot m)$. The adversary can simply square the right hand side to obtain $y^r \cdot m^2$, and then multiply by the inverse of m^2 to find y^r. Once the adversary has y^r, he simply multiplies c_2 by the inverse of y^r, which reveals m_b. Hence, the ElGamal encryption is trivially insecure if an adversary has access to an encryption oracle that encrypts with randomness that is related to that used in the challenge query. The question then is, can encryption schemes be made secure even if the adversary has access to such an oracle? The answer to this question is 'yes', as we shall see in the coming sections.

The previous example was merely theoretical, but we will now briefly discuss the Sony PS3 randomness failure that resulted in private signing keys being exposed [12]. Before we discuss the vulnerability we must briefly discuss Digital Signatures. Digital signature schemes are designed to provide message integrity in the public-key setting. Each user will generate his own key pair

(pk, sk), of which pk is made public and sk remains private. To provide integrity, the user 'signs' a message m with the private key, and outputs the message, and the signature, denoted σ. In order to verify the integrity of the message, the receiver inputs the message m, the signature σ and public key pk to the Verify algorithm. If the Verify algorithm outputs 1, then the signature is valid for the message, otherwise the signature is invalid. An ideal security property of such is scheme is that no adversary can produce signatures on a message of his choosing, but the security games and notions are irrelevant for our next discussion.

A popular digital signature scheme is ECDSA (Elliptic Curve Digital Signature Algorithm), which can be seen in Figure 5.2. The security of the scheme relies on the fact that a uniformly random r is chosen for each signature. Unfortunately, it was discovered that Sony's PS3 implementation of the algorithm was using the same 'random' value r for every signature. This allowed hackers to recover Sony's signing key, which obviously allowed them to forge signatures on any software they desired. We will now discuss how the hackers were able to obtain the signing key.

The algorithms for ECDSA can be seen in Figure 5.2. In the key generation phase, P is a point on the elliptic curve and q is a prime, both of which are randomly chosen. Suppose that the hacker obtains two signatures (k, s) and (k, s') for messages m and m' respectively for which the q most significant bits of Hash(m) and Hash(m') are distinct. Since the signatures are public and the first components of both signatures agree, the adversary can detect that the same randomness was used for both signatures, and can therefore proceed with the following attack. The values z and z' are easy to compute because they both use the public hash function. The hacker has the two values s and s', and can compute $s - s' = r^{-1}(z - z') \bmod q$. It is then possible to compute $r = (z - z')/(s - s') \bmod q$. Finally, we know that $s = r^{-1}(z + kd) \bmod q$. The

hacker now knows s, r, z and k, and can now obtain the signing key d via the computation $d = (sr - z)k^{-1} \bmod q$.

Alg. $\texttt{ECDSA.K}(1^\lambda)$:	Alg. $\texttt{ECDSA.S}(sk, m)$:	Alg. $\texttt{ECDSA.V}(pk, m, \sigma)$:
$(\mathbb{G}, P, q) \leftarrow_\$ \texttt{GenGrp}(1^\lambda)$	$e = \text{Hash}(m)$	$e = \text{Hash}(m)$
$d \leftarrow_\$ \{0, \ldots, q-1\}$	$z = \text{MSB}_q(e)$	$z = \text{MSB}_q(e)$
$pk = dP$	$r \leftarrow_\$ \{0, \ldots, q-1\}$, (1)	$w \leftarrow s^{-1} \bmod q$
$sk = d$	$R \leftarrow rP = (x_1, y_1)$	$u_1 \leftarrow zw \bmod q$
return (vk, sk).	$k = x_1 \bmod q$	$u_2 \leftarrow kw \bmod q$
	if $k = 0$,	$(x_1, y_1) \leftarrow u_1 P + u_2 dP$
	return to line (1)	if $k \equiv x_1 \bmod q$
	$s = r^{-1}(z + kd) \bmod q$	return 1
	if $s = 0$,	else
	return to line (1)	return 0.
	return $\sigma = (k, s)$.	

Figure 5.2: The ECDSA scheme.

5.1.3 Our contributions

RRA security model In this chapter, we provide a strong model and security definition for PKE in the RRA setting, which we name RRA-ATK security (where ATK = CPA or CCA) . Our model is inspired by that of Yilek for the RA setting: via access to an **Enc** oracle, we allow the adversary to get arbitrary messages encrypted under arbitrary public keys, using functions ϕ of an initial set of uniformly random, but unknown, values. The public keys can even be maliciously generated, and the adversary can of course know all the corresponding private keys. The adversary is tasked with winning an indistinguishability-style game, via an **LR** oracle which gives access to encryptions of left or right messages with respect to an honestly generated target public key pk^*, but again where the adversary can force the use of functions ϕ of the initial random values. When the functions ϕ are limited to coming from some set Φ, we speak of a Φ-restricted adversary.

Because the adversary may know all but one of the private keys, it can check that its challenger is behaving correctly with respect to its encryption queries. This also rules out the possibility of achieving RRA-ATK security for any randomness recovering PKE scheme, like RSA-OAEP [10] and PKE schemes based on the Fujisaki-Okamoto transformation [29]. Moreover, the encryption queries concern public keys that are outside the control of the challenger. This increases the technical challenge of achieving security in the RRA setting. This facet of the RRA setting bears comparison with the RKA setting for PKE [6, 8, 79]. In the RKA setting, the tampering via related key functions only affects the PKE scheme's private key, and so only comes into play when simulating *decryption* queries. By contrast, it is *encryption* queries that require special treatment in our RRA setting.

Given the power of the adversary in the RRA setting, we cannot hope to achieve security against all sets of adversarial queries. So we must restrict the adversary from achieving 'trivial wins'. In particular, no matter what set of functions Φ the adversary uses to modify the random values, it can win simply by making the same combination of messages and functions in its **LR** queries as in its encryption queries under target public key pk^*. These must then be ruled out by identifying forbidden combinations of queries. Thus we must assume the adversary is equality-pattern respecting, for a suitable definition that extends that of [81]. However, this alone is not enough: from the RKA setting and the results of [24], we already know that certain sets of functions Φ are too powerful, in allowing trivial wins for the adversary in that setting. We should not be surprised that the same is true in our RRA setting. For example, if constant functions are allowed in the RRA setting, an adversary would trivially be able to determine which of two challenge message is encrypted in a ciphertext from the **LR** oracle. Analogously to [7], we identify collision-resistance and output-unpredictability as necessary conditions on the

set of functions Φ which the adversary uses to transform random values in its attack.

ROM construction We are able to show that, in the ROM, these necessary conditions on the function set Φ are actually also sufficient. More specifically, we show how to transform any IND-ATK secure PKE scheme `PKE` into a new PKE scheme `Hash-PKE` that is RRA-ATK secure, for equality-pattern respecting, Φ-restricted adversaries, simply by hashing the random input together with the public key and message during encryption. In fact, this is just an application of the hedging approach from [68], and an instance of the randomized-encrypt-with-hash (REwH) scheme from [4]. Our result then shows that this approach also provides security in our new RRA setting.

Standard model constructions Having dealt with the ROM, we then turn our attention to constructions in the standard model. Reinforcing the connections to RKA security, we are able to show that any Φ-restricted RKA-PRF can be used to build a RRA-ATK secure PKE scheme for Φ-restricted adversaries, thus transferring security from the RKA setting for PRFs to the RRA setting for PKE. But the limited range of RKA-PRFs currently available in the literature [58, 5] essentially restricts the obtained RRA-ATK secure PKE scheme to a class of functions Φ consisting of polynomial functions. In the hope of achieving an RRA-ATK secure PKE scheme for richer classes of functions, we must seek alternative methods of construction.

Unfortunately, we have not been able to achieve our full RRA-ATK security notion for more interesting function classes using other constructions. So we must resort to exploring alternative versions of this notion in order to make progress. We relax RRA-ATK security along two independent dimensions: the degree of control that the adversary enjoys over the public keys under which it

can force encryptions for related random values, and the degree of adaptivity it has in the selection of functions $\phi \in \Phi$:

- We first consider the situation where the public keys are all honestly generated at the start of the security game, and the public keys and all but one of the private keys are then given to the adversary — the honest-key, related randomness attack (HK-RRA) setting. This is a reasonable relaxation in that, in practice, all the public keys that the adversary might be able to induce a user to encrypt under would be properly generated by users and then certified by a CA ahead of time. In this setting, we provide a generic construction for a scheme achieving HK-RRA-ATK security based on combining any IND-ATK secure PKE scheme with a Correlated-Input Secure (CIS) hash function [32]. Currently known instantiations of CIS hash functions allow us to obtain selective, HK-RRA-ATK security for Φ-restricted adversaries where Φ is a large class of polynomial functions. Here, selectivity refers to the adversary committing at the start of the game to the set of functions it will use.

- In the next chapter we will consider the situation where there is no restriction on public keys, but the adversary is committed up-front to a vector of functions $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ that it will use in its attack, and where security is in the end quantified over all choices of $\boldsymbol{\phi}$ from some set $\boldsymbol{\Phi}$. This quantification is subtly different from allowing the adversary a fully adaptive choice of functions $\phi \in \Phi$ (for a detailed discussion, see Sections 5.2 and 6.1). In this situation, we refer to the function-vector, related randomness attack (FV-RRA) model. Here, we are able to give a direct construction for a PKE scheme that is FV-RRA-ATK secure solely under the DDH assumption, assuming the component functions ϕ_i of $\boldsymbol{\phi}$ are simultaneously hard to invert on a random input. Our scheme

is inspired by a PKE scheme of Boneh et al. [17] that is secure in the so-called *auxiliary input setting*, wherein the adversary is given a hard-to-invert function of the secret key as part of its input. By swapping the roles of secret key and randomness in the Boneh et al. scheme, we are able to obtain security in a setting where a hard-to-invert function of the encryption randomness is leaked to the adversary. This leakage is then sufficient to allow us to simulate the encryptions for adversarially chosen public keys. For technical reasons, to obtain a construction, we must also limit our adversary to using the identity function when accessing its **LR** oracle. We then attempt to extend this model by providing definitions that allow an adversary to specify functions when querying its **LR** oracle. We prove that security in this setting can be reduced to the security of a particular type of reconstructive extractor (for a new definition, which we introduce), and we show that the proof technique can be used to provide a transform that converts any IND-ATK-secure scheme into an FV-RRA-ATK-secure scheme.

To summarise, in the standard model, we can achieve our full security notion, RRA-ATK security, but only for a limited class of functions Φ (inherited from known results on RKA-PRFs), whilst our alternative security notions allow us to protect against different classes Φ (in the case of the Function-Vector setting) or provide easier routes to achieve security (in the case of the Honest-Key setting).

5.2 Related Randomness Security for Public-Key Encryption

We now formalise our notions of related randomness security for PKE. We give a detailed treatment of our strongest notion, before sketching restricted

versions. The description of our security notions will utilise code-based games and the associated language (see Section 2.1.3 or [11] for a more thorough treatment).

Our strongest security notion, RRA-CCA security, is defined via the game in Figure 5.3. Here, a challenge key pair (pk^*, sk^*) for a PKE scheme $\mathtt{PKE} = (\mathtt{PKE.K}, \mathtt{PKE.E}, \mathtt{PKE.D})$ with randomness space $\mathtt{Rnd}$ is honestly generated, and the adversary is considered successful if it wins an indistinguishability game with respect to messages encrypted under pk^*. Extending the standard PKE setting, the adversary is able to control which one of polynomially many random values $r_i \in \mathtt{Rnd}$ is used in responding to each encryption query for pk^*; furthermore, the adversary is able to obtain the encryption of messages of its choice under arbitrary (possibly maliciously generated) public keys. Extending the model of Yilek [81], our adversary not only specifies which one of the random values r_i is to be used in each query, but also specifies, for each query he makes, a function ϕ on $\mathtt{Rnd}$; the value $\phi(r_i)$ is used for encryption in place of r_i. In the CCA setting, the adversary also has access to a regular decryption oracle for private key sk^*. Note that if the adversary uses *only* the identity function, then we recover the Resettability Attack (RA) model of Yilek [81].

It is not difficult to see that, as in the RA setting, an adversary may trivially win this game if no restrictions are placed on oracle queries. To see why, consider an adversary that requests the encryption of m under the target public key using coins $\phi(r_i)$ (that is, $\mathtt{PKE.E}(pk^*, m; \phi(r_i))$) and submits **LR** query (m, m', i, ϕ). The adversary guesses b is 0 if the two ciphertexts match, otherwise he guesses b is 1. This adversary wins the game with probability 1. As in the RA setting, such wins are unavoidable in our setting since encryption essentially becomes deterministic when the same random coins and functions ϕ are used. We will shortly introduce an *equality-pattern respecting* definition for adversaries, designed to prevent trivial wins of this kind. This extends

the related RA definition from [81]. However, restrictions on the functions ϕ will also be required. To illustrate the issue, consider as an extreme case the constant function ϕ_C (with $\phi_C(r) = C$ for all $r \in \mathtt{Rnd}$). Suppose the adversary submits **LR** query (m_0, m_1, j, ϕ_C) for any $m_0 \neq m_1$ and any $j \in \mathbb{N}$; the adversary receives a ciphertext c^* and then computes $c_0 = \mathtt{PKE.E}(pk^*, m_0; C)$; the adversary outputs guess $b' = 0$ if and only if $c^* = c_0$. It is easy to see that this adversary wins the RRA-ATK game with probability 1. This example is analogous to one in the related-key attack setting for PRFs in [7]. Hence, we will need to restrict the class of functions which the adversary is allowed to access in its queries to come from some set Φ, in which case we speak of Φ-restricted adversaries. We have already seen that constant functions must be excluded from Φ if we are to have any hope of achieving our related randomness security notion.

Thus we have two sets of constraints that we need to consider to prevent trivial wins: those on messages and randomness indices (analogous to the RA setting from [81]) and those on functions ϕ (analogous to the RKA setting for PRFs from [7]). Let us deal with the first set of constraints first and define what it means for an adversary to be equality-pattern respecting. The following definition is adapted from [81] for our purposes.

Definition 5.2.1. Let $\mathcal{A}$ be a Φ-restricted adversary in Game RRA-ATK that queries r different randomness indices to its **LR** and **Enc** oracles and makes $q_{i,\phi}$ queries to its **LR** oracle with index i and function $\phi \in \Phi$. Let $E_{i,\phi}$ be the set of all messages m such that $\mathcal{A}$ makes **Enc** query (pk^*, m, i, ϕ). Let $(m_0^{i,\phi,1}, m_1^{i,\phi,1}), \ldots, (m_0^{i,\phi,q_{i,\phi}}, m_1^{i,\phi,q_{i,\phi}})$ be $\mathcal{A}$'s **LR** queries for index $i \in [r]$ and $\phi \in \Phi$. Suppose that for all pairs $(i, \phi) \in [r] \times \Phi$ and for all $j \neq k \in [q_{i,\phi}]$, we have:

$$m_0^{i,\phi,j} = m_0^{i,\phi,k} \text{ iff } m_1^{i,\phi,j} = m_1^{i,\phi,k}$$

proc. Initialise(λ):
$b \leftarrow_{\$} \{0,1\}$;
$(pk^*, sk^*) \leftarrow_{\$} \mathtt{PKE.K}(1^\lambda)$;
CoinTab $\leftarrow \emptyset$;
$\mathcal{S} \leftarrow \emptyset$; return pk^*.

proc. Dec(c):
if $c \in \mathcal{S}$
    return $\perp$
else
    return $\mathtt{PKE.D}(sk^*, c)$.

proc. LR(m_0, m_1, i, ϕ):
if CoinTab[i] $=\perp$
    CoinTab[i] $\leftarrow_{\$}$ Rnd
$r_i \leftarrow$ CoinTab[i]
$c \leftarrow \mathtt{PKE.E}(pk^*, m_b; \phi(r_i))$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
return c.

proc. Enc(pk, m, i, ϕ):
if CoinTab[i] $=\perp$
    CoinTab[i] $\leftarrow_{\$}$ Rnd
$r_i \leftarrow$ CoinTab[i]
$c \leftarrow \mathtt{PKE.E}(pk, m; \phi(r_i))$
return c.

proc. Finalise(b'):
if $b = b'$
    return 1
else
    return 0.

Figure 5.3: Game RRA-ATK. (Note that if ATK = CPA, then the adversary's access to **proc. Dec** is removed.)

and that, for all pairs $(i, \phi) \in [r] \times \Phi$, and for all $j \in [q_{i,\phi}]$, we have:

$$m_0^{i,\phi,j} \notin E_{i,\phi} \wedge m_1^{i,\phi,j} \notin E_{i,\phi}.$$

Then we say that $\mathcal{A}$ is *equality-pattern respecting*.

Notice that if the adversary is restricted to using only the identity function, then this definition reduces to the equality-pattern respecting definition for the RA setting, cf. [81, Appendix A].

Definition 5.2.2. We define the advantage of an equality-pattern respecting, RRA-ATK adversary $\mathcal{A}$ against a PKE scheme PKE to be

$$\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1] - 1.$$

A PKE scheme PKE is said to be Φ-RRA-ATK secure if the advantage of any Φ-restricted, equality-pattern respecting, RRA-ATK adversary against PKE that runs in polynomial time is negligible in the security parameter λ.

5.2.1 Alternative security notions

The above definition for Φ-RRA-ATK security is very powerful: it allows an adversary to submit *any* public key to its encryption oracle and allows the adversary to *adaptively* choose the functions ϕ, the only restriction being that they lie in Φ. In Section 5.2.3 we will exhibit conditions that are both necessary and sufficient for achieving security in this sense in the ROM (given a starting PKE scheme that satisfies the usual definition of IND-ATK security). In the standard model, we will give a construction that relies on RKA-PRFs. Since constructions for these are currently limited in terms of the function classes they can handle, we will now consider alternative definitions of the Φ-RRA-ATK notion.

The first alternative notion we consider is called *Honest-Key Related Randomness* (HK-RRA) security. The security game is given in Figure 5.4 and has two parameters, λ and ℓ. Informally, the game itself generates a polynomial number ℓ of key pairs and returns the public keys to the adversary. The adversary then chooses which public key he wishes to be the target key, and is given the private keys corresponding to all the non-target public keys. Meanwhile, the adversary's queries to its **Enc** oracle are restricted to using the public keys generated by the game. Suitable Φ-HK-RRA-ATK security notions follow by analogy with our earlier definitions.

One may consider notions intermediate between Φ-RRA-ATK security and Φ-HK-RRA-ATK security. For example, a registered key notion could be defined, in which the adversary chooses and registers key pairs (pk, sk), with registration involving a test for validity by some procedure, and all queries involve only registered public keys. One may also consider weaker variants of these notions in which the adversary's choice of functions ϕ is non-adaptive (or *selective*). That is, the adversary must submit a set of functions $\{\phi\} \subset \Phi$

proc. Initialise(λ, ℓ):
$b \leftarrow_{\$} \{0,1\}$;
Keys $\leftarrow \emptyset$;
`target` $\leftarrow$ `false`;
CoinTab $\leftarrow \emptyset$;
$\mathcal{S} \leftarrow \emptyset$; $(pk^*, sk^*) \leftarrow \emptyset$
for $i = 1$ to ℓ
  $(pk_i, sk_i) \leftarrow_{\$}$ `PKE.K`(1^λ)
  Keys $\leftarrow$ Keys $\cup\, pk_i$
return Keys.

proc. Target(j):
if `target` $=$ `true`
  return $\perp$
else
  $(pk^*, sk^*) \leftarrow (pk_j, sk_j)$
`target` $\leftarrow$ `true`
return $\{sk_i\}_{i \neq j}$.

proc. Enc(pk, m, i, ϕ):
if `target` $=$ `false`
  return $\perp$
if $pk \notin$ Keys
  return $\perp$
if CoinTab$[i] = \perp$
  CoinTab$[i] \leftarrow_{\$}$ `Rnd`
$r_i \leftarrow$ CoinTab$[i]$
$c \leftarrow$ `PKE.E`$(pk, m; \phi(r_i))$
return c.

proc. Dec(c):
if `target` $=$ `false`
  return $\perp$
if $c \in \mathcal{S}$
  return $\perp$
else
  return `PKE.D`(sk^*, c).

proc. LR(m_0, m_1, i, ϕ):
if `target` $=$ `false`
  return $\perp$
if CoinTab$[i] = \perp$
  CoinTab$[i] \leftarrow_{\$}$ `Rnd`
$r_i \leftarrow$ CoinTab$[i]$
$c \leftarrow$ `PKE.E`$(pk^*, m_b; \phi(r_i))$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
return c.

proc. Finalise(b'):
if $b = b'$
  return 1
else
  return 0.

Figure 5.4: Game ℓ-HK-RRA-ATK. (Note that if ATK = CPA, then the adversary's access to **proc. Dec** is removed.)

of polynomial size to the game before he is allowed to see the target public key (or set of public keys, if playing in the Honest-Key setting). In this setting, we refer to Φ-sHK-RRA-ATK security.

Comparison of security notions The notion of HK-RRA-ATK security is easily seen to be a strictly weaker notion than full RRA-ATK security. A separation can be established by considering a scheme where public keys generated by the key generation algorithm always have a certain bit set to 0, and where the encryption algorithm, given a public key with this bit set to 1 (i.e. a maliciously generated public key), will expose the randomness used for the encryption. Likewise, the selective models are easily seen to be weaker than their adaptive counterparts.

It is not hard to see that our RRA security notions are incomparable with the CDA security notions of [4]. In the RA setting, Yilek defines only an

equivalent of our full RRA-ATK notion; it is clear that RRA-ATK security is stronger than his RA-ATK security whenever the function set Φ contains the identity function. The same would carry over to relaxed versions of RA-ATK security.

5.2.2 A simplifying lemma

Lemma 5.2.1. Consider an equality-pattern respecting, RRA-ATK adversary $\mathcal{A}$ that queries q_r distinct randomness indices and makes at most q_{LR} **LR** queries. Then there exists an equality-pattern respecting, RRA-ATK adversary $\mathcal{B}$ that queries at most one randomness index and makes at most q_{LR} **LR** queries such that

$$\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) \leq q_r \cdot \mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{B}}(\lambda),$$

where $\mathcal{B}$ runs in approximately the same time as $\mathcal{A}$. In the CCA setting, $\mathcal{B}$ makes the same number of decryption queries as $\mathcal{A}$.

Proof. To keep the notation from becoming too cluttered, we denote by $\mathcal{A}^{G(q_{LR},q_r)}$ an adversary $\mathcal{A}$ playing the RRA-ATK game that makes q_{LR} **LR** queries and queries q_r distinct randomness indices. To prove the lemma we will use an alternative, but equivalent, notion of adversarial advantage, namely:

$$\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) = \mathbb{P}[\mathcal{A}^{G(q_{LR},q_r)} \Rightarrow 1 \mid b = 1] - \mathbb{P}[\mathcal{A}^{G(q_{LR},q_r)} \Rightarrow 1 \mid b = 0].$$

Notice that we are now interested in the output of the adversary, rather than the output of the game. Let G_0 denote the RRA-ATK security game in Figure 5.3 where $b = 0$. Let G_{q_r} denote the same game where $b = 1$. If games G_j are as defined in Figure 5.5, then

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) &= \mathbb{P}[\mathcal{A}^{G_{q_r}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_0} \Rightarrow 1] \\
&= \sum_{j=0}^{q_r-1} \mathbb{P}[\mathcal{A}^{G_{j+1}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_j} \Rightarrow 1].
\end{aligned}$$

proc. Initialise(λ):
$(pk^*, sk^*) \leftarrow_{\$} \texttt{PKE.K}(1^\lambda)$;
CoinTab $\leftarrow \emptyset$;
$\mathcal{S} \leftarrow \emptyset$; return pk^*.

proc. Enc(pk, m, i, ϕ):
if CoinTab$[i] = \perp$
  CoinTab$[i] \leftarrow_{\$}$ Rnd
$r_i \leftarrow$ CoinTab$[i]$
$c \leftarrow \texttt{PKE.E}(pk, m; \phi(r_i))$
return c.

proc. LR(m_0, m_1, i, ϕ):
if CoinTab$[i] = \perp$
  CoinTab$[i] \leftarrow_{\$}$ Rnd
$r_i \leftarrow$ CoinTab$[i]$
if $i \leq q_r - j$,
  $c \leftarrow \texttt{PKE.E}(pk^*, m_0; \phi(r_i))$
if $i > q_r - j$,
  $c \leftarrow \texttt{PKE.E}(pk^*, m_1; \phi(r_i))$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
return c.

proc. Dec(c):
if $c \in \mathcal{S}$
  return $\perp$
else
  return $\texttt{PKE.D}(sk^*, c)$.

proc. Finalise(b'):
if $b = b'$
  return 1
else
  return 0.

Figure 5.5: The game G_j used in the proof of Lemma 5.2.1.

Without loss of generality, we will assume that games j^* and $j^* + 1$ have the largest difference. Then,

$$\mathbf{Adv}^{\text{rra-atk}}_{\texttt{PKE},\mathcal{A}}(\lambda) \leq q_r \cdot (\mathbb{P}[\mathcal{A}^{G_{j^*+1}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_{j^*}} \Rightarrow 1]).$$

The only difference between games j^* and $j^* + 1$ is in how the **LR** oracle responds to a query with randomness index $q_r - j^*$. If $\mathcal{A}$ can distinguish between games j^* and $j^* + 1$, then we can use this adversary to build an adversary $\mathcal{B}$ winning the RRA-ATK game and using only 1 randomness index. The **Initialise** procedure for $\mathcal{B}$'s RRA-ATK game is run, returning a target public key pk^* to $\mathcal{B}$. Then adversary $\mathcal{B}$ sets up the simulation for $\mathcal{A}$.

Setup

CoinTab $\leftarrow \emptyset$;

$\mathcal{S} \leftarrow \emptyset$.

Then $\mathcal{B}$ forwards the key pk^* to $\mathcal{A}$. Adversary $\mathcal{B}$ will simulate either game G_{j^*} or G_{j^*+1} for $\mathcal{A}$ by answering $\mathcal{A}$'s oracle queries as follows:

Enc query (pk, m, l, ϕ)

If $l = q_r - j^*$, $\mathcal{B}$ submits (pk, m, ϕ) to its **Enc** oracle, and returns the result to $\mathcal{A}$.

Otherwise, if CoinTab[l] $=\perp$, $\mathcal{B}$ chooses CoinTab[l] $\leftarrow_{\$}$ Rnd,

$r_l \leftarrow$ CoinTab[l], and $\mathcal{B}$ returns $\mathtt{PKE.E}(pk, m; \phi(r_l))$.

LR query (m_0, m_1, l, ϕ)

If $l = q_r - j^*$, $\mathcal{B}$ submits (ϕ, m_0, m_1) to its **LR** oracle and returns the result to $\mathcal{A}$.

Otherwise, if CoinTab[l] $=\perp$, $\mathcal{B}$ chooses CoinTab[l] $\leftarrow_{\$}$ Rnd

$r_l \leftarrow$ CoinTab[l]:

if $l < q_r - j^*$, $\mathcal{B}$ returns $\mathtt{PKE.E}(pk^*, m_0; \phi(r_l))$,

else if $l > q_r - j^*$, $\mathcal{B}$ returns $\mathtt{PKE.E}(pk^*, m_1; \phi(r_l))$.

$\mathcal{B}$ updates $\mathcal{S}$ to include the ciphertext returned to $\mathcal{A}$.

Dec query c

If $c \in \mathcal{S}$, then $\mathcal{B}$ returns $\perp$.

Otherwise $\mathcal{B}$ submits c to its **Dec** oracle and the output is returned to $\mathcal{A}$.

When $\mathcal{A}$ halts with output bit b', $\mathcal{B}$ halts and outputs the same bit b'.
When $b = 0$ (and $\mathcal{B}$ receives an encryption of m_0), it perfectly simulates G_{j^*} for $\mathcal{A}$. When $b = 1$, $\mathcal{B}$ provides a perfect simulation of G_{j^*+1}. It follows that

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) &\leq q_r \cdot (\mathbb{P}[\mathcal{A}^{G_{j^*+1}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_{j^*}} \Rightarrow 1]) \\
&= q_r \cdot (\mathbb{P}[\mathcal{B} \Rightarrow 1 | b = 1] - \mathbb{P}[\mathcal{B} \Rightarrow 1 | b = 0]) \\
&= q_r \cdot \mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{B}}(\lambda).
\end{aligned}$$

This completes the proof. □

The above lemma shows that we need only consider adversaries that use one randomness index. The lemma actually applies for all variations of related randomness security considered above, not just our strongest RRA-ATK notion. This enables us to make a simplifying step at the beginning of all our proofs (at the cost of a q_r factor in all advantages), and to use the following simplified equality-pattern definition in all our proofs:

Definition 5.2.3. Let $\mathcal{A}$ be a Φ-restricted adversary in Game RRA-ATK that queries 1 randomness index (assumed to be $j = 1$) to its **LR** and **Enc** oracles and makes q_ϕ queries to its **LR** oracle with function ϕ. Let E_ϕ be the set of all messages m such that $\mathcal{A}$ makes **Enc** query $(pk^*, m, 1, \phi)$. Let $(m_0^{\phi,1}, m_1^{\phi,1}), \ldots, (m_0^{\phi,q_\phi}, m_1^{\phi,q_\phi})$ be $\mathcal{A}$'s **LR** queries with function ϕ and randomness index 1. Suppose that for all $\phi \in \Phi$ and for all $j \neq k \in [q_\phi]$, we have:

$$m_0^{\phi,j} = m_0^{\phi,k} \text{ iff } m_1^{\phi,j} = m_1^{\phi,k}$$

and that, for all $\phi \in \Phi$ and for all $j \in [q_\phi]$, we have:

$$m_0^{\phi,j} \notin E_\phi \wedge m_1^{\phi,j} \notin E_\phi.$$

Then we say that $\mathcal{A}$ is *equality-pattern respecting.*

Yilek claimed in [81] that a further simplification is possible in his Reset Attack (RA) setting, namely that there is a reduction from any adversary making q **LR** queries to an adversary making just one **LR** query. One might hope that a corresponding result would be possible in our Related Randomness setting. Only a sketch proof is given for the RA setting claim in [81], but it appears to be flawed (see the paragraph below). While we do not have a separation between models with one and $q > 1$ **LR** queries for either Yilek's RA or our RRA setting, neither have we been able to prove the desired simplification from q to one **LR** query.

To see why Yilek's proof is flawed, consider an adversary with one **LR** query, which we will call $\mathcal{A}_1$. This adversary is supposed to simulate the game for $\mathcal{A}_q$, the adversary with q **LR** queries. Adversary $\mathcal{A}_1$ first guesses an index $i \in \{1, \ldots, q\}$. When adversary $\mathcal{A}_q$ makes its ith **LR** query, $\mathcal{A}_1$ passes the query to its own **LR** oracle. For all other queries, $\mathcal{A}_1$ is supposed to pass either m_0 or m_1 to its **Enc** oracle. However, this simulation is not always possible because of the necessary equality-pattern restrictions in the RA setting. For

example, an adversary making two **LR** queries may submit the pairs (m_0, m_1) and (m_1, m_0) to its **LR** oracle, as these are equality-pattern respecting in the model of [81]. Without loss of generality, the simulating adversary $\mathcal{A}_1$ passes the pair (m_0, m_1) to its **LR** oracle, and then, for **LR** query (m_1, m_0), submits either m_0 or m_1 to its **Enc** oracle. However, $\mathcal{A}_1$ would no longer be equality-pattern respecting, even though the original adversary is. Fortunately, the main construction in [81] is still secure against an adversary making multiple **LR** queries.

5.2.3 Function restrictions

Above, we briefly alluded to the fact that the class of functions Φ used by our RRA adversaries must be restricted in various ways. The example given showed that constant functions must always be excluded. Here, we exhibit much stronger necessary conditions on Φ that must be satisfied, namely output-unpredictability and collision-resistance. These notions are closely related to notions with the same names arising in the setting of related-key security for PRFs that was considered in [7]. Here, however, we are concerned with functions acting on the randomness used in PKE schemes rather than on PRF keys.

Definition 5.2.4 (Output-unpredictability for Φ). Let Φ be a set of functions from `Rnd` to `Rnd`. Let α and β be positive integers. Then the (α, β)-output-unpredictability of Φ is defined to be

$$\mathrm{InSec}^{\mathrm{up}}_{\Phi}(\alpha, \beta) = \max_{P \subseteq \Phi, X \subseteq \mathcal{R}, |P| \leq \alpha, |X| \leq \beta} \left\{ \mathbb{P}\left[r \leftarrow_{\$} \mathtt{Rnd} : \{\phi(r) : \phi \in P\} \cap X \neq \emptyset \right] \right\}.$$

Informally, the definition of output-unpredictability measures the (maximum) probability that a function (from a polynomial-sized set P) maps a random input r to a polynomial-sized set X.

Definition 5.2.5 (Collision-resistance for Φ). Let Φ be a set of functions from $\texttt{Rnd}$ to $\texttt{Rnd}$. Let α be a positive integer. Then the α-collision-resistance of Φ is defined to be

$$\mathrm{InSec}^{\mathrm{cr}}_{\Phi}(\alpha) = \max_{P \subseteq \Phi, |P| \leq \alpha} \left\{ \mathbb{P}\left[r \leftarrow_{\$} \texttt{Rnd} : |\{\phi(r) : \phi \in P\}| < |P| \right] \right\}.$$

The previous definition is merely measuring the (maximum) probability that two functions (from a polynomial-sized set P) map the same random r to the same output.

Regarding these two definitions, we have the two following results.

Theorem 5.2.1 (Necessity of output-unpredictability). Let Φ be a class of functions from $\texttt{Rnd}$ to $\texttt{Rnd}$. Suppose there are natural numbers $\alpha = \mathrm{poly}_1(\lambda)$ and $\beta = \mathrm{poly}_2(\lambda)$ such that $\mathrm{InSec}^{\mathrm{up}}_{\Phi}(\alpha, \beta) = p$, where $p := p(\lambda)$ is non-negligible. Then no *PKE* scheme can be RRA-ATK secure with respect to the class of functions Φ.

Proof. By definition, there exists a set $P \subseteq \Phi$ and a set $X \subseteq \texttt{Rnd}$, both of polynomial size, such that

$$\mathbb{P}\left[r \leftarrow_{\$} \texttt{Rnd} : \{\phi(r) : \phi \in P\} \cap X \neq \emptyset \right] = p.$$

We will construct a polynomial-time adversary $\mathcal{A}$ that has advantage at least p against any scheme $\texttt{PKE}$. Let pk^* denote the target public key in the RRA-ATK security game. The adversary, $\mathcal{A}$, chooses two distinct messages m_0 and m_1 and computes $E_0 = \{\texttt{PKE.E}(pk^*, m_0; r) : r \in X\}$ and $E_1 = \{\texttt{PKE.E}(pk^*, m_1; r) : r \in X\}$. Then $\mathcal{A}$ requests **LR** oracle outputs for $(m_0, m_1, 1, \phi)$ for all $\phi \in P$. Let $E_\phi = \{\texttt{PKE.E}(pk^*, m_b; \phi(r_1)) : \phi \in P\}$ denote the set of responses to $\mathcal{A}$'s **LR** queries. If $E_b \cap E_\phi \neq \emptyset$, then $\mathcal{A}$ outputs b, otherwise $\mathcal{A}$ chooses $b \leftarrow_{\$} \{0, 1\}$

and outputs b. Let **pred** denote the event that $E_b \cap E_\phi \neq \emptyset$, then

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) &= 2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1] - 1 \\
&= 2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1 \mid \mathbf{pred}] \cdot \mathbb{P}[\mathbf{pred}] \\
&\quad +2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1 \mid \overline{\mathbf{pred}}] \cdot \mathbb{P}[\overline{\mathbf{pred}}] - 1 \\
&= 2\left(p + \epsilon + \frac{1}{2}(1 - p - \epsilon)\right) - 1 \\
&= p + \epsilon.
\end{aligned}$$

The third line follows because if **pred** occurs then $\mathcal{A}$ wins with probability 1, whilst if **pred** does not occur then $\mathcal{A}$ guesses and hence wins with probability $1/2$. If any of the functions ϕ maps into X, then there will be a collision in E_X and E_b for some b, hence event **pred** will definitely occur. Therefore $\mathbb{P}[\mathbf{pred}] \geq p$. Let $\mathbb{P}[\mathbf{pred}] = p + \epsilon$ for some $\epsilon \geq 0$. The result easily follows. No **Dec** queries are required for this attack, so it applies to both the CPA and CCA settings. □

Theorem 5.2.2 (Necessity of collision-resistance). Let Φ be a class of functions from `Rnd` to `Rnd`. Suppose there is a natural number $\alpha = \text{poly}_1(\lambda)$ such that $\text{InSec}^{\text{cr}}_{\Phi}(\alpha) = p$, where $p := p(\lambda)$ is non-negligible. Then no *PKE* scheme can be RRA-ATK secure with respect to the class of functions Φ.

Proof. By definition, there exists a subset $P \subseteq \Phi$ of polynomial size such that

$$\mathbb{P}\left[r \leftarrow_{\$} \mathtt{Rnd} : |\{\phi(r) : \phi \in P\}| < |P|\right] = p.$$

We construct a polynomial-time adversary $\mathcal{A}$ that has advantage at least $p/2$. Let the target public key be pk^*. Then $\mathcal{A}$ chooses $|P| + 1$ distinct messages $m_0, m_1, \ldots, m_{|P|}$ and assigns an index to each $\phi \in P$. For i from 1 to $|P|$, $\mathcal{A}$ requests **LR** oracle output for query $(m_0, m_i, 1, \phi_i)$. Let the output of the **LR** oracle for the ith query be c_i. Let $\mathbf{coll}_r$ denote the event that $\phi_i(r_1) = \phi_j(r_1)$ for some $i \neq j$, where r_1 is the randomness chosen by the game for

index 1. Let $\mathbf{coll}_c$ denote that two cipher texts output by the **LR** oracle collide. Adversary $\mathcal{A}$ outputs $b = 0$ if $c_i = c_j$ for some $i \neq j$. Otherwise, $\mathcal{A}$ chooses $b \leftarrow_{\$} \{0,1\}$ and outputs b. Then,

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) &= 2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1] - 1 \\
&= 2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1 \mid \mathbf{coll}_c] \cdot \mathbb{P}[\mathbf{coll}_c] \\
&\quad +2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1 \mid \overline{\mathbf{coll}_c}] \cdot \mathbb{P}[\overline{\mathbf{coll}_c}] - 1 \\
&= 2\left(\frac{1}{2}p + \epsilon + \frac{1}{2}(1 - \frac{1}{2}p - \epsilon)\right) - 1 \\
&= \frac{1}{2}p + \epsilon.
\end{aligned}$$

This follows because $\mathbf{coll}_c$ occurs whenever $b = 0$ and $\mathbf{coll}_r$ occurs. Hence, $\mathbb{P}[\mathbf{coll}_c] \geq \mathbb{P}[b = 0 \cap \mathbf{coll}_r] = p/2$. If we let $\mathbb{P}[\mathbf{coll}_c] = p/2 + \epsilon$ for some $\epsilon \geq 0$, then the claim follows easily since $\mathcal{A}$ wins with probability 1 if $\mathbf{coll}_c$ occurs, whilst if $\mathbf{coll}_c$ does not occur then $\mathcal{A}$ guesses and hence wins with probability $1/2$. No **Dec** queries are required for this attack, so this argument applies equally well to both the CPA and CCA settings. □

These restrictions might seem to rule out the possibility of defending against some of the more interesting classes of functions. However, this is not the case. We note that many classes of functions that arise from practical attacks satisfy the output-unprecdictability and collision-resistance conditions. For example, the class of functions that flip bits at certain positions, or the class of functions that fix the value of certain bits, are both output-unpredictable and collision-resistant (provided at least a polynomial number of bits are not fixed, in the latter case).

Alg. $\texttt{Hash-PKE.K}(1^\lambda)$:	Alg. $\texttt{Hash-PKE.E}(pk, m)$:	Alg. $\texttt{Hash-PKE.D}(sk, c)$:
$(pk, sk) \leftarrow_\$ \texttt{PKE.K}(1^\lambda)$.	$r \leftarrow_\$ \texttt{Rnd}$ $c \leftarrow \texttt{PKE.E}(pk, m; H(pk\|\|m\|\|r))$ return c.	$m \leftarrow \texttt{PKE.D}(sk, m)$ return m.

Figure 5.6: Scheme Hash-PKE built from a PKE scheme PKE and a hash function H.

5.3 Construction in the Random Oracle Model

We have seen that the class of functions Φ must be collision-resistant and output-unpredictable in order for a scheme to be secure against related randomness attacks. In the ROM, these two conditions are in fact also *sufficient* to ensure security in our *strongest* RRA-ATK models, in the following sense: given a hash function H, any PKE scheme PKE that is IND-ATK secure, and a set of functions Φ that is both collision-resistant and output-unpredictable, the scheme Hash-PKE constructed from PKE as in Figure 5.6 is Φ-RRA-ATK secure in the ROM. The next result formalises this claim.

Theorem 5.3.1. Suppose $\mathcal{A}$ is a Φ-restricted, equality-pattern respecting adversary in the RRA-ATK game against the scheme Hash-PKE defined in Figure 5.6. Suppose $\mathcal{A}$ requests encryptions for q_ϕ distinct functions, queries q_r randomness indices, and makes q_{RO} random oracle queries. Then there exists an IND-ATK adversary $\mathcal{C}$ against PKE such that

$$\begin{aligned}\mathbf{Adv}^{\text{rra-atk}}_{\texttt{Hash-PKE},\mathcal{A}}(\lambda) \;\leq\; & q_r \cdot q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda) + 2q_r \cdot \text{InSec}^{\Phi}_{\text{cr}}(q_\phi) \\ & + 2q_r \cdot \text{InSec}^{\Phi}_{\text{up}}(q_\phi, q_{RO}).\end{aligned}$$

Adversary $\mathcal{C}$'s running time is approximately the same as that of $\mathcal{A}$. In the CCA game, $\mathcal{C}$ makes the same number of decryption queries as $\mathcal{A}$.

Proof. First, we invoke Lemma 5.2.1, so that we now only have to prove the theorem for an adversary using just one randomness value, which we assume

to be r^*. Without loss of generality, we assume that queries to the random oracle take the form $pk||m||r$, where pk is a public key, m is a message and r is a randomness value. Let **pred** denote the event that $\mathcal{A}$ makes a query $X = pk||m||r$ to the random oracle and this value X is used by the **Enc** or **LR** oracle as an input to the random oracle to encrypt a message m with randomness $\phi(r^*)$ under public key pk. Let **coll** denote the event that $\mathcal{A}$ queries distinct functions ϕ_1 and ϕ_2 such that $\phi_1(r^*) = \phi_2(r^*)$. Then

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\texttt{Hash-PKE},\mathcal{A}}(\lambda) &= 2 \cdot \mathbb{P}[\text{RRA-RO-ATK}^{\mathcal{A}}_{\texttt{Hash-PKE}} \Rightarrow\ 1] - 1 \\
&\leq 2 \cdot \mathbb{P}[\text{RRA-RO-ATK}^{\mathcal{A}}_{\texttt{Hash-PKE}} \Rightarrow 1 \mid \overline{\mathbf{coll} \cup \mathbf{pred}}] \\
&\quad + 2 \cdot \mathbb{P}[\mathbf{coll} \cup \mathbf{pred}] - 1 \\
&\leq 2 \cdot \mathbb{P}[\text{IND-ATK}^{\mathcal{B}}_{\texttt{PKE}} \Rightarrow 1] - 1 + 2 \cdot \mathbb{P}[\mathbf{coll}] + 2 \cdot \mathbb{P}[\mathbf{pred}] \\
&\leq \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{B}}(\lambda) + 2 \cdot \text{InSec}^{\Phi}_{\text{cr}}(q_\phi) + 2 \cdot \text{InSec}^{\Phi}_{\text{up}}(q_\phi, q_{RO}) \\
&\leq q_{LR}\mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda) + 2\text{InSec}^{\Phi}_{\text{cr}}(q_\phi) + 2\text{InSec}^{\Phi}_{\text{up}}(q_\phi, q_{RO}).
\end{aligned}$$

The third line follows because if neither **coll** nor **pred** occurs, then the inputs to the random oracle are distinct and unknown, so the outputs may be replaced with random values chosen independently and uniformly at random. Hence, a standard IND-ATK adversary $\mathcal{B}$ can simulate this game for $\mathcal{A}$. For $\mathcal{A}$'s **Enc** queries, $\mathcal{B}$ chooses a fresh random value and uses this to encrypt. For **LR** queries, $\mathcal{B}$ forwards the message pair to his own **LR** oracle. **Dec** queries are forwarded to $\mathcal{B}$'s **Dec** oracle and $\mathcal{B}$ returns a random value for random oracle queries. From $\mathcal{A}$'s perspective, $\mathcal{B}$ provides a perfect simulation. When $\mathcal{A}$ outputs a bit, $\mathcal{B}$ outputs the same bit. The final line comes from a straightforward hybrid argument. $\mathcal{B}$ is allowed multiple **LR** queries, but this may be simulated by a standard IND adversary that is allowed only one **LR** query, with a security loss. The simulator $\mathcal{C}$ guesses an index $j \in \{1 \ldots, q_{LR}\}$ and forwards the target public key. For $\mathcal{B}$'s ith query, if $i < j$ (resp. $i > j$) $\mathcal{C}$ encrypts m_0 (resp. m_1) with fresh randomness and returns the ciphertext to $\mathcal{B}$.

If $i = j$, $\mathcal{C}$ forwards (m_0, m_1) to his own **LR** oracle and forwards the output to $\mathcal{B}$. At the end of the simulation $\mathcal{C}$ outputs the same bit as $\mathcal{B}$. This completes the proof. □

Bellare et al. [4] introduced the *randomised-encrypt-with-hash* (REwH) method to protect against randomness failures in public-key encryption. This method amounts to incorporating as much context as possible via hashing when setting up randomness. It was further developed in [68] as a general purpose technique applicable to multiple cryptographic primitives. The construction in Figure 5.6 can be seen as an instance of this method, in that the 'random value' used during encryption is replaced with a hash of the public key, the message to be encrypted, and the actual random value. Theorem 5.3.1 then shows that hedging in this way not only protects against the various forms of randomness failure considered in [4, 68], but also protects against failures in our related randomness setting, to the maximum extent possible.

Now that we have shown a necessary and sufficient condition for RRA security in the ROM, the challenge is to extend our results to the standard model. The remainder of this chapter is concerned with achieving this goal.

5.4 Related Randomness Security for PKE from RKA-PRFs

Since the RA setting of Yilek [81] is a special case of our RRA setting, an obvious way to try to achieve RRA security is to extend the main construction from [81]. Recall that the construction of Yilek combines a PRF with an IND-ATK secure PKE scheme. Specifically, the randomness r is used as a key for the PRF, and the input to the PRF is the 'context' $pk||m$; the output from the PRF is then used as the actual randomness for encryption. This construction

Alg. $\texttt{PRF-PKE.K}(1^\lambda)$:	Alg. $\texttt{PRF-PKE.E}(pk, m)$:	Alg. $\texttt{PRF-PKE.D}(sk, c)$:
$(pk, sk) \leftarrow_\$ \texttt{PKE.K}(1^\lambda)$.	$r \leftarrow_\$ \texttt{Rnd}$ $r' \leftarrow F_r(pk\|\|m)$ $c \leftarrow \texttt{PKE.E}(pk, m; r')$ return c.	$m \leftarrow \texttt{PKE.D}(sk, c)$ return m.

Figure 5.7: Scheme PRF-PKE built from a standard PKE scheme, PKE and a PRF, F.

extends directly to our setting, and security is guaranteed against Φ-restricted adversaries in our strongest RRA-ATK models, under the assumption that the PRF is Φ-RKA-secure (that is, secure against related-key attacks for the *same* class of functions Φ). Thus the construction transfers RKA security for PRFs to RRA-ATK security for PKE. Figure 5.7 formalises the construction, and Theorem 5.4.1 our security result. Notice that our RO scheme in Section 5.3 may be interpreted as an instantiation of the scheme in Figure 5.7, since a random oracle can be viewed as an (unkeyed) RKA-PRF.

Theorem 5.4.1. Suppose $\mathcal{A}$ is a Φ-restricted, equality-pattern respecting adversary in the RRA-ATK game against the scheme PRF-PKE defined in Figure 5.7. Suppose $\mathcal{A}$ makes q_{LR} **LR** queries, q_s **Enc** queries, and uses q_r randomness indices. Then there exists a Φ-restricted RKA-PRF adversary $\mathcal{B}$ and an IND-ATK adversary $\mathcal{C}$ such that

$$\mathbf{Adv}^{\text{rra-atk}}_{\texttt{PRF-PKE},\mathcal{A}}(\lambda) \leq q_{LR} \cdot q_r \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda) + 2q_r \cdot \mathbf{Adv}^{\text{rka-prf}}_{F,\mathcal{B}}(\lambda).$$

Adversaries $\mathcal{B}$ and $\mathcal{C}$ run in approximately the same time as $\mathcal{A}$. Adversary $\mathcal{C}$ makes one **LR** query and the same number of **Dec** queries as $\mathcal{A}$. Adversary $\mathcal{B}$ makes at most $q_{LR} + s$ queries to its oracle.

Proof. We first apply Lemma 5.2.1, so that we may concentrate on an adversary using just one randomness value. Let G_0 be the real RRA-ATK security game played by an adversary $\mathcal{A}$ against the scheme PRF-PKE and let G_1 be the game where outputs of the PRF F are replaced with values chosen uniformly

at random. That is, in G_1, each encryption uses fresh random coins rather than using outputs from F. We first claim that there is an adversary $\mathcal{B}$ against the Φ-RKA-PRF security of F such that

$$\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}^{\text{rka-prf}}_{F,\mathcal{B}}(\lambda).$$

Our construction of Adversary $\mathcal{B}$ is as follows. Adversary $\mathcal{B}$ flips a bit b and generates a key pair (pk^*, sk^*). Then, $\mathcal{B}$ gives pk^* to $\mathcal{A}$ and runs $\mathcal{A}$. When $\mathcal{A}$ submits an **Enc** query $(pk, m, 1, \phi)$, $\mathcal{B}$ sends $(\phi, pk||m)$ to its oracle, uses the output, r' to encrypt, so that $c \leftarrow \texttt{PKE.E}(pk, m; r')$, and returns c to $\mathcal{A}$. When $\mathcal{A}$ submits an **LR** query $(m_0, m_1, 1, \phi)$, $\mathcal{B}$ sends $(\phi, pk^*||m_b)$ to its oracle and uses the output, r', to encrypt, setting $c \leftarrow \texttt{PKE.E}(pk^*, m_b; r')$ and returning c to $\mathcal{A}$. In the CCA game, $\mathcal{B}$ generated sk^* so he may use this to respond to any decryption queries. At the end of the simulation, when $\mathcal{A}$ outputs a bit b', $\mathcal{B}$ outputs 1 if and only if $b = b'$. If $\mathcal{B}$ is in the real world, he simulates G_0, otherwise he simulates G_1.

Now adversary $\mathcal{A}$'s queries in game G_1 can be simulated by an IND-ATK adversary $\mathcal{C}$ against PKE as follows. The adversary $\mathcal{C}$ must guess an index $j \in \{1, \ldots, q_{LR}\}$. For the ith **LR** query made by $\mathcal{A}$, when $i < j$ (resp. $i > j$) $\mathcal{C}$ responds with an encryption of m_0 (resp. m_1). For the jth **LR** query made by $\mathcal{A}$, $\mathcal{C}$ forwards the query to his own **LR** oracle and returns the output. At the end of the simulation $\mathcal{C}$ outputs the same bit as $\mathcal{A}$ playing G_1. A standard hybrid argument shows that

$$2 \cdot \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - 1 \leq q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda).$$

Hence,

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\texttt{PRF-PKE},\mathcal{A}}(\lambda) &= 2 \cdot \mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - 1 \\
&= 2(\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1]) + 2 \cdot \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - 1 \\
&\leq 2 \cdot \mathbf{Adv}^{\text{rka-prf}}_{F,\mathcal{B}}(\lambda) + q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda).
\end{aligned}$$

Combining with the randomness reduction lemma gives the desired result. □

The previous theorem is seductively simple, but currently of limited application because the set of known RKA-secure PRFs is rather sparse. RKA-PRFs were first formalised in 2003 by Bellare and Kohno [7], and some initial (though not fully satisfactory) constructions were given in [7] and [58]. Setting these aside, the only known constructions are due to Bellare and Cash [5] and Abdalla et al. [1]. Bellare et al. gave a first construction for an RKA-PRF (based on the Naor-Reingold PRF) that is provably secure under the DDH assumption for related key functions Φ corresponding to component-wise multiplication on the key-space $(\mathbb{Z}_p^*)^{n+1}$. They also provided a second construction achieving a similar result under the DLIN assumption. A third construction for related key functions Φ corresponding to component-wise *addition* on the key-space $(\mathbb{Z}_p)^n$ was recently withdrawn by the authors of [5]. However, the recent work of Abdalla et al. [1] has since repaired this withdrawn construction. In addition, by employing a slightly stronger assumption (the Decisional Diffie-Hellman Inversion assumption) the authors are able to avoid the exponential-time security reduction that was inherent in the (flawed) proof of [5]. Moreover, using this stronger Decisional Diffie-Hellman Inversion assumption, Abdalla et al. were able to provide an RKA-PRF that is secure against bounded-polynomial RKD functions that act component-wise on the key space. For their particular construction, the key space is $\mathbb{Z}_p^n$ and the RKD functions ϕ are a collection of functions, so $\phi = (\phi_1, \ldots, \phi_n)$, with each ϕ_i acting on component i of the key space. Additionally their RKD functions have the constraints that no ϕ_i can be a constant function, and all ϕ_i must be polynomials in one variable of bounded degree.

The limited nature of existing RKA-PRF families forces us to find alternative approaches to achieving security in the RRA setting. The application

for RKA-PRFs implied by Theorem 5.4.1 also provides yet more motivation for the fundamental problem of constructing RKA-PRFs for richer classes of related key function.

5.5 Related Randomness PKE from CIS Hash Functions

To address some of the limitations encountered in the previous approach, we show how a PKE scheme secure in the RRA setting can be constructed using correlated-input secure (CIS) hash functions as introduced in [32]. Whilst the currently known instantiations of CIS hash functions only allow us to obtain selective HK-RRA-ATK security for a class of polynomial functions, this avenue is still worth exploring. Any advances in the state-of-the-art concerning CIS hash functions will be immediately applicable to the theorem we prove in this section, resulting in improvements to results concerning related randomness attacks. Furthermore, there is evidence to suggest that achieving CIS hash security is easier than obtaining RKA-PRF security. As we will see shortly, any CIS hash can be used to construct an RKA-secure *weak* PRF. Therefore, it is likely in future that the scheme presented in this chapter may protect against larger classes of functions than our RKA-PRF construction (although this is not currently the case).

Before proceeding, it is worth comparing the classes of polynomial functions that our RKA-PRF and CIS hash constructions protect against. Current instantiations of CIS hash functions are secure against selective, uniform-output, bounded-degree polynomials. In comparison, as mentioned in the previous section, there are RKA-PRFs that are secure against adaptive, non-constant, bounded-degree polynomials. Hence, the RKA-PRF construction

enjoys two advantages over the concrete instantiation we will see in this chapter: adaptivity of function selection, and a relaxed requirement on the output distribution of the polynomials (they need not be uniform-output). However, for the reasons outlined above, we study our CIS hash construction in the hope that stronger results concerning CIS hashes will be available in future.

In its strongest form, a CIS hash function h (with key k, sampled by the key generation algorithm `CI-HASH.K`) will yield output $h_k(x)$ which is still pseudorandom, even when given the hash value of multiple correlated input values $(h_k(\phi_1(x)), \ldots, h_k(\phi_q(x)))$, where the correlation functions $\phi_1, \ldots, \phi_q$ are maliciously chosen. This type of CIS hash function is closely related to RKA-secure PRFs. In fact, the authors of [32] show that given a CIS hash function h, an RKA-secure *weak* PRF F can be obtained simply by exchanging the role of the key and the input of h:

$$F_K(x) := h_x(K).$$

Recall that weak PRF security does not allow an adversary to choose the function inputs, but instead, the inputs are chosen uniformly at random in the security game.

The authors of [32] furthermore give a concrete construction of a CIS hash function secure for a class of correlation functions consisting of uniform-output polynomials of bounded degree, by which uniform-output means its output range is equal to its domain (that is, evaluating the polynomial on all values in the domain will again yield the elements of the domain), albeit in a restricted security model where the adversary's function queries are non-adaptive. This then yields a non-adaptive, RKA-secure weak PRF.

Unfortunately, such a PRF this is not sufficient for our purposes. Surprisingly, however, by making a relatively simple modification to the above

proc. Initialise(λ, ℓ):
$(\mathcal{K}, \mathcal{D}, \mathcal{R}, h) \leftarrow \texttt{GenFun}(1^\lambda)$
For $i = 1$ to ℓ
  $k_i \leftarrow_\$ \mathcal{K} \setminus \{k_{i-1}, \ldots, k_1\}$
$x \leftarrow_\$ \mathcal{D}$
$b \leftarrow_\$ \{0, 1\}$
$\mathcal{S} \leftarrow \emptyset$
$\texttt{func} \leftarrow \texttt{false}$
$\texttt{chal} \leftarrow \texttt{false}$
return $(\mathcal{K}, \mathcal{D}, \mathcal{R}, h)$.

proc. Functions$(\phi_1, \ldots, \phi_q)$:
if $\texttt{func} = \texttt{true}$
  return $\perp$
$\texttt{func} \leftarrow \texttt{true}$
return $k_1, \ldots, k_\ell$.

proc. Hash(i, j):
if $\texttt{func} = \texttt{false}$
  return $\perp$
if $\texttt{chal} = \texttt{true}$,
  return $\perp$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(i, j)\}$
return $h_{k_i}(\phi_j(x))$.

proc. Chal(i^*, j^*):
if $\texttt{func} = \texttt{false}$
  return $\perp$
if $\texttt{chal} = \texttt{true}$,
  return $\perp$
if $(i^*, j^*) \in \mathcal{S}$,
  return $\perp$
$y_0 \leftarrow_\$ \mathcal{R}$
$y_1 \leftarrow h_{k_{i^*}}(\phi_{j^*}(x))$
$\texttt{chal} \leftarrow \texttt{true}$
return y_b.

proc. Finalise(b'):
if $b = b'$, return 1.

Figure 5.8: Game ℓ-MK-SCI-PR for a family $\mathcal{H}$ of keyed hash functions defined by `GenFun`.

construction of PRFs from CIS hash functions, it is possible to obtain a primitive similar to an RKA-secure (standard) PRF. More specifically, consider a CIS hash function h and a standard PRF f. We introduce a public parameter c of F that will correspond to the key for h, and then, instead of using the output of h directly, we use h to derive a key for f. More specifically, we define

$$F_{c,K}(x) := f_{h_c(K)}(x).$$

Whilst not strictly an RKA-secure PRF due to the presence of the public parameter c, this primitive allows adaptively chosen inputs x, while remaining secure under related-key attacks. This 'partial' RKA-secure PRF will allow us to obtain HK-RRA-ATK secure encryption schemes for the function families of the underlying CIS hash function h. However, to achieve this, we need to extend the definitions and theorems of [32] to the multi-key setting (reflecting the fact that, in the HK-RRA setting, the adversary can interact with multiple public keys).

We formally define *multi-key selective correlated-input pseudorandomness* (MK-SCI-PR) for a family of keyed hash functions via the security game shown

Alg. $\texttt{CI-Hash-PKE.K}(1^\lambda)$:	Alg. $\texttt{CI-Hash-PKE.E}(\hat{pk}, m)$:	Alg. $\texttt{CI-Hash-PKE.D}(\hat{sk}, c)$:
$(pk, sk) \leftarrow_\$ \texttt{PKE.K}(1^\lambda)$	$(pk\|\|k) \leftarrow \hat{pk}$	$m \leftarrow \texttt{PKE.D}(\hat{sk}, c)$
$k \leftarrow_\$ \texttt{CI-HASH.K}(1^\lambda)$	$r \leftarrow_\$ \texttt{Rnd}$	return m.
$(\hat{pk}, \hat{sk}) \leftarrow (pk\|\|k, sk)$.	$r' \leftarrow h_k(r)$	
	$r'' \leftarrow F_{r'}(\hat{pk}\|\|m)$	
	$c \leftarrow \texttt{PKE.E}(pk, m; r'')$	
	return c.	

Figure 5.9: Scheme CI-Hash-PKE built from PKE scheme PKE, PRF F, and hash function family $\mathcal{H}$.

in Figure 5.8. The definition is selective in the sense that the adversary is required to submit the correlation functions before seeing the hash function keys used in the game. As in the definition of related randomness security, we consider Φ-restricted adversaries, which are adversaries who are restricted to submit correlation functions belonging to a given class of functions Φ.

Definition 5.5.1. A family $\mathcal{H}$ of keyed hash functions is said to be (Φ, ℓ)-MK-SCI-PR secure if for all Φ-restricted adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ against $\mathcal{H}$, defined as

$$\mathbf{Adv}^{\ell\text{-mk-sci-pr}}_{\mathcal{H},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\ell\text{-MK-SCI-PR}^{\mathcal{A}}_{\mathcal{H}}(\lambda) \Rightarrow 1] - 1,$$

is negligible in the security parameter λ.

Based on an ordinary PKE scheme PKE, a PRF F, and a family of hash functions $\mathcal{H}$, we construct a PKE scheme CI-Hash-PKE as shown in Figure 5.9. The following theorem establishes the selective ℓ-HK-RRA-ATK security of this scheme based on the IND-ATK security of PKE, the multi-key selective CIS security of $\mathcal{H}$, and the (regular) pseudorandomness of F.

Theorem 5.5.1. Suppose $\mathcal{A}$ is a Φ-restricted, equality pattern respecting adversary in the selective ℓ-HK-RRA-ATK game against the scheme CI-Hash-PKE in Figure 5.9. Suppose $\mathcal{A}$ makes q_{LR} **LR** queries, uses q_r randomness indices, and uses q_ϕ functions in its oracle queries. Then there exists a Φ-restricted, multi-key, selective correlated-input secure hash adversary $\mathcal{B}$, a *PRF* adversary $\mathcal{C}$

and an *IND-ATK* adversary $\mathcal{D}$ such that

$$\begin{aligned}\mathbf{Adv}^{\ell\text{-shk-rra-atk}}_{\texttt{CI-Hash-PKE},\mathcal{A}}(\lambda) \;\leq\; & 2q_\phi \cdot q_r \cdot \mathbf{Adv}^{\ell\text{-mk-sci-pr}}_{\mathcal{H},\mathcal{B}}(\lambda) + 2q_\phi \cdot q_r \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{C}}(\lambda) \\ & + \ell \cdot q_{LR} \cdot q_r \cdot \mathbf{Adv}^{\text{ind-atk}}_{\text{PKE},\mathcal{D}}(\lambda) + \frac{\ell^2 \cdot q_r}{|\text{HashKeySpace}|}.\end{aligned}$$

Adversaries $\mathcal{B}$, $\mathcal{C}$ and $\mathcal{D}$ run in approximately the same time as $\mathcal{A}$. Adversary $\mathcal{C}$ makes at most q_{LR} queries, and $\mathcal{D}$ makes one **LR** query and as many **Dec** queries as $\mathcal{A}$.

Proof. First, we invoke Lemma 5.2.1, so that we now only have to prove the theorem for an adversary using just one randomness value. We prove the theorem via a sequence of game hops and hybrid arguments. Let G_0 be the real correlated-input secure hash PKE game, which is the non-adaptive version of the game in Figure 5.4. The game G_1 is the same except for queries on the target key. Rather than using the outputs of the hash function, the game picks uniformly random values and uses these values as keys for the PRF.

$$\begin{aligned}\mathbf{Adv}^{\ell\text{-hk-rra-atk}}_{\texttt{CI-Hash-PKE},\mathcal{A}}(\lambda) &= 2 \cdot \mathbb{P}[G_0^{\mathcal{A}}] - 1 \\ &\leq 2 \cdot (\mathbb{P}[G_0^{\mathcal{A}} \mid \overline{\mathbf{coll}}] + \mathbb{P}[\mathbf{coll}]) - 1 \\ &= 2 \cdot (\mathbb{P}[G_0^{\mathcal{A}} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G_1^{\mathcal{A}} \mid \overline{\mathbf{coll}}]) + 2 \cdot \mathbb{P}[\mathbf{coll}] - 1 \\ &\quad + 2 \cdot \mathbb{P}[G_1^{\mathcal{A}} \mid \overline{\mathbf{coll}}].\end{aligned}$$

If there are no collisions in the hash function keys, then the difference between G_0 and G_1 is negligible. This can be stated formally as:

Lemma 5.5.1. The difference between the success probability of $\mathcal{A}$ in games G_0 and G_1 is bounded by the advantage of a multi-key, CIS hash adversary $\mathcal{B}$. That is:

$$\mathbb{P}[G_0^{\mathcal{A}} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G_1^{\mathcal{A}} \mid \overline{\mathbf{coll}}] \leq \mathbf{Adv}^{\ell\text{-mk-sci-pr}}_{\mathcal{H},\mathcal{B}}(\lambda).$$

Proof. We will prove this via a hybrid argument. Let $G_{0,i}$ denote the game in which, for the target public key, for function $k \leq q_\phi - i$ the output of the hash function is real, whereas for function $k > q_\phi - i$ the output is random, rather than using the hash function. Notice that $G_0 = G_{0,0}$ and $G_1 = G_{0,q_\phi}$, so

$$\begin{aligned}\mathbb{P}[G_0^{\mathcal{A}} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G_1^{\mathcal{A}} \mid \overline{\mathbf{coll}}] &= \sum_{i=0}^{q_\phi - 1} \mathbb{P}[G_{0,i}^{\mathcal{A}} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G_{0,i+1}^{\mathcal{A}} \mid \overline{\mathbf{coll}}] \\ &\leq q_\phi \cdot (\mathbb{P}[G_{0,j^*}^{\mathcal{A}} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G_{0,j^*+1}^{\mathcal{A}} \mid \overline{\mathbf{coll}}]),\end{aligned}$$

where we have assumed (without loss of generality) that hybrid games j^* and $j^* + 1$ have the largest difference. If an adversary can distinguish games j^* and $j^* + 1$ when there are no collisions of hash keys, then we may use this adversary to construct a multi-key CIS hash adversary that distinguishes with the same probability. The CIS hash adversary $\mathcal{B}$ will simulate either the game G_{0,j^*} or G_{0,j^*+1} for $\mathcal{A}$.

Adversary $\mathcal{B}$ initiates the CIS hash game and is given the description of a hash function. Then, $\mathcal{B}$ sets up the game for $\mathcal{A}$.

Setup

$b \leftarrow_\$ \{0, 1\}$

`funchoice` $\leftarrow$ `false`; `target` $\leftarrow$ `false`

Keys $\leftarrow \emptyset$; Functions $\leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$; $(p\hat{k}^*, s\hat{k}^*) \leftarrow \emptyset$.

When $\mathcal{B}$ has finished this setup procedure, he forwards the description of the hash function to $\mathcal{A}$. Adversary $\mathcal{B}$ will then answer $\mathcal{A}$'s **Func** and **Target** queries as follows:

Func query $(\phi_1, \ldots, \phi_{q_\phi})$

if `functions` = `true`, $\mathcal{B}$ returns $\perp$

otherwise $\mathcal{B}$ forwards the query to his challenger and receives $k_1, \ldots, k_\ell$

for $i = 1$ to ℓ, $\mathcal{B}$ generates $(pk_i, sk_i) \leftarrow_\$ \mathtt{PKE.K}(1^\lambda)$

$\mathcal{B}$ sets $p\hat{k}_i = pk_i || k_i$, and $s\hat{k}_i = sk_i$

$\texttt{functions} \leftarrow \texttt{true}$

$\mathcal{B}$ returns $\{\hat{pk}_i\}$ to $\mathcal{A}$.

Target query (j)

if $\texttt{target} = \texttt{true}$, $\mathcal{B}$ returns $\perp$

otherwise $\mathcal{B}$ sets $(\hat{pk}^*, \hat{sk}^*) \leftarrow (\hat{pk}_j, \hat{sk}_j)$

$\texttt{target} \leftarrow \texttt{true}$

for $i \neq j$, $\mathcal{B}$ returns $\{\hat{sk}_i\}$.

Adversary $\mathcal{A}$ is denied access to the other oracles until `target`=`true` and `functions`=`true`. When `target`=`true` and `functions`=`true`, $\mathcal{B}$ submits (i, κ) to his CIS hash oracle for all $(i, \kappa) \neq (j, j^*)$ Then $\mathcal{B}$ submits (j, j^*) to his CIS challenge oracle. $\mathcal{B}$ keeps a table of inputs and outputs to the CIS oracle. $\mathcal{B}$'s challenge oracle will return a value r, which is either the real output for $h_{k_j}(\phi_{q_\phi - j^*}(r))$ or a uniformly random value. This value r should be stored alongside $(k_j, q_\phi - j^*)$ in the table. For input pairs (k_j, κ), where $\kappa > q_\phi - j^*$, $\mathcal{B}$ replaces the oracle outputs in the table with uniformly random values. N.B. Adversary $\mathcal{B}$ could, and would in practice, generate this table 'on-the-fly' in response to $\mathcal{A}$'s **Enc** and **LR** queries. However, to keep our presentation clear and uncluttered, we adopt the current approach of generating the whole table before answering $\mathcal{A}$'s **Enc** or **LR** queries.

Enc query $(\hat{pk}, m, 1, \kappa)$

if $\texttt{target} = \texttt{false}$ or $\texttt{functions} = \texttt{false}$, return $\perp$

$\mathcal{B}$ parses $\hat{pk} = pk_i || k_i$

$\mathcal{B}$ finds $(k_i, \kappa, r_{i,\kappa})$ in the table and returns $\texttt{PKE.E}(pk, m; F_{r_{i,\kappa}}(\hat{pk}||m))$.

LR query $(m_0, m_1, 1, \kappa)$

if $\texttt{target} = \texttt{false}$ or $\texttt{functions} = \texttt{false}$, return $\perp$

$\mathcal{B}$ finds (k_j, κ) in the table and gets corresponding value $r_{j,\kappa}$

$\mathcal{B}$ returns $\texttt{PKE.E}(pk^*, m_b; F_{r_{j,\kappa}}(\hat{pk}^*||m_b))$

$\mathcal{B}$ adds the ciphertext to the set $\mathcal{S}$.

Dec query (c)

if $\texttt{target} = \texttt{false}$ or $\texttt{functions} = \texttt{false}$, return $\perp$

if c is not in $\mathcal{S}$, $\mathcal{B}$ returns $\texttt{PKE.D}(sk_j, c)$

else $\mathcal{B}$ returns $\perp$.

At the end of the simulation, $\mathcal{B}$ outputs 1 if $\mathcal{A}$ outputs $b = b'$. If $\mathcal{B}$ is in the real world, he simulates G_{0,j^*} perfectly. In the random world, he simulates G_{0,j^*+1} perfectly. Hence, we may conclude that

$$\mathbb{P}[G^{\mathcal{A}}_{0,j^*} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G^{\mathcal{A}}_{0,j^*+1} \mid \overline{\mathbf{coll}}] \leq \mathbf{Adv}^{\ell\text{-mk-sci-pr}}_{\mathcal{H},\mathcal{B}}(\lambda).$$

□

Game 2 is the same as G_1, except that, for the target public key, a fresh output is chosen for each encryption rather than using the PRF. If an adversary can distinguish games 1 and 2 when there are no collisions in the hash keys, then we may use this adversary to win the PRF game.

Lemma 5.5.2. The difference between the success probabilities of any adversary $\mathcal{A}$ is bounded by a PRF adversary $\mathcal{B}$ as follows:

$$\mathbb{P}[G^{\mathcal{A}}_1 \mid \overline{\mathbf{coll}}] - \mathbb{P}[G^{\mathcal{A}}_2 \mid \overline{\mathbf{coll}}] \leq q_\phi \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{C}}(\lambda).$$

Proof. Let $G_{1,i}$ denote the game in which, for function $k \leq i$, a uniformly random value is chosen rather than using the PRF, whereas for $k > i$ the PRF is used. Observe that $G_1 = G_{1,0}$ and $G_2 = G_{1,q_\phi}$, from which we see that

$$\begin{aligned}
\mathbb{P}[G^{\mathcal{A}}_1 \mid \overline{\mathbf{coll}}] - \mathbb{P}[G^{\mathcal{A}}_2 \mid \overline{\mathbf{coll}}] &= \sum_{i=0}^{q_\phi - 1} \mathbb{P}[G^{\mathcal{A}}_{1,i} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G^{\mathcal{A}}_{1,i+1} \mid \overline{\mathbf{coll}}] \\
&\leq q_\phi \cdot (\mathbb{P}[G^{\mathcal{A}}_{1,j^*} \mid \overline{\mathbf{coll}}] - \mathbb{P}[G^{\mathcal{A}}_{1,j^*+1} \mid \overline{\mathbf{coll}}]).
\end{aligned}$$

Again, we have assumed without las of generality that hybrid games j^* and $j^* + 1$ have the largest difference. If an adversary can distinguish games G_{1,j^*}

and G_{1,j^*+1}, then we may use this adversary to construct a PRF adversary with the same advantage in the PRF game. The PRF adversary $\mathcal{C}$ will simulate either game G_{1,j^*} or G_{1,j^*+1}. Adversary $\mathcal{C}$'s setup procedure for the simulation is as follows:

Setup

$b \leftarrow_\$ \{0,1\}$

$r \leftarrow_\$ \texttt{Rnd}$

choose $q_\phi - j^* - 1$ uniformly random PRF keys $\rho_{j^*+2}, \ldots, \rho_{q_\phi}$

$\texttt{funchoice} \leftarrow \texttt{false}$; $\texttt{target} \leftarrow \texttt{false}$

Keys $\leftarrow \emptyset$; Functions $\leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$; $(\hat{pk}^*, \hat{sk}^*) \leftarrow \emptyset$.

When $\mathcal{C}$ finishes this setup procedure, he forwards the description of the hash function to $\mathcal{A}$. Then $\mathcal{C}$ responds to $\mathcal{A}$'s queries as follows:

Func query $(\phi_1, \ldots, \phi_{q_\phi})$

if $\texttt{functions} = \texttt{true}$, $\mathcal{C}$ returns $\perp$

otherwise $\mathcal{C}$ chooses hash keys $k_1, \ldots, k_\ell$ uniformly at random, making sure they are all distinct

for $i = 1$ to ℓ, $\mathcal{C}$ generates $(pk_i, sk_i) \leftarrow_\$ \texttt{PKE.K}(1^\lambda)$

$\mathcal{C}$ sets $\hat{pk}_i = pk_i || k_i$ and $\hat{sk}_i = sk_i$

$\texttt{functions} \leftarrow \texttt{true}$

$\mathcal{C}$ returns $\{\hat{pk}_i\}$ to $\mathcal{A}$.

Target query (j)

if $\texttt{target} \leftarrow \texttt{true}$, $\mathcal{C}$ returns $\perp$

otherwise $\mathcal{C}$ sets $(\hat{pk}^*, \hat{sk}^*) \leftarrow (\hat{pk}_j, \hat{sk}_j)$

$\texttt{target} \leftarrow \texttt{true}$

for $i \neq j$, $\mathcal{C}$ returns $\{\hat{sk}_i\}$.

Enc query $(\hat{pk}, m, 1, \kappa)$

$\mathcal{C}$ parses $\hat{pk} = pk_i || k_i$

if $i = j$

if $\kappa < j^* + 1$, $\mathcal{C}$ chooses $r' \leftarrow_{\$} \texttt{Rnd}$

else if $\kappa = j^* + 1$, $\mathcal{C}$ submits $\hat{pk_i}||m$ to his oracle and receives r'

else if $\kappa > j^* + 1$, $\mathcal{C}$ computes $r' \leftarrow F_{\rho_\kappa}(\hat{pk}||m)$

else $\mathcal{C}$ computes $r' \leftarrow F_{h_{k_i}(\phi_\kappa(r))}(m)$

$\mathcal{C}$ returns $c \leftarrow \texttt{PKE.E}(pk_i, m; r')$.

LR query $(m_0, m_1, 1, \kappa)$

if $\kappa < j^* + 1$, $\mathcal{C}$ chooses $r' \leftarrow_{\$} \texttt{Rnd}$

else if $\kappa = j^* + 1$, $\mathcal{C}$ submits $\hat{pk_j}||m_b$ to his oracle and receives output r'

else if $\kappa > j^* + 1$, $\mathcal{C}$ computes $r' \leftarrow F_{\rho_\kappa}(\hat{pk_j}||m_b)$

$\mathcal{C}$ returns $\texttt{PKE.E}(pk^*, m_b; r')$ $\mathcal{C}$ adds the ciphertext c to $\mathcal{S}$.

Dec query (c)

if c is not in $\mathcal{S}$, $\mathcal{C}$ returns $\texttt{PKE.D}(sk_j, c)$

else $\mathcal{C}$ returns $\perp$.

At the end of the simulation, $\mathcal{C}$ outputs 1 if and only if $\mathcal{A}$ outputs $b = b'$. If $\mathcal{C}$ is in the real world, he simulates G_{1,j^*} perfectly. In the random world, he simulates G_{1,j^*+1} perfectly. Hence, we may conclude that

$$\mathbb{P}[G^{\mathcal{A}}_{1,j^*} \mid \overline{\textbf{coll}}] - \mathbb{P}[G^{\mathcal{A}}_{1,j^*+1} \mid \overline{\textbf{coll}}] \leq \textbf{Adv}^{\text{prf}}_{F,\mathcal{C}}(\lambda).$$

□

Finally, the success in game G_2 may be related to that of a standard IND-ATK adversary. Specifically:

Lemma 5.5.3. For any adversary $\mathcal{A}$, we may bound $\mathcal{A}$'s advantage by an IND-ATK adversary's advantage as follows:

$$2 \cdot \mathbb{P}[G^{\mathcal{A}}_2 \mid \overline{\textbf{coll}}] - 1 \leq \ell \cdot q_{LR} \cdot \textbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{D}}(\lambda).$$

Proof. Again, we use a hybrid argument. Let $G_{2,i}$ denote the game in which, for the kth **LR** query, if $k \leq q_{LR} - i$, **LR** queries are answered with an

encryption of m_0, whilst if $k > q_{LR} - i$ **LR** queries are answered with an encryption of m_1. We have

$$\begin{aligned} 2 \cdot \mathbb{P}[G_2^{\mathcal{A}} \mid \overline{\mathbf{coll}}] - 1 &= \mathbb{P}[\mathcal{A}^{G_{2,q_{LR}}} \Rightarrow 1 \mid \overline{\mathbf{coll}}] - \mathbb{P}[\mathcal{A}^{G_{2,0}} \Rightarrow 1 \mid \overline{\mathbf{coll}}] \\ &\leq q_{LR} \cdot (\mathbb{P}[\mathcal{A}^{G_{2,j^*+1}} \Rightarrow 1 \mid \overline{\mathbf{coll}}] - \mathbb{P}[\mathcal{A}^{G_{2,j^*}} \Rightarrow 1 \mid \overline{\mathbf{coll}}]). \end{aligned}$$

If an adversary $\mathcal{A}$ can distinguish the games G_{2,j^*} and G_{2,j^*+1}, then we may construct an adversary that distinguishes in the standard IND-ATK game.

The IND-ATK adversary $\mathcal{D}$ first runs his **Initialise** procedure and is given a public key pk^*. Then $\mathcal{D}$ sets-up the simulation for $\mathcal{A}$ as follows:

Setup

choose a uniformly random $t \in \{1, \ldots, \ell\}$

choose $r \leftarrow_{\$} \texttt{Rnd}$

ctr $\leftarrow 1$

$\texttt{funchoice} \leftarrow \texttt{false}$; $\texttt{target} \leftarrow \texttt{false}$

Keys $\leftarrow \emptyset$; Functions $\leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$; $(\hat{pk}^*, \hat{sk}^*) \leftarrow \emptyset$.

When completed, the IND-ATK adversary $\mathcal{D}$ forwards the description of the hash function to $\mathcal{A}$ and answers $\mathcal{A}$'s oracle queries as follows:

Func query $(\phi_1, \ldots, \phi_{q_\phi})$

if $\texttt{functions} = \texttt{true}$, $\mathcal{D}$ returns $\perp$

otherwise $\mathcal{D}$ chooses hash keys $k_1, \ldots, k_\ell$ uniformly at random, making sure they are all distinct

for $i \in \{1, \ldots, t-1, t+1, \ldots, \ell\}$ $\mathcal{D}$ generates $(pk_i, sk_i) \leftarrow_{\$} \texttt{PKE.K}(1^\lambda)$

$\mathcal{D}$ sets $pk_t = pk^*$

for $i = 1$ to ℓ, $\mathcal{D}$ sets $\hat{pk}_i = pk_i || k_i$ and $\hat{sk}_i = sk_i$

$\texttt{functions} \leftarrow \texttt{true}$

$\mathcal{D}$ returns $\{\hat{pk}_i\}$ to $\mathcal{A}$.

Target query (j)

if $\texttt{target} = \texttt{true}$, $\mathcal{D}$ returns $\perp$

$\mathcal{D}$ sets $(\hat{pk}^*, \hat{sk}^*) \leftarrow (\hat{pk}_j, \hat{sk}_j)$

$\texttt{target} \leftarrow \texttt{true}$

for $i \neq j$, $\mathcal{D}$ returns $\{\hat{sk}_i\}$.

Enc query $(\hat{pk}, m, 1, \kappa)$

$\mathcal{D}$ parses $\hat{pk} = pk_i || k_i$

if $i = j$, $\mathcal{D}$ chooses $r' \leftarrow_{\$} \texttt{Rnd}$

else $\mathcal{D}$ computes $r' \leftarrow F_{h_{h_i}(\phi_\kappa(r))}(m)$

$\mathcal{D}$ returns $c \leftarrow \texttt{PKE.E}(pk_i, m; r')$.

LR query $(m_0, m_1, 1, \kappa)$

if ctr $= q_{LR} - j^*$, $\mathcal{D}$ submits (m_0, m_1) to his **LR** oracle and receives c

else, $\mathcal{D}$ chooses $r' \leftarrow_{\$} \texttt{Rnd}$

if ctr $< q_{LR} - j^*$, $\mathcal{D}$ computes $c \leftarrow \texttt{PKE.E}(pk_j, m_0; r')$

else if ctr $> q_{LR} - j^*$, $\mathcal{D}$ computes $c \leftarrow \texttt{PKE.E}(pk_j, m_1; r')$

ctr $\leftarrow$ ctr $+ 1$

$\mathcal{D}$ adds the ciphertext c to the set $\mathcal{S}$

$\mathcal{D}$ returns c.

Dec query (c)

if c is not in $\mathcal{S}$, $\mathcal{D}$ returns $\texttt{PKE.D}(sk_j, c)$.

else $\mathcal{D}$ returns $\perp$.

At the end of the simulation, if $\mathcal{A}$ has chosen pk_t as his target public key then $\mathcal{D}$ outputs the same bit as $\mathcal{A}$, otherwise $\mathcal{D}$ outputs a uniformly random bit. If $\mathcal{A}$ has chosen pk_t as his target public key (which he does with probability $1/\ell$) then, when $\mathcal{D}$ is given an encryption of m_0, he simulates G_{2,j^*} perfectly. If he receives an encryption of m_1, he simulates G_{2,j^*+1} perfectly. Hence, we may conclude that

$$\frac{1}{\ell}(\mathbb{P}[\mathcal{A}^{G_{2,j^*+1}} \Rightarrow 1 \mid \overline{\mathbf{coll}}] - \mathbb{P}[\mathcal{A}^{G_{2,j^*}} \Rightarrow 1 \mid \overline{\mathbf{coll}}] \leq \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{D}}(\lambda).$$

□

The theorem follows by combining the preceding lemmas. □

It remains to show that we can instantiate a hash function satisfying the above defined multi-key correlated-input security notion. This is achieved by extending the security results for the CIS hash function defined in [32]. Concretely, the CIS hash function from [32] is defined as follows:

$\texttt{GenFun}(1^\lambda)$: Pick a group $\mathbb{G}$ of prime order p, a generator g, and set the keyspace to $\mathcal{K} = \mathbb{Z}_p$, the domain to $\mathcal{D} = \mathbb{Z}_p$, and the range to $\mathcal{R} = \mathbb{G}$. Return $(\mathcal{K}, \mathcal{D}, \mathcal{R}, h, g)$ where h is a description of the function defined below.

$h_k(x)$: For $k \in \mathcal{K}$ and $x \in \mathcal{D}$, return

$$h_k(x) = g^{\frac{1}{x+a}},$$

where $1/(x+a)$ is computed modulo p.

Based on the q-Decisional Diffie Hellman Inversion (q-DDHI) assumption in $\mathbb{G}$ (see Section 2.1.2), and extending the results of [32], it was shown in [65] that the above hash function achieves multi-key correlated-input pseudorandomness for a class of functions consisting of uniform-output polynomials of bounded degree.

Note 1. Our 'partial' RKA-secure PRF is only secure when an adversary's function queries are non-adaptive, which is why we are only able to prove selective HK-RRA-ATK security.

Note 2. The above construction is only shown to achieve HK-RRA-ATK security, as opposed to RRA-ATK security. The technical reason for this is that public keys include a hash key, and the CIS hash function is only assumed to be secure for honestly generated keys. An alternative solution would be to introduce a *common reference string* (CRS) containing a single hash key, and let all users make use of this. While this requires a trusted third party to

initially set up the CRS, it would be possible to show RRA-ATK security of the above construction in a security model appropriately extended to model the presence of a CRS.

Likewise, if we had a multi-key CIS hash function that remained secure for maliciously chosen keys, then we would be able to obtain full RRA-ATK security for the above construction. Unfortunately, we are currently unaware of how to obtain such CIS hash functions.

5.6 Related Work

Since the publication of the material in this chapter, there has appeared another academic paper concerning related randomness attacks by Yuen et al. [83]. The paper considers various cryptosystems including Identity Based Encryption (IBE), PKE, and Digital Signatures. They allow an adversary to tamper with both the private key and the randomness for encryption (or signing, in the case of signatures). In the PKE setting, when the adversary is not allowed to tamper with the private key, their results are broadly comparable to our results in Section 5.4. However, the paper by Yuen et al. makes no mention of multiple randomness indices or multiple **LR** queries. In addition, adversaries are not allowed to see encryptions under maliciously chosen public keys, whereas they are in the work presented in this thesis. Hence, whilst some results of [83] are broadly similar to ours, there are some major differences in the security models that seem to make the different models incomparable.

5.7 A Brief Detour into Symmetric Encryption

In this section we will briefly explore bad randomness for symmetric-key encryption (SKE). The reason for this brief detour is that we are able to improve on current results in this area by using the techniques of the previous sections. We will see that there are symmetric encryption schemes in the literature that are easily seen to be secure when an adversary is able to *choose* the randomness for encryption, as long as we impose suitable equality-pattern restrictions on adversarial queries. Recall that these equality-patterns were also required in the public-key setting in order to prevent trivial wins by an adversary. By imposing these minimal conditions on adversaries, we achieve a rather strong result. In order to keep the notation consistent with other sections, we will henceforth refer to symmetric encryption schemes as Data Encapsulation Mechanisms (DEMs).

Figure 5.10 defines the Chosen Ciphertext and Randomness Attack (CCRA) game in the symmetric setting. The CCRA notion was first proposed by Kamara and Katz [48], but the game we present is a modified version of their game. Notice that the Related Randomness Attack game for PKE (Figure 5.3) allowed adversaries to access an **Enc** oracle, but the CCRA game (Figure 5.10) does not have such an oracle. This is because the **Enc** oracle was designed to give adversaries the ability to see encryptions under an adversarially-chosen public keys with the *unknown* randomness $\phi(r)$. In this new setting, the randomness is clearly not unknown (because it is adversarially-chosen), so the adversary does not need the **Enc** oracle.

Definition 5.7.1. Let $\mathcal{A}$ be an adversary in Game CCRA. Suppose $\mathcal{A}$ makes q **LR** queries with randomness r. Let $(m_0^{r,1}, m_1^{r,1}), \ldots, (m_0^{r,q}, m_1^{r,q})$ be $\mathcal{A}$'s **LR** queries for randomness r. Suppose that for all $r \in \texttt{Rnd}$ and for all

proc. Initialise(λ):	**proc. LR**(m_0, m_1, r):	**proc. Dec**(c):	**proc. Finalise**(b'):
$b \leftarrow_\$ \{0,1\}$;	$c \leftarrow \mathtt{SKE.E}(K, m_b; r)$	if $c \in \mathcal{S}$	if $b = b'$
$K \leftarrow_\$ \mathtt{SKE.K}(1^\lambda)$.	$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$	return $\perp$	return 1
	return c.	else	else
		return $\mathtt{SKE.D}(K, c)$.	return 0.

Figure 5.10: Game CCRA.

Alg. $\mathtt{PRF\text{-}SKE.K}(1^\lambda)$:	Alg. $\mathtt{PRF\text{-}SKE.E}(K, m; r)$:	Alg. $\mathtt{PRF\text{-}SKE.D}(K, c)$:
$K_1 \leftarrow_\$ \mathtt{SKE.K}(1^\lambda)$	$r' \leftarrow F_{K_2}(m \| r)$	$m \leftarrow \mathtt{SKE.D}(K_1, c)$
$K_2 \leftarrow_\$ \mathtt{PRF.K}(1^\lambda)$	$c \leftarrow \mathtt{SKE.E}(K_1, m; r')$	return m.
$K \leftarrow (K_1, K_2)$	return c.	
return K.		

Figure 5.11: Scheme PRF-SKE built from a standard SKE scheme SKE and a PRF F.

$j \neq k \in [q]$, we have:

$$m_0^{r,j} = m_0^{r,k} \text{ iff } m_1^{r,j} = m_1^{r,k}.$$

Then we say that $\mathcal{A}$ is equality-pattern respecting.

Definition 5.7.2. We define the advantage of an equality-pattern respecting, CCRA adversary $\mathcal{A}$ against an SKE scheme SKE to be

$$\mathbf{Adv}^{\text{ccra}}_{\mathtt{SKE},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\text{CCRA}^{\mathcal{A}}_{\mathtt{SKE}}(\lambda) \Rightarrow 1] - 1.$$

An SKE scheme SKE is said to be CCRA secure if the advantage of any equality-pattern respecting, CCRA adversary against SKE that runs in polynomial time is negligible in the security parameter λ.

The scheme in Figure 5.11 was proposed in [48] with the intention of protecting against (their definition of) Chosen Ciphertext and Randomness Attacks. Their model allowed an adversary to see encryptions under the target private key using adversarially-chosen randomness. However, the adversary was not allowed to choose the randomness for the challenge encryptions. Instead, the **LR** queries used fresh, uniform randomness. However, the encryption algorithm presented in [48] is actually secure in our stronger model (Figure 5.10), as we shall now see.

Theorem 5.7.1. Consider an equality-pattern respecting, CCRA adversary $\mathcal{A}$ attacking scheme PRF-SKE in Figure 5.11. There exists a PRF adversary $\mathcal{B}$ and an IND-CCA adversary $\mathcal{C}$ such that

$$\mathbf{Adv}^{\text{ccra}}_{\texttt{PRF-SKE},\mathcal{A}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{B}}(\lambda) + \mathbf{Adv}^{\text{ind-cca}}_{\texttt{SKE},\mathcal{A}}(\lambda),$$

where $\mathcal{C}$ makes the same number of decryption queries as $\mathcal{A}$, and the running times of $\mathcal{B}$ and $\mathcal{C}$ are approximately the same as that of $\mathcal{A}$.

Proof. Let G_0 denote the CCRA game, and let G_1 denote the game where every encryption uses fresh, uniform randomness (instead of using the PRF). Then

$$\begin{aligned}
\mathbf{Adv}^{\text{ccra}}_{\texttt{PRF-SKE},\mathcal{A}}(\lambda) &= 2 \cdot \mathbb{P}[\text{CCRA}^{\mathcal{A}}_{\texttt{PRF-SKE}}(\lambda) \Rightarrow 1] - 1 \\
&= 2 \cdot \mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - 1 \\
&= 2 \cdot (\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1]) + 2 \cdot \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - 1.
\end{aligned}$$

If adversary $\mathcal{A}$ can distinguish games 0 and 1, then we may construct an adversary $\mathcal{B}$ that wins the PRF game. The adversary $\mathcal{B}$ runs as follows. First, $\mathcal{B}$ randomly picks a bit $b \leftarrow_{\$} \{0,1\}$, then runs $\texttt{SKE.K}$ to obtain a uniformly random key K. When $\mathcal{A}$ submits **LR** query (m_0, m_1, r), $\mathcal{B}$ forwards $m_b||r$ to his oracle and obtains output r'. Then $\mathcal{B}$ calculates $c \leftarrow \texttt{SKE.E}(K, m_b; r')$ and returns c to $\mathcal{A}$. When $\mathcal{A}$ submits decryption query c, $\mathcal{B}$ returns $\texttt{SKE.D}(K, c)$. Finally, when $\mathcal{A}$ halts and outputs bit b', $\mathcal{A}$ halts and outputs 1 if and only if $b = b'$. It follows that

$$\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}^{\text{prf}}_{F,\mathcal{B}}(\lambda).$$

Furthermore, the game G_1 may be simulated by a standard IND-CCA adversary (since fresh, uniform randomness is used for every encryption). Hence

$$2 \cdot \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - 1 \leq \mathbf{Adv}^{\text{ind-cca}}_{\texttt{SKE},\mathcal{C}}(\lambda).$$

The theorem follows by combing the three previous equations. □

5.8 Conclusions

In this chapter we have introduced models for a new type of attack, which we call the Related Randomness Attack. We have proved necessary and sufficient conditions for achieving this new notion of security in the Random Oracle Model, and we have provided two generic transforms in the standard model that convert IND-ATK secure PKE schemes into schemes that are secure with respect to our new related randomness notion. Furthermore, we have been able to provide concrete instantiations of these schemes. In addition, our transforms develop interesting connections with RKA-PRFs and CIS hash functions.

Open problems include achieving security against broader classes of functions, and an interesting direction for future research would be to develop this related randomness theme further, by examining security in a combined RKA/RRA setting, where the adversary would be able to simultaneously tamper with all the inputs to a PKE scheme. Initial research in this direction was recently presented by Yuen et al. [83].

Chapter 6

Function-Vector Related Randomness Attacks

The previous standard model constructions for PKE secure against Related Randomness Attacks from Chapter 5 concerned polynomial functions ϕ of certain types (i.e. non-constant and bounded degree). In this chapter we turn our attention to alternative classes of functions. We begin by formalising an alternative notion of Related Randomness Attacks, which we call the Function-Vector Related Randomness Attack. The new security game is closely connected to the game for PKE with auxiliary-inputs. We exploit this connection by taking a well-known scheme that is secure in the auxiliary-input game, and we modify it in such a way that we can prove its security in our new related randomness game. The class of functions ϕ for which our scheme is secure is the set of hard-to-invert functions. Unfortunately, our basic Function-Vector game restricts challenge functions to the identity function. In order to generalise this approach, we provide security games for a setting where challenge functions are not limited to the identity function, and we show that a particular type of reconstructive-extractor will yield a secure PKE scheme in this model when combined with other primitives.

6.1 Function-Vector Related Randomness Security

In the standard model, in Sections 5.4 and 5.5, we considered Related Randomness Attacks and Honest-Key Related Randomness Attacks, respectively. For each model, we provided instantiations of PKE schemes that were secure against certain classes of polynomial functions. In this section we turn our attention to an alternative notion, which we call *Function-Vector Related Randomness Attack* (FV-RRA) security, and is based on the game in Figure 6.1. Here, the adversary is parametrised by a vector of functions $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$, and is limited to using only these functions in its oracle queries. Additionally, we restrict the adversary by demanding that the **LR** queries use only the identity function. However, once again, the adversary has complete freedom over public keys submitted to its encryption oracle. Furthermore, security will be quantified over *all* choices of vector from a particular class. Specifically, in our construction in Section 6.1, we will demand that security holds over all vectors $\boldsymbol{\phi}$ that are simultaneously hard-to-invert on a common random input r. This quantification actually makes our notion rather strong. In this section we will introduce the formal definitions and the security model for our notion of Function-Vector Related Randomness Attacks. We begin with the definition of an equality-pattern respecting adversary, which should be familiar from the previous chapter. Here, the definition has been adapted specifically to our new security game in Figure 6.1. Recall that the equality-patterns are designed to prevent trivial wins for an adversary (by checking ciphertext equality). For the constructions we will see, there will be no intersection between functions used for **LR** queries and functions used for **Enc** queries. Hence, an adversary cannot trivially win the game by matching **Enc** and **LR** queries, and we can therefore use the simplified definition seen below.

Definition 6.1.1. Let $\mathcal{A}$ be an adversary in Game $\boldsymbol{\phi}$-FV-RRA-ATK that queries

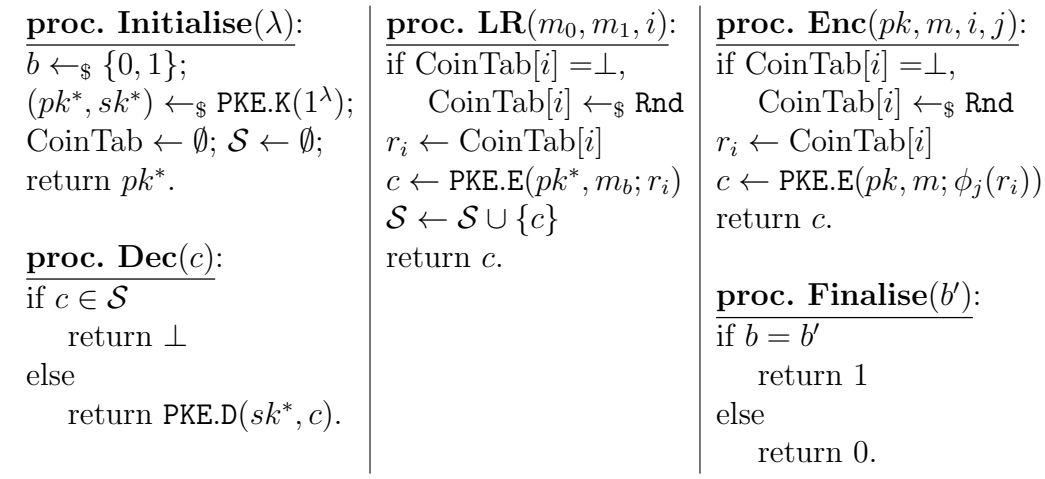

Figure 6.1: Game $\boldsymbol{\phi}$-FV-RRA-ATK, where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$. (As usual, if ATK = CPA, then the adversary's access to **proc. Dec** is removed.)

r different randomness indices to its **LR** and **Enc** oracles and makes q_i queries to its **LR** oracle with index i. Let $(m_0^{i,1}, m_1^{i,1}), \ldots, (m_0^{i,q_i}, m_1^{i,q_i})$ be $\mathcal{A}$'s **LR** queries for index $i \in [r]$. Suppose that for all $i \in [r]$ and for all $j \neq k \in [q_i]$, we have

$$m_0^{i,j} = m_0^{i,k} \text{ iff } m_1^{i,j} = m_1^{i,k}.$$

Then we say that $\mathcal{A}$ is *equality-pattern respecting*.

The following definition encompasses our notion of security with respect to the security game in Figure 6.1.

Definition 6.1.2. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ be a vector of $q := q(\lambda)$ functions. We define the advantage of an equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-ATK adversary $\mathcal{A}$ against a PKE scheme PKE to be

$$\mathbf{Adv}^{\boldsymbol{\phi}\text{-fv-rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\boldsymbol{\phi}\text{-FV-RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1] - 1.$$

If $\boldsymbol{\Phi}$ is a set of vectors of functions, then a PKE scheme PKE is said to be $\boldsymbol{\Phi}$-FV-RRA-ATK secure if, for all $\boldsymbol{\phi} \in \boldsymbol{\Phi}$, the advantage of any equality-pattern respecting $\boldsymbol{\phi}$-FV-RRA-ATK adversary against PKE that runs in polynomial time is negligible in the security parameter λ.

In order or to simplify proofs of security in this model, we will utilise the lemma below. It is an easy extension of Lemma 5.2.1 and is therefore presented without a proof.

Lemma 6.1.1. Consider an equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-ATK adversary $\mathcal{A}$ that queries q_r distinct randomness indices and makes at most q_{LR} **LR** queries. There exists an equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-ATK adversary $\mathcal{B}$ that queries at most one randomness index and makes at most q_{LR} **LR** queries such that

$$\mathbf{Adv}^{\boldsymbol{\phi}\text{-fv-rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) \leq q_r \cdot \mathbf{Adv}^{\boldsymbol{\phi}\text{-fv-rra-atk}}_{\mathtt{PKE},\mathcal{B}}(\lambda),$$

where $\mathcal{B}$ runs in approximately the same time as $\mathcal{A}$. In the CCA setting, $\mathcal{B}$ makes the same number of decryption queries as $\mathcal{A}$.

In Section 6.3 we will provide definitions for a slightly stronger model in which the challenge functions are not required to be the identity. Unfortunately, we are unable to provide any concrete instantiations that achieve this strong notion, so we defer discussion of this security game and definition to the second half of this chapter

Comparison of security notions The relation between full RRA-ATK security and FV-RRA-ATK security is not immediately obvious. Aside from the restriction on **LR**-queries in FV-RRA-ATK security, there is a subtle distinction between requiring security for all vectors $\boldsymbol{\phi}$ of functions from a particular set Φ and requiring security for a fully adaptive choice of functions $\phi \in \Phi$. In particular, the former notion will allow a security reduction to consider multiple runs of an adversary with different random coins for a fixed choice of function vector $\boldsymbol{\phi}$, whereas the latter notion will leave open the possibility that an adversary will chose a different sequence of functions ϕ in each run. Also note that FV-RRA-ATK security guarantees that there is no choice of $\boldsymbol{\phi}$

for which the considered scheme is weak, even if this choice might be computationally hard for an adaptive adversary to find. Furthermore, the relation between the notions might also be influenced by the considered class of functions Φ. It remains future work to fully explore and categorise the possible notions of RRA security.

6.1.1 The modified BHHO scheme

In this section we will propose a construction for a PKE scheme that is $\boldsymbol{\Phi}$-FV-RRA-CPA secure for the set $\boldsymbol{\Phi}$ of vectors of functions that are hard-to-invert, in a sense that we make precise next.

Definition 6.1.3. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ denote a vector of functions on a set $\mathtt{Rnd}_\lambda$, where $q := q(\lambda)$ is polynomial in the security parameter λ. Let $\delta(\lambda)$ be a function. We say that $\boldsymbol{\phi}$ is $\delta(\lambda)$-*hard-to-invert* if, for all polynomial-time algorithms $\mathcal{A}$ and all sufficiently large λ, we have:

$$\mathbb{P}[r \leftarrow \mathcal{A}(\phi_1(r), \ldots, \phi_q(r)) : r \leftarrow_{\$} \mathtt{Rnd}_\lambda] \leq \delta(\lambda).$$

We say that a set of vectors of functions $\boldsymbol{\Phi}$ is δ-hard-to-invert if each vector $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ is δ-hard-to-invert (note that the vectors in such a set $\boldsymbol{\Phi}$ need not all be of the same dimension, but we assume they each have dimension that is polynomial in λ).

We will now construct a PKE scheme that offers $\boldsymbol{\Phi}$-FV-RRA-CPA security, where $\boldsymbol{\Phi}$ is the set of *all* sufficiently hard-to-invert vectors of functions on the scheme's randomness space $\mathtt{Rnd}$. As noted in Section 5.2, security in this setting is quantified over *all* vectors in $\boldsymbol{\Phi}$, and the adversary is allowed to work with any set of public keys (even maliciously generated) in its attack. This makes our result relatively strong.

To ease the security analysis of our scheme, we will use a variant of the

standard DDH assumption.

Definition 6.1.4. Let $\mathbb{G}$ be a cyclic group of prime order p. The game q-DDH in $\mathbb{G}$ selects generators $g_1, \ldots, g_q$ from $\mathbb{G}$ and a bit $b \leftarrow_\$ \{0,1\}$. The game chooses $(r_1, \ldots, r_q) \leftarrow_\$ \mathbb{Z}_p^q$ and $r \leftarrow_\$ \mathbb{Z}_p$. If $b = 1$, the game returns $g_1, \ldots, g_q, g_1^r, \ldots, g_q^r$ to the adversary. Otherwise, the game returns $g_1, \ldots, g_q, g_1^{r_1}, \ldots, g_q^{r_q}$ to the adversary. When the adversary returns a bit b', the game outputs 1 if and only if $b = b'$. We then define the advantage of a q-DDH adversary $\mathcal{A}$ to be

$$\mathbf{Adv}^{q\text{-ddh}}_{\mathbb{G},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[q\text{-DDH}^{\mathcal{A}}_{\mathbb{G}}(\lambda) \Rightarrow 1] - 1.$$

Assumption 6.1.1 (The q-Decisional Diffie Hellman (q-DDH) Assumption). For any polynomial-time adversary $\mathcal{A}$, and any q that is polynomial in λ, we have:

$$\mathbf{Adv}^{q\text{-ddh}}_{\mathbb{G},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda).$$

The q-DDH assumption follows from the standard Decisional Diffie Hellman assumption [63].

With these definitions in hand, Figure 6.2 defines our PKE scheme `mBHHO` which offers security in the FV-RRA-CPA setting. This scheme is obtained by modifying a PKE scheme of Boneh et al. [17] (the `BHHO` scheme) that Dodis et al. [23] showed to be secure in the auxiliary-input setting. In the auxiliary-input setting, the adversary plays the IND-ATK game, but is given some additional information. Specifically, the adversary is given a function of the private key $h(sk)$, where h maps from the domain of private keys to arbitrarily-long bit strings. Dodis et al. showed that the BHHO scheme was secure in this model, and our idea is to switch the roles of the randomness and private key in the BHHO scheme, and hence obtain a scheme that is secure when we leak $h(r)$ for some random value r. We can then interpret this function h as a vector of related randomness functions.

We will now discuss the scheme in more detail. The scheme makes use of a KDF f and a PRF F with certain domains and ranges, and a DEM $\mathtt{DEM}$. To arrive at our modified scheme $\mathtt{mBHHO}$, we swap the roles of secret key and randomness in the original $\mathtt{BHHO}$ scheme. This then enables us to provide the values $\phi_i(r)$ as auxiliary inputs without undermining the usual IND-CPA security of the scheme; in turn, these values enables our security reduction to properly handle **Enc** queries involving any function ϕ_i. For technical reasons discussed below, we also need to set the randomness space of the scheme to be $\{0,1\}^k$ where $k := k(\lambda)$ denotes a polynomial function of λ. Theorem 6.1.2 gives our formal result concerning the FV-RRA-CPA security of this scheme, but before we discuss the theorem and proof we will introduce a theorem of Dodis et al. [23] that we will require in our proof.

Theorem 6.1.1. Let p be a prime, and let H be an arbitrary subset of $\mathbb{Z}_p$. Let $f : H^n \to \{0,1\}^*$ be any (possibly randomised) function. If there is a distinguisher $\mathcal{D}$ that runs in time t such that

$$\begin{aligned} &|\mathbb{P}[\mathbf{r} \leftarrow H^n, y \leftarrow f(\mathbf{r}), \mathbf{s} \leftarrow \mathbb{Z}_p^n : \mathcal{D}(y, \mathbf{s}, \langle \mathbf{r}, \mathbf{s} \rangle) = 1] \\ - \;&\mathbb{P}[\mathbf{r} \leftarrow H^n, y \leftarrow f(\mathbf{r}), \mathbf{s} \leftarrow \mathbb{Z}_p^n, u \leftarrow \mathbb{Z}_p : \mathcal{D}(y, \mathbf{s}, u) = 1]| = \epsilon \end{aligned}$$

then there is an inverter $\mathcal{A}$ that runs in time $t' = t \cdot \mathrm{poly}(n, |H|, 1/\epsilon)$ such that

$$\mathbb{P}[\mathbf{r} \leftarrow H^n, y \leftarrow f(\mathbf{r}) : \mathcal{A}(y) = \mathbf{r}] \geq \frac{\epsilon^3}{512 \cdot n \cdot p^3}. \tag{6.1.1}$$

The bound we have quoted is slightly different to the bound stated in [23], and we discuss this further in Section 6.2. Throughout the remainder of this thesis, we will use the bound stated in Equation (6.1.1).

Theorem 6.1.2. Let $\mathbf{\Phi}$ be the set of δ-hard-to-invert vectors of functions on $\{0,1\}^k$. Consider any polynomial-size vector of functions $\boldsymbol{\phi} \in \mathbf{\Phi}$ and any equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-CPA adversary $\mathcal{A}$ against $\mathtt{mBHHO}$. Suppose $\mathcal{A}$ makes q_{LR} **LR** queries and uses q_r randomness indices. Then

Alg. $\texttt{mBHHO.K}(1^\lambda)$:	Alg. $\texttt{mBHHO.E}(pk, m)$:	Alg. $\texttt{mBHHO.D}(sk, (c_1, c_2))$:
$g_1, \ldots, g_k \leftarrow_\$ \mathbb{G}$	$r \leftarrow_\$ \{0,1\}^k$	$(K, r') \leftarrow f(c_1^x)$
$x \leftarrow_\$ \mathbb{Z}_p$	$c_1 = \prod_{i=1}^k g_i^{r_i}$	$m \leftarrow \texttt{DEM.D}(K, c_2)$
$pk = (g_1, \ldots, g_k, g_1^x \ldots, g_k^x)$	$(K, r') \leftarrow f(\prod_{i=1}^k (g_i^x)^{r_i})$	return m.
$sk = x$.	$r'' \leftarrow F_{r'}(pk \| m)$	
	$c_2 = \texttt{DEM.E}(K, m; r'')$	
	return (c_1, c_2).	

Figure 6.2: Modified BHHO scheme mBHHO, constructed using a PRF, F, a KDF, f, and a DEM DEM.

there exists a k-DDH adversary $\mathcal{B}$, a KDF adversary $\mathcal{D}$, a PRF adversary $\mathcal{E}$, and an IND-CPA adversary $\mathcal{F}$, all running in polynomial time, such that

$$\begin{aligned}\mathbf{Adv}^{\phi\text{-fv-rra-cpa}}_{\texttt{mBHHO},\mathcal{A}}(\lambda) \;<\; & 2q_r \cdot \mathbf{Adv}^{k\text{-ddh}}_{\mathbb{G},\mathcal{B}}(\lambda) + 2q_r \cdot \mathbf{Adv}^{\text{kdf}}_{f,\mathcal{D}}(\lambda) \\ & + 2q_r \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{E}}(\lambda) + q_r \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\texttt{DEM},\mathcal{F}}(\lambda) \\ & + 2q_r \sqrt[3]{512\delta k p^4}.\end{aligned}$$

In particular, when δ is sufficiently small, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

Proof. In what follows, we let $r_{1,i}$ denote the ith bit of r_1 and, without loss of generality, we assume that the function of vectors is of size $q = \text{poly}(\lambda)$. First, we invoke Lemma 5.2.1, so that we now only have to prove the theorem for an adversary using just one randomness value, which we will call r_1. The proof then uses a sequence of games, as follows:

$\mathbf{G_0}$: G_0 is the real game with the scheme defined in Figure 6.2.

$\mathbf{G_1}$: G_1 is the same as G_0, except the target public key components $g_1^x, \ldots, g_k^x$ are replaced with $g^{u_1}, \ldots, g^{u_k}$ where g is a group generator and $u_i \leftarrow_\$ \mathbb{Z}_p$. If $\mathcal{A}$'s success probability is significantly different in games G_0 and G_1, then we can use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that wins the k-DDH game.

$\mathbf{G_2}$: G_2 is the same as G_1, except for the challenge ciphertexts, which use g^w as the input to the KDF where $w \leftarrow_\$ \mathbb{Z}_p$, rather than using $\prod_{i=1}^{k}(g^{u_i})^{r_{1,i}}$. If $\mathcal{A}$'s success probability is significantly different in games G_1 and G_2, then we can use $\mathcal{A}$ to build an adversary $\mathcal{C}$ that inverts the vector of functions $(\phi_1, \ldots, \phi_q)$.

$\mathbf{G_3}$: G_3 is the same as G_2, except that the output of the KDF for the challenge ciphertexts is replaced by a uniformly random value. If $\mathcal{A}$'s success probability is significantly different in games G_2 and G_3, then we can use $\mathcal{A}$ to build an adversary $\mathcal{D}$ that wins the KDF security game.

$\mathbf{G_4}$: G_4 is the same as G_3, except that the PRF outputs used in constructing the challenge ciphertexts are replaced by uniformly random values. If $\mathcal{A}$'s success probability is significantly different in games G_3 and G_4, then we can use $\mathcal{A}$ to build an adversary $\mathcal{E}$ that wins the PRF security game. Finally, $\mathcal{A}$'s success in G_4 can be related to that of an IND-CPA adversary $\mathcal{F}$ against the DEM component of the scheme.

We now analyse each of the game transitions in more detail.

$\mathbf{G_0}$ – $\mathbf{G_1}$: We will prove the following:

Lemma 6.1.2. For any adversary $\mathcal{A}$, there exists a λ-DDH adversary $\mathcal{B}$ such that

$$\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}^{\lambda\text{-ddh}}_{\mathbb{G},\mathcal{B}}(\lambda).$$

Proof. The k-DDH adversary $\mathcal{B}$ with access to $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ will simulate either G_0 or G_1 for $\mathcal{A}$. Adversary $\mathcal{B}$ is given $g_1, \ldots, g_k, g'_1, \ldots, g'_k$, where $g'_1, \ldots, g'_k$ is either $g_1^x, \ldots, g_k^x$ or $g^{u_1}, \ldots, g^{u_k}$ for uniformly random u_i. Adversary $\mathcal{B}$ sets $b \leftarrow_\$ \{0,1\}$ and $r_1 \leftarrow_\$ \{0,1\}^k$. It then simulates **proc. Initialise** in the $\boldsymbol{\phi}$-FV-RRA-CPA security game by forwarding the public-key $pk^* = (g_1, \ldots, g_k, g'_1, \ldots, g'_k)$ to $\mathcal{A}$. Then $\mathcal{B}$ answers $\mathcal{A}$'s queries as follows:

Enc query (pk, m, i)

$\mathcal{B}$ returns $\texttt{mBHHO.E}(pk, m; \phi_i(r_1))$ to $\mathcal{A}$.

LR query (m_0, m_1)

$\mathcal{B}$ returns $\texttt{mBHHO.E}(pk^*, m_b; r_1)$ to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs a bit b', $\mathcal{B}$ halts and outputs 1 if and only if $b = b'$. If $g'_1, \ldots, g'_k = g^x_1, \ldots, g^x_k$, then $\mathcal{B}$ perfectly simulates G_0. If $g'_1, \ldots, g'_k = g^{u_1}, \ldots, g^{u_k}$, then $\mathcal{B}$ perfectly simulates G_1. Since the distributions of $g^{u_1}, \ldots, g^{u_k}$ and $g_1^{x_1}, \ldots, g_k^{x_k}$ are identical, we may conclude that

$$\begin{aligned}\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] &= \mathbb{P}[\mathcal{B} \Rightarrow 1 | (g'_1, \ldots, \ldots, g'_k) = (g^x_1, \ldots, g^x_k)] \\ &\quad - \mathbb{P}[\mathcal{B} \Rightarrow 1 | (g'_1, \ldots, \ldots, g'_k) = (g^{u_1}, \ldots, g^{u_k})] \\ &= \mathbf{Adv}^{k\text{-ddh}}_{\mathbb{G},\mathcal{B}}(\lambda).\end{aligned}$$

□

$\mathbf{G_1}$ – $\mathbf{G_2}$: We will show that if $\mathcal{A}$'s success probability is significantly different in games G_1 and G_2, then we can use $\mathcal{A}$ to build an adversary $\mathcal{C}$ that inverts the vector of functions $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$. Specifically, we show:

Lemma 6.1.3. For any polynomial-time adversary $\mathcal{A}$, we have

$$|\mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1]| < \sqrt[3]{512\delta p^4 k}.$$

Here, recall that p is the size of the group $\mathbb{G}$ while $\boldsymbol{\phi}$ is a δ-hard-to-invert vector of functions.

Proof. If the success of $\mathcal{A}$ differs between games G_1 and G_2, then we can construct an adversary $\mathcal{C}$ that inverts the vector of functions $\boldsymbol{\phi}$ on input r_1. First, we consider an intermediate step. Suppose adversary $\mathcal{A}'$ is attempting to distinguish tuples of the form $T_1 = (g_1, \ldots, g_k, \prod_{i=1}^k g_i^{r_{1,i}}, \boldsymbol{\phi}(r_1), u, \langle r_1, u \rangle)$ from tuples of the form $T_2 = (g_1, \ldots, g_k, \prod_{i=1}^k g_i^{r_{1,i}}, \boldsymbol{\phi}(r_1), u, w)$, where w is uniformly

random. If $\mathcal{A}$ can distinguish games 1 and 2 with probability ϵ, then $\mathcal{A}'$ can distinguish the previous two tuples with probability ϵ. The simulation runs as follows. Distinguisher $\mathcal{A}'$ is given a tuple $(g_1, \ldots, g_k, \prod_{i=1}^{k} g_i^{r_{1,i}}, \boldsymbol{\phi}(r_1), u, z)$, where z is either $\langle r_1, u \rangle$ or uniformly random. Then $\mathcal{A}'$ chooses a uniformly random generator g and bit b. $\mathcal{A}'$ uses the generators and the vector u to form a public key $pk^* = (g_1, \ldots, g_k, g^{u_1}, \ldots, g^{u_k})$, where u_i is the ith component of the vector u, and forwards this public key to $\mathcal{A}$. Then the distinguisher $\mathcal{A}'$ will answer the oracle queries of $\mathcal{A}$ as follows:

Enc query (pk, m, i)

return $\texttt{mBHHO.E}(pk, m; \phi(r_1))$ to $\mathcal{A}$.

LR query (m_0, m_1)

$c_1 \leftarrow \prod_{i=1}^{k} g_i^{r_{1,i}}$

$(K, r) \leftarrow f(g^z)$

$r' \leftarrow F_r(pk^* || m_b)$

$c_2 \leftarrow \texttt{DEM.E}(K, m_b; r')$

return (c_1, c_2) to $\mathcal{A}$.

When $\mathcal{A}$ outputs bit b', $\mathcal{A}'$ outputs 1 if and only if $b = b'$. It follows that

$$|\mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1]| = |\mathbb{P}[\mathcal{A}'_{T_1} \Rightarrow 1] - \mathbb{P}[\mathcal{A}'_{T_2} \Rightarrow 1]|,$$

where $\mathcal{A}'_{T_i} \Rightarrow 1$ denotes that $\mathcal{A}'$ outputs 1 when given tuple T_i.

Now we shall use the distinguisher $\mathcal{A}'$ to construct our adversary $\mathcal{C}$ that will invert the vector of functions by using a modified version of Theorem 1 of [23]. The theorem shows how an adversary may invert a function when given access to a distinguisher that distinguishes tuples of the form $(\boldsymbol{\phi}(r_1), u, \langle r_1, u \rangle)$ from tuples of the form $(\boldsymbol{\phi}(r_1), u, w)$, where w is uniformly random. We therefore need a slight modification of their proof in order to invert the function when given tuples of the form T_1 or T_2 since these tuples require the extra parameters $g_1, \ldots, g_k, \prod_{i=1}^{k} g_i^{r_{1,i}}$. The inverter $\mathcal{C}$ can easily choose the generators, but the

value $\prod_{i=1}^{k} g_i^{r_{1,i}}$ must be guessed by $\mathcal{C}$ (since he does not have the necessary information to form this value correctly), hence $\mathcal{C}$'s advantage is conditioned on the probability that he correctly supplies this value to $\mathcal{A}'$. The modified theorem we require is as follows:

Theorem 6.1.3. Let p be a prime, let $\mathbb{G}$ be a group of size p, and let $H = \mathbb{Z}_2$. Let $\boldsymbol{\phi} : \mathbb{Z}_2^k \to \{0,1\}^*$ be a vector of functions, and let $g_1, \ldots, g_k$ be generators of $\mathbb{G}$. To ease notation, let $(r_1, u, \{g_i\}) \leftarrow \texttt{params}$ denote the event that $r_1 \leftarrow \mathbb{Z}_2^k$, $u \leftarrow \mathbb{Z}_p^k$, and $\{g_i\}_{i=1\ldots k} \leftarrow \mathbb{G}^k$. If there is a distinguisher $\mathcal{A}'$ that runs in time t such that

$$\epsilon = |\mathbb{P}[(r_1, u, \{g_i\}) \leftarrow \texttt{params} : \mathcal{A}'(g_1, \ldots, g_k, \prod_{i=1}^{k} g_i^{r_{1,i}}, \boldsymbol{\phi}(r_1), u, \langle r_1, u \rangle) \Rightarrow 1]$$

$$-\mathbb{P}(r_1, u, \{g_i\}) \leftarrow \texttt{params}, w \leftarrow \mathbb{Z}_p : \mathcal{A}'(g_1, \ldots, g_k, \prod_{i=1}^{k} g_i^{r_{1,i}}, \boldsymbol{\phi}(r_1), u, w) \Rightarrow 1]|,$$

then there is an inverter $\mathcal{C}$ that runs in time $t' = \text{poly}(k, 2, 1/\epsilon)$ such that

$$\mathbb{P}[r_1 \leftarrow \mathbb{Z}_2^k : \mathcal{C}(\boldsymbol{\phi}(r_1)) \Rightarrow r_1] \geq \frac{\epsilon^3}{512 p^4 k}.$$

Proof. The proof is very similar to that of Theorem 1 of [23], so we highlight only the modifications. Before $\mathcal{C}$ runs the simulation he first chooses generators $g_1, \ldots, g_k$ because these must be provided to $\mathcal{A}'$ (but they are not needed in [23]). Next, $\mathcal{C}$ needs to calculate the value $\prod_{i=1}^{k} g_i^{r_{1,i}}$ since this must also be given to $\mathcal{A}'$ (but is also not needed in [23]). Unfortunately, $\mathcal{C}$ does not have the required information to do this, so he randomly guesses an element $g^* \in \mathbb{G}$. Everything then proceeds as in the proof of Theorem 1 of [23], except when $\mathcal{C}$ is required to run $\mathcal{A}'$ he runs this algorithm with the extra inputs $g_1, \ldots, g_k, g^*$ (as well as the inputs that were required in [23]). If $\mathcal{C}$'s guess g^* for $\prod_{i=1}^{k} g_i^{r_{1,i}}$ is correct then $\mathcal{C}$ will provide tuples of the correct form to $\mathcal{A}'$. Hence, we condition on the probability that the guess is correct ($1/p$, the size of the group) and when the guess is correct we may use the same

argument as Theorem 1 of [23] to bound $\mathcal{C}$'s advantage (notice that we ignore the probability of inverting when the guess is incorrect because the probability is always greater than or equal to zero and we only need a lower bound for $\mathcal{C}$). We obtain:

$$\begin{aligned}
\mathbb{P}[\mathcal{C}(\boldsymbol{\phi}(r_1)) \Rightarrow r_1] &= \frac{1}{p} \cdot \mathbb{P}[\mathcal{C}(\boldsymbol{\phi}(r_1)) \Rightarrow r_1 \mid g^* = \prod_{i=1}^{k} g_i^{r_{1,i}}] \\
&\quad + \frac{p-1}{p} \cdot \mathbb{P}[\mathcal{C}(\boldsymbol{\phi}(r_1)) \Rightarrow r_1 \mid g^* \neq \prod_{i=1}^{k} g_i^{r_{1,i}}] \\
&> \frac{1}{p} \cdot \mathbb{P}[\mathcal{C}(\boldsymbol{\phi}(r_1)) \Rightarrow r_1 \mid g^* = \prod_{i=1}^{k} g_i^{r_{1,i}}] \\
&\geq \frac{|\mathbb{P}[\mathcal{A}'_{T_1} \Rightarrow 1] - \mathbb{P}[\mathcal{A}'_{T_2} \Rightarrow 1]|^3}{512p^4k} \\
&= \frac{\epsilon^3}{512p^4k}.
\end{aligned}$$

□

Here, the first line conditions on g^* being correct or not. The second line ignores the (positive) probability of inverting when g^* is not correct. The third line uses Theorem 6.1.1, and the final line follows from the fact that the advantage of A' is defined to be ϵ. Since $\mathcal{A}'$ has the same advantage as $\mathcal{A}$, we may conclude that

$$\mathbb{P}[\mathcal{C}(\boldsymbol{\phi}(r_1)) \Rightarrow r_1] > \frac{|\mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1]|^3}{512p^4k}.$$

Furthermore, we must have

$$|\mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1]| < \sqrt[3]{512\delta p^4 k}$$

because otherwise $\mathcal{C}$ would invert the vector of functions with probability greater than δ, which is impossible by assumption. □

$\mathbf{G_2}$ – $\mathbf{G_3}$: We will prove the following:

Lemma 6.1.4. For any adversary $\mathcal{A}$, there exists a KDF adversary $\mathcal{D}$ such that

$$\mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_3^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}^{\mathrm{kdf}}_{f,\mathcal{D}}(\lambda).$$

Proof. The KDF adversary $\mathcal{D}$ against f with access to $\boldsymbol{\phi}$ will simulate either the game G_2 or G_3 for $\mathcal{A}$. Adversary $\mathcal{D}$ is given as input a value Z that is either a uniformly random value from the range of the KDF or an output of the KDF on a uniformly random value from the domain of the KDF. Adversary $\mathcal{D}$ simulates **proc. Initialise** in the $\boldsymbol{\phi}$-FV-RRA-CPA security game by setting $b \leftarrow_{\$} \{0,1\}$, $r_1 \leftarrow_{\$} \{0,1\}^k$ and by choosing and forwarding $pk^* = (g_1, \ldots, g_k, g^{u_1}, \ldots, g^{u_k})$ to $\mathcal{A}$. Adversary $\mathcal{D}$ then answers $\mathcal{A}$'s oracle queries as follows:

Enc query (pk, m, i)

return $\mathtt{mBHHO.E}(pk, m; \phi(r_1))$ to $\mathcal{A}$.

LR query (m_0, m_1)

$c_1 \leftarrow \prod_{i=1}^{k} g_i^{r_{1,i}}$

$(K, r) \leftarrow Z$

$r' \leftarrow F_r(pk^* || m_b)$

$c_2 = \mathtt{DEM.E}(K, m_b; r')$

return (c_1, c_2) to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs a bit b', $\mathcal{D}$ halts and outputs 1 if and only if $b = b'$. When $\mathcal{D}$ is given a random Z, it simulates G_3 perfectly. Otherwise, it provides a perfect simulation for G_2. Hence,

$$\begin{aligned} \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_3^{\mathcal{A}} \Rightarrow 1] &= \mathbb{P}[\mathrm{KDFReal}_f^{\mathcal{D}}(\lambda) \Rightarrow 1] - \mathbb{P}[\mathrm{KDFRand}_{\$}^{\mathcal{D}}(\lambda) \Rightarrow 1] \\ &= \mathbf{Adv}^{\mathrm{kdf}}_{f,\mathcal{D}}(\lambda). \end{aligned}$$

□

$\mathbf{G_3}$ – $\mathbf{G_4}$: Notice that we now have a uniformly random output from the KDF

in the **LR** queries, which means we now have a uniformly random key for the PRF in these queries. Hence, we can prove the following:

Lemma 6.1.5. For any adversary $\mathcal{A}$, there exists a PRF adversary $\mathcal{E}$ such that

$$\mathbb{P}[G_3^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_4^{\mathcal{A}} \Rightarrow 1] \leq \mathbf{Adv}^{\mathrm{prf}}_{F,\mathcal{E}}(\lambda).$$

Proof. The PRF adversary $\mathcal{E}$ will simulate either G_3 or G_4 for $\mathcal{A}$. Adversary $\mathcal{E}$ simulates **proc. Initialise** in the ϕ-FV-RRA-CPA security game by setting $b \leftarrow_{\$} \{0,1\}$, $r_1 \leftarrow_{\$} \{0,1\}^k$, $K \leftarrow_{\$} \mathtt{DEM.K}(\lambda)$ and by choosing and forwarding $pk^* = (g_1, \ldots, g_k, g^{u_1}, \ldots, g^{u_k})$ to $\mathcal{A}$. Adversary $\mathcal{E}$ then answers $\mathcal{A}$'s oracle queries as follows:

Enc query (pk, m, i)

return $\mathtt{mBHHO.E}(pk, m; \phi_i(r_1))$ to $\mathcal{A}$.

LR query (m_0, m_1)

$c_1 = \prod_{i=1}^{k} g_i^{r_{1,i}}$

forward $pk^*||m_b$ to $\mathcal{E}$'s PRF oracle, receiving output r^*

$c_2 = \mathtt{DEM.E}(K, m_b; r^*)$

return (c_1, c_2) to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs b', $\mathcal{E}$ halts and outputs 1 if and only if $b = b'$. When $\mathcal{E}$ is playing the game PRFReal (in which case the oracle outputs are those of the PRF), he simulates G_3 perfectly. Otherwise, when $\mathcal{E}$ is playing the game PRFRand (and the oracle outputs are uniformly random), he simulates G_4 perfectly. Hence,

$$\begin{aligned} \mathbb{P}[G_3^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_4^{\mathcal{A}} \Rightarrow 1] &= \mathbb{P}[\mathrm{PRFReal}_F^{\mathcal{E}}(\lambda) \Rightarrow 1] - \mathbb{P}[\mathrm{PRFRand}_{\$}^{\mathcal{E}}(\lambda) \Rightarrow 1] \\ &= \mathbf{Adv}^{\mathrm{prf}}_{F,\mathcal{E}}(\lambda). \end{aligned}$$

□

Finally, since in handling $\mathcal{A}$'s **LR** queries, the outputs of the KDF f and the

PRF F have now both been replaced with uniformly random values, we have uniformly random K and r''. Hence, the game G_4 may be simulated by a standard IND-CPA DEM adversary, $\mathcal{F}$. More formally:

Lemma 6.1.6. For any adversary $\mathcal{A}$, there exists an IND-CPA DEM adversary $\mathcal{F}$ such that

$$2 \cdot \mathbb{P}[G_4^{\mathcal{A}} \Rightarrow 1] - 1 \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathtt{DEM},\mathcal{F}}(\lambda).$$

Proof. The adversary $\mathcal{F}$ will simulate G_4 for $\mathcal{A}$. Adversary $\mathcal{F}$ chooses $r_1 \leftarrow_{\$} \{0,1\}^k$ and generates a public key pk^* of the form $(g_1, \ldots, g_k, g^{u_1}, \ldots, g^{u_k})$. Adversary $\mathcal{F}$ gives pk^* to $\mathcal{A}$, and answers $\mathcal{A}$'s oracles queries as follows:

Enc query (pk, m, i)

return $\mathtt{mBHHO.E}(pk, m; \phi_i(r_1))$.

LR query (m_0, m_1)

$c_1 = \prod_{i=1}^{k} g_i^{r_{1,i}}$

forward (m_0, m_1) to $\mathcal{F}$'s encryption oracle, receiving as output c_2

return (c_1, c_2) to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs b', $\mathcal{F}$ halts and outputs b'. We conclude that

$$2 \cdot \mathbb{P}[G_4^{\mathcal{A}} \Rightarrow 1] - 1 \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathtt{DEM},\mathcal{F}}(\lambda).$$

□

The theorem follows by combining all these inequalities. □

The class of related randomness functions which our scheme `mBHHO` can tolerate is quite different from those in our previous constructions: bounded-degree polynomials are certainly not hard-to-invert in general. Our proof of Theorem 6.1.2 actually shows that even if $\boldsymbol{\phi}(r)$ were to completely leak to the adversary (instead of merely being indirectly accessible via **Enc** queries), the

scheme mBHHO would still be secure. This would not be the case if the analogous $\phi(r)$ values were to leak in our earlier schemes PRF-PKE and CI-Hash-PKE, since the adversary could actually reconstruct r from this leakage for the relevant ϕ functions and win the security game. Furthermore, the functions are not required to be collision-resistant or output-unpredictable. These restrictions are only strictly required of the functions queried to the **LR** oracle. However, since an adversary is restricted to using only the identity function (which *is* collision-resistant and output-unpredictable) in its **LR** queries, the functions in $\mathbf{\Phi}$ do not need to satisfy these conditions.

6.2 Goldreich-Levin Theorem for Large Fields

In this section we will discuss and repair what we believe to be an error in the proof of the Goldreich-Levin theorem for large fields that was presented in [23]. Recall that we used the repaired version in the proof of Theorem 6.1.2. For a fixed m and a prime p, the proof requires that the adversary find the smallest natural number c such that $p^c > m$. Then it is necessary to find a subset $S \subset \mathbb{Z}(p)^c \backslash \{0^c\}$ of cardinality m such that all elements of S are pairwise linearly independent. However, when p^c and m are too close (in the sense of their difference), it will not be possible to find such a set. For example, the authors of [23] suggest setting the first coordinate of every element to be 1, and then choosing all other coordinates arbitrarily. Unfortunately, using this method there are only p^{c-1} possible vectors (since the first component is fixed, and the other $c-1$ components can take values from 0 to $p-1$). Therefore, if $m > p^{c-1}$ this method will be unable to yield a set S of cardinality m. Notice also that we almost always have $m > p^{c-1}$, since by definition of c we know $m \geq p^{c-1}$. As an extreme example, consider what happens if $p^c = m + 1$. Clearly finding $m = p^c - 1$ pairwise linearly independent vectors

in $\mathbb{Z}(p)^c \backslash \{0^c\}$ (which only contains $p^c - 1$ distinct vectors) is impossible. One way to circumvent this issue is to insist that c be the smallest value such that $p^{c-1} > m$. It is then possible to find m pairwise linearly independent vectors from $S \subset \mathbb{Z}(p)^c \backslash \{0^c\}$ by using the method outlined above (setting the first coordinate of every vector to be 1, and then choosing all other coordinates arbitrarily).

The success analysis of the Goldreich-Levin algorithm in [23] shows that the adversary $\mathcal{A}$ will succeed in recovering the input to the leakage function with probability $\epsilon/4p^c$. Then, since c is defined to be the smallest integer such that $p^c > m$, we know that $p^{c-1} \leq m$. This allows the authors to deduce that

$$4p^c = 4p^{c-1} \cdot p \leq 4mp$$

and then m is bounded to give the desired result. With the modification outlined above, we instead have that c is the smallest integer such that $p^{c-1} > m$, which means that $p^{c-2} \leq m$. We would then have

$$4p^c = 4p^{c-2} \cdot p^2 \leq 4mp^2.$$

The value m is fixed, so is bounded in exactly the same way. Hence, the only modification to the theorem statement is that an extra p appears in the denominator of the bound. That is, instead of the success of the inverter being at least

$$\frac{\epsilon^3}{512np^2},$$

it should be

$$\frac{\epsilon^3}{512np^3}.$$

6.3 Generalised FV-RRA Security

In this section we extend the results of the previous section by providing a more generic approach to achieving FV-RRA-ATK security. In addition, we

will demonstrate how to achieve the CCA notion of function-vector security (recall that our scheme in the previous section only attained the CPA notion). Furthermore, we will provide a transform that will convert any IND-ATK secure scheme into an FV-RRA-ATK secure scheme. We shall also generalise the notion of FV-RRA-ATK security to allow an adversary to use functions in his **LR** queries (recall that in the original game the adversary was restricted to using only the identity function for **LR** queries). We prove a general result in this extended attack model, but unfortunately we are unable to provide any concrete instantiations that are secure according to this strong notion. As a further contribution, we will explore connections between reconstructive extractors and CIS hash functions, which were introduced in the previous chapter.

The transform achieving FV-RRA-ATK security makes use of a technical tool called an *auxiliary-input reconstructive extractor*. Classically, an *extractor* is a function `Ext`, which, given an input and a seed, produces an output that is statistically indistinguishable from elements chosen uniformly at random from some set Σ, provided the input is chosen from a distribution with sufficient min-entropy and the seed is chosen uniformly at random. A *reconstructive* extractor is an extractor with the additional property that, roughly speaking, allows the efficient reconstruction of the input x from any distinguisher $\mathcal{D}$ that successfully distinguishes the output of the extractor from random. This is formalised in terms of the existence of an oracle machine `Rec` outputting x. Then an *auxiliary-input* reconstructive extractor is a reconstructive extractor in which the output still remains indistinguishable when the distinguisher $\mathcal{D}$ is also given access to the output of a leakage function $h(\cdot)$ on input x. Our actual definition (Definition 6.3.4) extends this idea further still: the distinguisher $\mathcal{D}$ is given either a set of uniformly random values or the set of outputs of the extractor when evaluated on $\phi(x)$ for all $\phi \in \boldsymbol{\phi}$, where $\boldsymbol{\phi}$ is a vector of

functions defined by the game.

Equipped with an auxiliary-input reconstructive extractor, our transform to achieve FV-RRA-ATK security is conceptually simple:

- We append a uniformly random extractor seed to each public key.
- The encryption algorithm consumes a random value r from some set of bit strings; this is fed into the extractor (using the seed from the public key). The output of the extractor is used as a key for a PRF, and the input to the PRF is the public key appended with the message. Finally, the output of the PRF is used as the actual randomness for encryption, and we simply encrypt with the original encryption algorithm.
- Decryption works exactly as in the original decryption algorithm.

Intuitively, a challenge encryption constructed using randomness value $\phi(r)$ remains secure, since the extractor guarantees an output indistinguishable from random, even when the adversary gains access to encryptions under the related randomness values $\phi'(r)$. Hence, the PRF, which uses the extractor output as a key, will guarantee that independent randomness values are used for different public key and messages pairs, which implies that the adversary is forced to break the security of the underlying PKE scheme to learn anything about the encrypted challenge messages. That this approach attains FV-RRA-ATK security is formally proven in Theorem 6.3.1.

The schemes obtained from using our transform with this extractor have significant benefits compared to the concrete FV-RRA-CPA-secure scheme from Section 6.1. For example, we obtain shorter public keys and a tighter security reduction compared to the scheme from Section 6.1. Most importantly, we obtain FV-RRA-CCA security in a completely generic way.

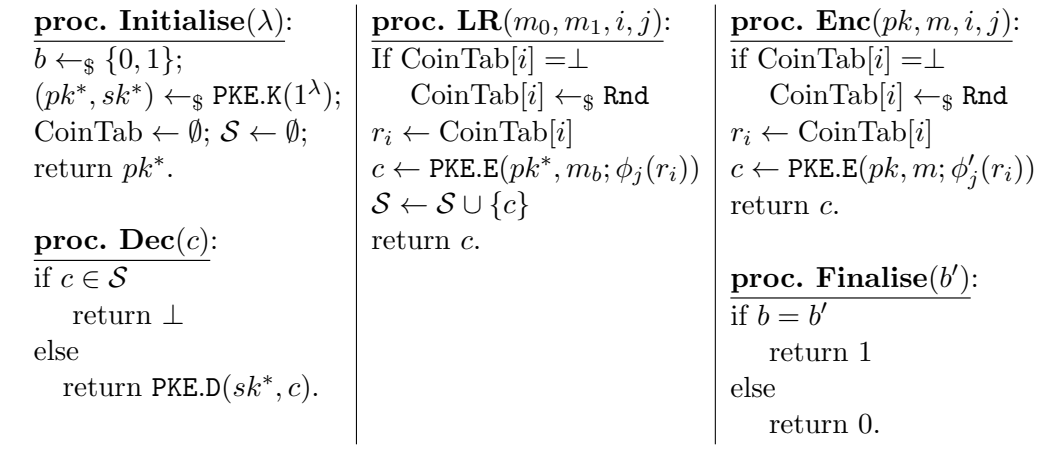

Figure 6.3: Game $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK, where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ and $\boldsymbol{\phi}' = (\phi'_1, \ldots, \phi'_{q'})$. (If ATK = CPA, then the adversary's access to **proc. Dec** is removed.)

6.3.1 Extended function-vector related randomness security

In this section we will introduce the new definitions that aim to generalise the FV-RRA-ATK notion from Section 5.2. Our generalisation will allow an adversary to manipulate the randomness used for the **LR** queries, instead of being restricted to using only the identity function. The security game for our new notion is in Figure 6.3. The major difference is that the game is parametrised by two sets of functions, $\boldsymbol{\phi}$ and $\boldsymbol{\phi}'$. An adversary may only use functions from $\boldsymbol{\phi}$ in its **LR** queries, and the functions in $\boldsymbol{\phi}'$ may only be used for **Enc** queries. Notice that if $\boldsymbol{\phi} = \{\mathrm{id}\}$, then this definition simplifies to the basic FV-RRA-ATK notion. This simplification will be needed when we discuss an extension of standard auxiliary-input reconstructive extractors in Section 6.3.2.

The following definition is an adaptation of Definition 5.2.1. It has been modified to be applicable to the generalised function-vector setting. Recall that this definition captures natural restrictions that must be placed on an adversary in order to prevent trivial wins.

Definition 6.3.1. Let $\mathcal{A}$ be an adversary in Game $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK that queries r different randomness indices to its **LR** and **Enc** oracles and makes $q_{i,\phi}$ queries to its **LR** oracle with index i and function $\phi \in \boldsymbol{\Phi}$. Let $(m_0^{i,\phi,1}, m_1^{i,\phi,1}), \ldots, (m_0^{i,\phi,q_{i,\phi}}, m_1^{i,\phi,q_{i,\phi}})$ be $\mathcal{A}$'s **LR** queries for index $i \in [r]$ and $\phi \in \boldsymbol{\Phi}$. Suppose that for all pairs $(i, \phi) \in [r] \times \boldsymbol{\Phi}$ and for all $j \neq k \in [q_{i,\phi}]$, we have

$$m_0^{i,\phi,j} = m_0^{i,\phi,k} \text{ iff } m_1^{i,\phi,j} = m_1^{i,\phi,k}.$$

Then we say that $\mathcal{A}$ is *equality-pattern respecting.*

Note that any adversary that is not equality-pattern respecting can trivially win the game in Figure 6.3. More specifically, the adversary can simply queries its **LR** oracle with the tuples (m_0, m_1, i, j) and (m_0, m_2, i, j), where m_0, m_1 and m_2 are all distinct. The values i and j can be arbitrary values from the appropriate domain. If the bit b is equal to 0, the adversary will receive identical ciphertexts, whereas the ciphertexts will differ if b equals 1. This results in a trivial win for an adversary. In contrast, an equality-respecting adversary cannot exploit the available oracles in this particular way, and is forced to mount a non-trivial attack against the scheme to win the security game.

With the above definition in place, we can now formally define our generalised notion of FV-RRA-ATK security.

Definition 6.3.2. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ and $\boldsymbol{\phi}' = (\phi'_1, \ldots, \phi'_{q'})$ be vectors of $q := q(\lambda)$ and $q' := q'(\lambda)$ functions respectively. We define the advantage of an equality-pattern respecting, $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK adversary $\mathcal{A}$ against a PKE scheme $\mathtt{PKE}$ to be

$$\mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-fv-rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[(\boldsymbol{\phi}, \boldsymbol{\phi}')\text{-FV-RRA-ATK}^{\mathcal{A}}_{\mathtt{PKE}}(\lambda) \Rightarrow 1] - 1.$$

If $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}'$ are sets of vectors of functions, then a PKE scheme $\mathtt{PKE}$ is said to be $(\boldsymbol{\Phi}, \boldsymbol{\Phi}')$-FV-RRA-ATK secure if, for all $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ and for all $\boldsymbol{\phi}' \in \boldsymbol{\Phi}'$, the

advantage of any equality-pattern respecting, $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK adversary against PKE that runs in polynomial time is negligible in the security parameter λ.

Similar to the notion defined in Section 5.2 it is possible to reduce the above defined FV-RRA-ATK security to a simpler notion in which the security game involves only a single uniformly random value used in all oracle queries. The following lemma follows easily from Lemma 5.2.1 of Section 5.2 and is therefore presented without a proof.

Lemma 6.3.1. Consider an equality-pattern respecting, $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK adversary $\mathcal{A}$ that queries q_r distinct randomness indices and makes at most q_{LR} **LR** queries. Then there exists an equality-pattern respecting, $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK adversary $\mathcal{B}$ that queries at most one randomness index and makes at most q_{LR} **LR** queries such that

$$\mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-fv-rra-atk}}_{\mathtt{PKE},\mathcal{A}}(\lambda) \leq q_r \cdot \mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-fv-rra-atk}}_{\mathtt{PKE},\mathcal{B}}(\lambda),$$

where $\mathcal{B}$ runs in approximately the same time as $\mathcal{A}$. In the CCA setting, $\mathcal{B}$ makes the same number of decryption queries as $\mathcal{A}$.

6.3.2 Obtaining FV-RRA security from auxiliary-input reconstructive extractors

In this section we will explore the main result of this section. We will improve upon the approach of Section 6.1 by proposing a transform that converts any IND-ATK scheme into an FV-RRA-ATK scheme via the use of an auxiliary-input reconstructive extractor. We then provide an instantiation of our transform that not only meets the stronger FV-RRA-CCA notion, but also provides shorter public keys and a tighter security reduction compared to the scheme from Section 6.1.

Before introducing the extractors we utilise in our transform, we first need to define the notion of a vector of functions being δ-hard-to-compute with respect to another vector of functions.

Definition 6.3.3. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ and $\boldsymbol{\phi}' = (\phi'_1, \ldots, \phi'_{q'})$ denote vectors of functions on a set $\mathtt{Rnd}_\lambda$, where $q := q(\lambda)$ and $q' := q'(\lambda)$ are polynomial in the security parameter λ. Let $\delta(\lambda)$ be a function. We say that $\boldsymbol{\phi}$ is $\delta(\lambda)$-*hard-to-compute with respect to* $\boldsymbol{\phi}'$ if, for all polynomial time algorithms $\mathcal{A}$ and all sufficiently large λ, we have

$$\mathbb{P}[\phi_i(r) \leftarrow \mathcal{A}(\phi'_1(r), \ldots, \phi'_q(r)) : r \leftarrow_{\$} \mathtt{Rnd}_\lambda] \leq \delta(\lambda),$$

for all $i \in \{1, \ldots, q\}$. We say that a set of vectors of functions $\boldsymbol{\Phi}$ is δ-hard-to-compute with respect to $\boldsymbol{\Phi}'$ if each vector $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ is δ-hard-to-compute with respect to every vector in $\boldsymbol{\Phi}'$ (note that the vectors in such a set $\boldsymbol{\Phi}$ need not all be of the same dimension, but we assume they each have dimension that is polynomial in λ). If $\delta = \mathrm{negl}(\lambda)$, then we simply say that $\boldsymbol{\Phi}$ is hard-to-compute with respect to $\boldsymbol{\Phi}'$.

With this definition to hand, we may now introduce our generalised definition of an auxiliary-input reconstructive extractor.

Definition 6.3.4. An $(\epsilon, \delta, \boldsymbol{\Phi}, \boldsymbol{\Phi}')$-*auxiliary-input reconstructive extractor* is a pair of functions $(\mathtt{Ext}, \mathtt{Rec})$ such that $\mathtt{Ext}$ is an extractor that maps from $\{0,1\}^n \times \{0,1\}^d$ to Σ, and $\mathtt{Rec}$ is an oracle machine that on input $(1^n, 1/\epsilon)$ runs in time $\mathrm{poly}(n, 1/\epsilon, \log(|\Sigma|))$. Furthermore, for every $x \in \{0,1\}^n$, every $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q) \in \boldsymbol{\Phi}$, every $\boldsymbol{\phi}' \in \boldsymbol{\Phi}'$, and every function $\mathcal{D}$ such that

$$\begin{aligned}|\mathbb{P}_{s \leftarrow_{\$} \{0,1\}^d}[\mathcal{D}(s, \{\mathtt{Ext}(\phi_i(x), s)\}_{i \in \{1,\ldots,q\}}, \boldsymbol{\phi}'(x)) = 1] - \\ \mathbb{P}_{\substack{s \leftarrow_{\$} \{0,1\}^d \\ \sigma_i \leftarrow_{\$} \Sigma}}[\mathcal{D}(s, \{\sigma_i\}_{i \in \{1,\ldots,q\}}, \boldsymbol{\phi}'(x)) = 1]| \geq \epsilon\end{aligned}$$

we require that

$$\mathbb{P}[\mathtt{Rec}^{\mathcal{D}}(1^n, 1/\epsilon, \boldsymbol{\phi}'(x)) = \phi_i(x)] \geq \delta$$

Alg. $\texttt{EXT-PKE.K}(1^\lambda)$:	Alg. $\texttt{EXT-PKE.E}(\hat{pk}, m)$:	Alg. $\texttt{EXT-PKE.D}(\hat{sk}, c)$:
$(pk, sk) \leftarrow \texttt{PKE.K}(1^\lambda)$	$r \leftarrow_\$ \texttt{Rnd}$	$m \leftarrow \texttt{PKE.D}(sk, c)$
$s \leftarrow \texttt{seeds}$	$K \leftarrow \texttt{Ext}(r, s)$	return m.
$\hat{pk} \leftarrow (pk, s)$	$r' \leftarrow F_K(\hat{pk}\|\|m)$	
$\hat{sk} \leftarrow (sk)$	$c \leftarrow \texttt{PKE.E}(pk, m; r')$	
return $\hat{pk}$.	return c.	

Figure 6.4: Scheme EXT-PKE built from a reconstructive extractor, a PKE scheme PKE, and a PRF F.

for some $i \in \{1, \ldots, q\}$, where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$, $q := q(\lambda)$ is polynomial, and the probability is over the coin tosses of Rec. If, for every $\mathcal{D}$ with non-negligible ϵ, Rec reconstructs $\phi_i(x)$ with non-negligible probability, we may simply say that $(\texttt{Ext}, \texttt{Rec})$ is a $(\boldsymbol{\Phi}, \boldsymbol{\Phi}')$-auxiliary-input reconstructive extractor.

Armed with this new definition of an auxiliary-input reconstructive extractor, we are ready to prove the main theorem which will establish the main result of this section. We show that any extractor satisfying Definition 6.3.4 can be used in conjunction with an IND-ATK secure PKE scheme and a PRF to meet the FV-RRA-ATK security notion in Figure 6.3. The encryption scheme that achieves this result is in Figure 6.4. The algorithm works by appending a uniformly random extractor seed to each public key, but leaving the private key unmodified. The encryption algorithm generates a uniformly random r, which is then fed into the extractor (using the seed from the public key). The output of the extractor is used as a key for a PRF, and the input to the PRF is the public key appended with the message. Finally, the output of the PRF is used as the new randomness for encryption, and then we simply encrypt with the standard encryption algorithm.

Theorem 6.3.1. If $\boldsymbol{\Phi}$ is hard-to-compute with respect to $\boldsymbol{\Phi}'$ and $(\texttt{Ext}, \texttt{Rec})$ is an $(\boldsymbol{\Phi}, \boldsymbol{\Phi}')$-auxiliary-input reconstructive extractor, then the PKE scheme EXT-PKE in Figure 6.4 is $(\boldsymbol{\Phi}, \boldsymbol{\Phi}')$-FV-RRA-ATK secure when instantiated with a secure PRF and an IND-ATK secure PKE scheme PKE. More precisely,

consider any polynomial-size vectors of functions $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ and $\boldsymbol{\phi}' \in \boldsymbol{\Phi}'$, any $(\epsilon, \delta, \boldsymbol{\Phi}, \boldsymbol{\Phi}')$-auxiliary-input reconstructive extractor $(\mathtt{Ext}, \mathtt{Rec})$, and any equality-pattern respecting, $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-FV-RRA-ATK adversary $\mathcal{A}$ against the scheme EXT-PKE. Suppose $\mathcal{A}$ makes q_{LR} **LR** queries and uses q_r randomness indices. Then, either $\boldsymbol{\Phi}$ is not δ-hard-to-compute with respect to $\boldsymbol{\Phi}'$, or there exists a PRF adversary $\mathcal{B}$, and an IND-ATK adversary $\mathcal{C}$, all running in polynomial time, such that

$$\mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-fv-rra-atk}}_{\mathtt{EXT\text{-}PKE},\mathcal{A}}(\lambda) < 2q_r \cdot q \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{B}}(\lambda) + q_r \cdot q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\mathtt{PKE},\mathcal{C}}(\lambda) + 2q_r\epsilon.$$

Proof. First, we invoke Lemma 6.3.1, so that we now only have to prove the theorem for an adversary using just one randomness value. Furthermore, we assume that an adversary never repeats a query. Identical queries will result in identical outputs, hence any adversary that repeats a query may be replaced by a more efficient adversary with the same advantage. We proceed via a sequence of game hops. The games are as follows:

$\mathbf{G_0}$: G_0 is the real game with the scheme defined in Figure 6.4.

$\mathbf{G_1}$: G_1 is the same as G_0, except for **LR** queries, where the output of the extractor is replaced with a uniformly random value. If there existed an adversary that could distinguish between these two games, then we could use this adversary to compute one of the outputs of $\phi_i(r)$.

$\mathbf{G_2}$: G_2 is the same as G_1, except for **LR** queries, where the outputs of the PRF are replaced with uniformly random values. Finally, G_2 may be simulated by a standard IND-ATK adversary.

$\mathbf{G_0}$ – $\mathbf{G_1}$: We will prove the following:

Lemma 6.3.2. For any adversary $\mathcal{A}$, the difference in success probabilities in

games G_0 and G_1 is bounded by ϵ:

$$\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] < \epsilon.$$

Proof. Consider a distinguisher $\mathcal{D}$, attempting to distinguish whether the extractor game has returned a real or random output. Adversary $\mathcal{D}$ is given a seed s, two vectors of functions $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ and $\boldsymbol{\phi}' = (\phi'_1, \ldots, \phi'_{q'})$, the vector $\boldsymbol{\phi}' = (\phi'_1(r), \ldots, \phi'_{q'}(r))$, and a vector $\mathbf{z}$ which is either $\{\texttt{Ext}(\phi_i(r); s)\}_{i \in \{1,\ldots,q\}}$ or $\{\sigma_1, \ldots, \sigma_q\}$ where each σ_i is chosen uniformly at random from the range Σ of the extractor. The distinguisher $\mathcal{D}$ sets-up the simulation as follows:

Setup

$(pk^*, sk^*) \leftarrow \texttt{PKE.K}(1^\lambda)$

$(\hat{pk}^*, \hat{sk}^*) \leftarrow (pk^* || s, sk^*)$

$b \leftarrow_\$ \{0, 1\}$.

Adversary $\mathcal{C}$ forwards the public key $\hat{pk}^*$ to $\mathcal{A}$. Then $\mathcal{D}$ answers $\mathcal{A}$'s queries as follows:

Enc query $(\hat{pk}, m, i)$

$\mathcal{D}$ returns $\texttt{EXT-PKE.E}(\hat{pk}, m; \phi'_i(r))$ to $\mathcal{A}$.

LR query (m_0, m_1, i)

$r' \leftarrow F_{z_i}(\hat{pk}^* || m_b)$

$c \leftarrow \texttt{PKE.E}(pk^*, m_b; r')$

return c to $\mathcal{A}$.

Dec query c

return $\texttt{PKE.D}(sk^*, c)$.

When $\mathcal{A}$ halts and outputs b', $\mathcal{C}$ halts and outputs 1 if and only if $b = b'$. If $\mathbf{z} = \{\texttt{Ext}(\phi_i(r); s)\}_{i \in \{1,\ldots,q\}}$ a perfect simulation of G_0 is provided. Otherwise,

a perfect simulation of G_1 is provided. Hence,

$$\begin{aligned}&|\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1]| \\ &= |\mathbb{P}[\mathcal{D}(s, \{\texttt{Ext}(\phi_i(r); s)\}, \boldsymbol{\phi}'(r)) \Rightarrow 1] - \mathbb{P}[\mathcal{D}(s, \{\sigma_i\}_{i\in\{1,\ldots,q\}}, \boldsymbol{\phi}'(r)) \Rightarrow 1]|.\end{aligned}$$

If an adversary can distinguish outputs of the extractor from uniformly random values with probability greater than or equal to ϵ in polynomial time, then there exists an algorithm $\texttt{Rec}$ that will output $\phi_i(r)$ (for some $i \in \{1, \ldots, q\}$) with probability greater than δ (cf. Definition 6.3.4). However, this is a contradiction since the $\boldsymbol{\phi}$ is δ-hard-to-compute with respect to $\boldsymbol{\phi}'$. Hence, we must have

$$\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] < \epsilon.$$

□

$\mathbf{G_1}$ – $\mathbf{G_2}$: If an adversary can distinguish games 1 and 2 with a certain probability, then we may construct a PRF adversary $\mathcal{B}$ that wins the PRF game with the same probability. Specifically:

Lemma 6.3.3. The difference between games G_1 and G_2 is bounded by the advantage of a PRF adversary. More specifically,

$$\mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1] \leq q \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{B}}(\lambda).$$

Proof. The proof uses a hybrid argument. Consider the hybrid game $G_{1,j}$, where the outputs of the PRF for functions $\{\phi_i\}_{i<j+1}$ are replaced with uniformly random values. Notice that $G_1 = G_{1,0}$ and $G_2 = G_{1,q}$. The setup procedure of $\mathcal{B}$ is as follows:

Setup

$b \leftarrow_{\$} \{0, 1\}$

$j \leftarrow_{\$} \{0, \ldots, q-1\}$

$s \leftarrow_{\$} \texttt{seeds}$

$r \leftarrow_{\$} \texttt{Rnd}$

$K_{j+2}, \ldots, K_q \leftarrow_{\$} \texttt{PRF.K}(1^\lambda)$

$(pk^*, sk^*) \leftarrow \texttt{PKE.K}(1^\lambda)$

$(\hat{pk}^*, \hat{sk}^*) \leftarrow (pk^*||s, sk^*)$.

The adversary $\mathcal{B}$ returns the public key $\hat{pk}^*$ to $\mathcal{A}$, and answers $\mathcal{A}$'s queries as follows:

Enc query $(\hat{pk}, m, i)$

return $\texttt{EXT-PKE.E}(\hat{pk}, m; \phi'_i(r))$ to $\mathcal{A}$.

LR query (m_0, m_1, i)

if $i < j + 1$

$\mathcal{B}$ chooses uniformly random r'

if $i = j + 1$

forward $pk^*||m_b$ to the PRF oracle; $\mathcal{B}$ receives output r'

if $i > j + 1$

quad compute $r' \leftarrow F_{K_i}(pk^*||m_b)$

$c \leftarrow \texttt{PKE.E}(\hat{pk}^*, m_b; r')$

return c to $\mathcal{A}$.

Dec query c

return $\texttt{PKE.D}(sk, c)$.

When $\mathcal{A}$ halts and outputs b', $\mathcal{B}$ halts and outputs 1 if and only if $b = b'$. When $\mathcal{B}$'s outputs are from a PRF he simulates $G_{1,j}$ perfectly. Otherwise, if $\mathcal{B}$'s outputs are uniformly random he simulates $G_{1,j+1}$ perfectly. Hence, since there are q hybrid games, we have

$$\mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1] - \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1] \leq q \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{B}}(\lambda).$$

□

Finally, the success in game G_2 may be related to the success of a standard IND-ATK PKE adversary, $\mathcal{C}$. More formally:

Lemma 6.3.4. The success probability of an adversary attacking G_2 can be bound as follows:

$$2 \cdot \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1] - 1 \leq q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda).$$

Proof. The adversary $\mathcal{C}$ will simulate G_2 for $\mathcal{A}$. To setup, adversary $\mathcal{C}$ runs the following procedure:

Setup

$r \leftarrow_{\$} \texttt{Rnd}$

$s \leftarrow_{\$} \texttt{seeds}$

$j \leftarrow_{\$} \{1, \ldots, q_{LR}\}$

ctr $\leftarrow 1$ $(pk^*, sk^*) \leftarrow \texttt{PKE.K}(1^\lambda)$

$(\hat{pk}^*, \hat{sk}^*) \leftarrow (pk^* || s, sk^*)$.

Adversary $\mathcal{C}$ returns $\hat{pk}^*$ to $\mathcal{A}$, and answers $\mathcal{A}$'s oracles queries as follows:

Enc query (pk, m, i)

return $\texttt{EXT-PKE.E}(pk, m; \phi'_i(r))$.

LR query (m_0, m_1, i)

if ctr $< j$,

$r' \leftarrow_{\$} \texttt{Rnd}$

$c \leftarrow \texttt{PKE.E}(pk, m_0; r')$

else if ctr $> j$,

$r' \leftarrow_{\$} \texttt{Rnd}$

$c \leftarrow \texttt{PKE.E}(pk, m_1; r')$

else submit (m_0, m_1) to $\mathcal{C}$'s **LR** oracle, receive c

ctr $\leftarrow$ ctr $+ 1$

return c to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs b', $\mathcal{C}$ halts and outputs b'. Via a hybrid argument we conclude that

$$2 \cdot \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1] - 1 \leq q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\mathtt{PKE},\mathcal{C}}(\lambda).$$

□

The theorem follows by combining all of these inequalities. □

6.3.3 Instantiation of an auxiliary-input reconstructive extractor

We have previously introduced a more general notion of FV-RRA-ATK security, and we have shown that the existence of an auxiliary-input extractor in the mould of Definition 6.3.4 would give rise to a scheme that meets this generalised security notion. It now remains to see what extractors exist that satisfy Definition 6.3.4. The strongest extractor we are aware of is the Goldreich-Levin extractor, whose properties are analysed in Theorem 1 of [23], and whose result was stated in Theorem 6.1.1 of this thesis. Recall that the theorem states the following (with the notation changed to remain consistent with ours, and also fixing the error, as discussed in Section 6.2):

Theorem. Let p be a prime, and let H be an arbitrary subset of $\mathbb{Z}_p$. Let $f : H^n \to \{0,1\}^*$ be any (possibly randomised) function. If there is a distinguisher $\mathcal{D}$ that runs in time t such that

$$\begin{aligned} &|\mathbb{P}[\mathbf{r} \leftarrow H^n, y \leftarrow f(\mathbf{r}), \mathbf{s} \leftarrow \mathbb{Z}_p^n : \mathcal{D}(y, \mathbf{s}, \langle \mathbf{r}, \mathbf{s} \rangle) = 1] \\ - \quad &\mathbb{P}[\mathbf{r} \leftarrow H^n, y \leftarrow f(\mathbf{r}), \mathbf{s} \leftarrow \mathbb{Z}_p^n, u \leftarrow \mathbb{Z}_p : \mathcal{D}(y, \mathbf{s}, u) = 1]| = \epsilon \end{aligned}$$

then there is an inverter $\mathcal{A}$ that runs in time $t' = t \cdot \text{poly}(n, |H|, 1/\epsilon)$ such that

$$\mathbb{P}[\mathbf{r} \leftarrow H^n, y \leftarrow f(\mathbf{r}) : \mathcal{A}(y) = \mathbf{r}] \geq \frac{\epsilon^3}{512 \cdot n \cdot p^3}. \tag{6.3.1}$$

Alg. EXT-PKE.K(1^λ):	Alg. EXT-PKE.E($\hat{pk}, m$):	Alg. EXT-PKE.D($\hat{sk}, c$):
$(pk, sk) \leftarrow$ PKE.K(1^λ)	$r \leftarrow_\$ H^\lambda$	$m \leftarrow$ PKE.D(sk, c)
$s \leftarrow \mathbb{Z}_p^\lambda$	$K \leftarrow \langle r, s \rangle$	return m.
$\hat{pk} \leftarrow (pk, s)$	$r' \leftarrow F_K(\hat{pk}\|\|m)$	
$\hat{sk} \leftarrow (sk)$	$c \leftarrow$ PKE.E($pk, m; r'$)	
return $\hat{pk}$.	return c.	

Figure 6.5: Scheme EIP-PKE (Euclidean Inner Product) built from a PKE scheme PKE, and a PRF F. Here, H denotes a subset of $\mathbb{Z}_q$.

This theorem can be used to obtain an auxiliary-input reconstructive extractor. Now, consider the extractor Ext that maps from $H^n \times \mathbb{Z}_p^n$ to $\mathbb{Z}_p$ (where H is a subset of $\mathbb{Z}_p$) defined as

$$\texttt{Ext}(r, s) = \langle r, s \rangle.$$

If we match the notation of Theorem 6.1.1 with Definition 6.3.4, then Rec is now $\mathcal{A}$, $\mathbf{\Phi} = \{\text{id}\}$, $\boldsymbol{\phi}'$ is the function f, $\mathbf{\Phi}'$ is the set of δ-hard-to-invert vectors of functions, and the extractor Ext is easily seen to be an $(\epsilon, \delta, \{\text{id}\}, \mathbf{\Phi}')$-auxiliary-input reconstructive extractor, where

$$\epsilon = \sqrt[3]{512\delta\lambda p^3}.$$

Note that, in the proof of Dodis et al., the theorem is stated with one function f. However, we now use a vector of functions $(\phi_1, \ldots, \phi_q)$ in our proof. Fortunately this is not problematic, since we can interpret f as a function of functions. That is, $f(r) = (\phi_1(r), \ldots, \phi_q(r))$.

By combining Theorem 6.1.1 with Theorem 6.3.1, we easily obtain the following theorem.

Theorem 6.3.2. Let $\mathbf{\Phi}$ be a set of hard-to-invert vectors of functions on $\{0,1\}^\lambda$. Then PKE scheme EIP-PKE in Figure 6.5 is $(\text{id}, \mathbf{\Phi})$-FV-RRA-ATK secure. More precisely, consider any polynomial-size vector of functions $\boldsymbol{\phi} \in \mathbf{\Phi}$ which is δ-hard-to-invert, and any equality-pattern respecting, $(\text{id}, \boldsymbol{\phi})$-FV-RRA-ATK

adversary $\mathcal{A}$ against EIP-PKE. Suppose $\mathcal{A}$ makes q_{LR} **LR** queries and uses q_r randomness indices. Then there exists a PRF adversary $\mathcal{B}$ and an IND-ATK adversary $\mathcal{C}$, all running in polynomial time, such that

$$\mathbf{Adv}^{(\mathrm{id},\boldsymbol{\phi})\text{-fv-rra-atk}}_{\texttt{EIP-PKE},\mathcal{A}}(\lambda) < 2q_r \cdot \mathbf{Adv}^{\mathrm{prf}}_{F,\mathcal{B}}(\lambda) + q_r \cdot q_{LR} \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{PKE},\mathcal{C}}(\lambda) + 2q_r \sqrt[3]{512\delta\lambda p^3}.$$

The above theorem does not achieve the strongest levels of security we desire since the challenge functions modifying the input to the extractor are limited to being the identity function, which is equivalent to the security game studied in Section 6.1. However, the schemes resulting from our transform using the above reconstructive extractor still enjoy many advantages over the scheme that was presented in Section 6.1. Most notably, Section 6.1 only gave one concrete scheme, which is only secure in the CPA version of the FV-RRA-ATK game. Our theorem not only shows how to achieve CCA security, but also shows how to convert *any* IND-CCA scheme into an FV-RRA-CCA secure scheme. Furthermore, the security bound of our theorem is tighter than that of Section 6.1, and our theorem facilitates the use of much smaller public keys. For comparison, consider the scheme obtained from combining our transform with the hybrid encryption scheme of Kurosawa and Desmedt (KD) [53]. The KD scheme makes use of public keys consisting of the five components; four group elements (which includes the group generators used in the scheme) and a hash function key. When using our transform with the above Goldreich-Levin extractor, the public key is modified to include λ components from $H \subset \mathbb{Z}_q$. Hence, the public key of the transformed scheme will consist of $\lambda + 5$ components. In contrast, the modified BHHO scheme presented in earlier requires public keys consisting of $2 \cdot k(\lambda)$ group elements (where k is a polynomial). Furthermore, the loss of security in the security reduction of the modified BHHO scheme includes the component $\sqrt[3]{512\delta k p^4}$, which originates from the reduction to the δ-hard-to-invert functions. In comparison, the corresponding loss of security obtained from applying our transform is $\sqrt[3]{512\delta\lambda p^3}$, which

leads to a much weaker requirement on the δ-hard-to-invert functions.

It would of course be desirable to find extractors that meet our strongest definition, or alternative extractors that have, for example, shorter seeds. However, this seems difficult at present. A standard technique to obtain an (ϵ, δ)-auxiliary-input reconstructive extractor is to use complexity-leveraging with a standard reconstructive extractor. Unfortunately, this technique does not appear to work in the FV-RRA-ATK setting. More specifically, if we wish to use complexity-leveraging, we require the range of the auxiliary function to be smaller than the domain. However, for our FV-RRA-ATK game to make sense, we require that for each ϕ we have $\mathcal{D}(\phi) = \mathcal{R}(\phi) = \texttt{Rnd}$. Hence, complexity-leveraging seems to be incompatible with the FV-RRA-ATK model.

6.4 Connections with Correlated-Input Secure Hash Functions

We will now briefly explore the connections between $(\epsilon, \delta, \mathbf{\Phi}, \mathbf{\Phi}')$-auxiliary-input reconstructive extractors and correlated-input secure (CIS) hash functions. In particular, we will show that any reconstructive extractor can be used to construct a secure CIS hash function. Correlated-input secure hash functions were first studied by Goyal et al. in [32]. They introduced several definitions of security, but the one we shall be concerned with is the pseudo-randomness notion, which we briefly studied in Section 5.5.

As noted in [32], CIS hash functions have applications to password-based login and efficient searches on encrypted data. Furthermore, they share interesting connections with Related-Key Attack PRFs (RKA-PRFs) and RKA-secure symmetric encryption schemes. In particular, [32] highlighted that an RKA-PRF can be obtained from a CIS hash function, simply by reversing

proc. Initialise(λ):	**proc. Challenge**(j):	**proc. Query**(i):
$b \leftarrow_\$ \{0,1\}$;	if $b = 0$,	return $h_c(\phi'_i(r))$
$h_c \leftarrow_\$ \mathcal{H}$	$z \leftarrow_\$ \mathcal{R}(h_c)$	
$r \leftarrow_\$ \mathcal{D}(h_c)$	return z	**proc. Finalise**(b'):
return h_c.	else,	if $b = b'$, return 1
	return $h_c(\phi_j(r))$.	else, return 0.

Figure 6.6: The $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-CIS hash game, where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ and $\boldsymbol{\phi}' = (\phi'_1, \ldots, \phi'_{q'})$.

the roles of the input message and the key. Here, we show that they share further connections with auxiliary-input reconstructive extractors. However, to explore this connection we must consider a variant of the CIS hash security game that was presented in [32]. The security game is shown in Figure 6.6 while our definition of security is given below.

Definition 6.4.1. The advantage of an adversary $\mathcal{A}$ against a family of hash functions $\mathcal{H}$ in the $(\boldsymbol{\phi}, \boldsymbol{\phi}')$-CIS game (Figure 6.6) is defined to be

$$\mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-cis}}_{\mathcal{H},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[(\boldsymbol{\phi}, \boldsymbol{\phi}')\text{-CIS}^{\mathcal{A}}_{\mathcal{H}}(\lambda) \Rightarrow 1] - 1.$$

Definition 6.4.2. A family of hash functions $\mathcal{H}$ is said to be $(\boldsymbol{\Phi}, \boldsymbol{\Phi}')$-pseudorandom correlated-input secure if, for all $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ and for all $\boldsymbol{\phi}' \in \boldsymbol{\Phi}'$,and all polynomial time adversaries $\mathcal{A}$, we have

$$\mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-cis}}_{\mathcal{H},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda).$$

Notice that in our new definition, instead of letting the adversary adaptively choose the functions, as in [32], the security game itself is parametrised with function vectors $\boldsymbol{\phi}$ and $\boldsymbol{\phi}'$, and security is required to hold for all choices of $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ and $\boldsymbol{\phi}' \in \boldsymbol{\Phi}'$. It is worth stressing that there is a subtle difference between the two approaches to defining security for CIS hash functions, which makes the two security notions incomparable.

With these definitions and notions in place we can define our hash function

as follows:

$$h_c(r) := \texttt{Ext}(r, c).$$

The following theorem establishes the correlated-input security of the hash function, based on the security of the underlying auxiliary-input reconstructive extractor.

Theorem 6.4.1. Let $\texttt{Ext}$ be an $(\epsilon, \delta, \boldsymbol{\Phi}, \boldsymbol{\Phi}')$-auxiliary-input reconstructive extractor. For the hash function defined as $h_c(r) := \texttt{Ext}(c, r)$ and for any $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ and $\boldsymbol{\phi}' \in \boldsymbol{\Phi}'$, either $\boldsymbol{\Phi}$ is not δ-hard-to-compute with respect to $\boldsymbol{\Phi}'$ or for all polynomial time adversaries $\mathcal{A}$, we have

$$\mathbf{Adv}^{(\boldsymbol{\phi},\boldsymbol{\phi}')\text{-cis}}_{\mathcal{H},\mathcal{A}}(\lambda) < \epsilon.$$

Proof. We will firstly give an overview of the proof. If $\boldsymbol{\Phi}$ is not δ-hard-to-invert with respect to $\boldsymbol{\Phi}'$, then there is nothing to prove. Otherwise, if an adversary $\mathcal{A}$ has advantage greater than or equal to ϵ in the CIS hash game, we would be able to build an extractor adversary $\mathcal{D}$ that distinguishes the outputs of the extractor from randomwith probability ϵ. This in turn would allow us to build a function $\texttt{Rec}$ that computes $\phi_i(r)$ for some i with probability greater than δ (cf. Definition 6.3.4), which is not possible by assumption. Hence, we have a contradiction, so the advantage of the adversary $\mathcal{A}$ must be less than ϵ. This completes the overview of the proof, and we will now explain the reduction in detail.

The extractor adversary $\mathcal{D}$ is given a seed s, a vector of functions $\boldsymbol{\phi}'(r)$ for some r, and $\{z_i\}_{i\in\{1,\ldots,q\}}$, which is either $\{\texttt{Ext}(\phi_i(r), s)\}_{i\in\{1,\ldots,q\}}$ or $\{\sigma_i\}_{i\in\{1,\ldots,q\}}$ for randomly chosen sigmas. Adversary $\mathcal{D}$ forwards the seed s to $\mathcal{A}$. Adversary $\mathcal{D}$ then answers the oracle queries as follows.

Query i

$\mathcal{D}$ responds with $\texttt{Ext}(\phi'_i(r), c)$.

Challenge j

$\mathcal{D}$ responds with z_j.

When $\mathcal{A}$ outputs a bit b, $\mathcal{D}$ outputs the same bit b. It is clear that $\mathcal{D}$ has the same advantage as $\mathcal{A}$. Since $\mathcal{D}$ must have advantage less than ϵ, we conclude that $\mathcal{A}$ also has advantage less than ϵ. □

A concrete instantiation of such a CIS hash is possible via Equation (6.3.1). If we define

$$h_c(r) := \langle r, c \rangle, \tag{6.4.1}$$

where $c \in \mathbb{Z}_p^\lambda$ and $r \in H^\lambda$ for $H \subset \mathbb{Z}_p$, then the following corollary is obvious.

Corollary 6.4.1. Consider the hash function defined in Equation 6.4.1. Let $\mathbf{\Phi}$ be the set of δ-hard-to-invert functions. Then, for all $\boldsymbol{\phi} \in \mathbf{\Phi}$, and all polynomial time adversaries $\mathcal{A}$, we have

$$\mathbf{Adv}_{\mathcal{H},\mathcal{A}}^{(\mathrm{id},\boldsymbol{\phi})\text{-cis}}(\lambda) < \sqrt[3]{512\delta\lambda p^3}.$$

6.5 Conclusions

In this chapter we have proposed an alternative security notion for Related Randomness Attacks. Furthermore, we have provided constructions that are secure in this new model when an adversary may only use hard-to-invert functions in its oracle queries. The results of this chapter develop interesting connections with auxiliary-input reconstructive extractors. Moreover, we show that this specific type of extractor may be used to construct CIS hash functions (albeit in a security game that differs slightly from the original), thereby introducing connections to the previous chapter in which we used CIS hash functions to obtain related randomness security in the honest-key setting. The major open problem from this section is to find a concrete instantiation of the

most general type of auxiliary-input reconstructive extractor that we have defined in this chapter.

Chapter 7

Related Randomness Attacks for Key Encapsulation Mechanisms

7.1 Introduction

In this chapter we will continue to study Related Randomness Attacks, but we will turn our attention towards Key Encapsulation Mechanisms (KEMs). A drawback to public-key encryption is that the message space is often restricted in some manner. For example, in the case of ElGamal (Section 5.1.2, Figure 5.1) the message space is a group $\mathbb{G}$ for which the Decisional Diffie-Hellman problem is hard. Furthermore, the group operations required for public-key encryption are much more costly than the corresponding operations required for symmetric schemes, resulting in PKE schemes being less efficient than their SKE counterparts. Therefore, in practice it is common to use a PKE scheme to transport an SKE private key, and then continue communications via an SKE scheme with this transported key. KEM schemes are a formalisation of the technique for transporting keys, whilst the KEM/DEM setting formalises the whole process of transporting the key and then encrypting messages using the symmetric scheme. As a consequence, it is logical to

Alg. $\texttt{KEM-DEM.K}(1^\lambda)$:	Alg. $\texttt{KEM-DEM.E}(ek, m)$:	Alg. $\texttt{KEM-DEM.D}(dk, c_{kem}, c_{dem})$:
$(ek, dk) \leftarrow_\$ \texttt{KEM.K}(1^\lambda)$	$r \leftarrow_\$ \texttt{Rnd}$	$k \leftarrow \texttt{KEM.D}(dk, c_{kem})$
return (ek, dk).	$(c_{kem}, k) \leftarrow \texttt{KEM.E}(ek; r)$	$m \leftarrow \texttt{DEM.D}(k, c_{dem})$
	$c_{dem} \leftarrow \texttt{DEM.E}(k, m)$	return m.
	return (c_{kem}, c_{dem}).	

Figure 7.1: Standard construction of a PKE scheme KEM-PKE from a KEM scheme KEM and a DEM scheme DEM.

study Related Randomness Attacks specifically in this setting.

To summarise the KEM/DEM setting, the KEM component encrypts an encoding of a symmetric key, and then the DEM will be used to encrypt the message with the symmetric key. This is the reason why the KEM/DEM paradigm is slightly more efficient than exclusively using public key encryption schemes (and there are fewer restrictions on the message space). Furthermore, since the symmetric key is independent for every message sent, the DEM scheme need only be one-time secure, which is a simpler notion to achieve than the more general notions of symmetric security. The KEM/DEM approach was formalised by Cramer and Shoup [21]. In Figure 7.1 we have reproduced their original construction of a PKE scheme from a KEM and a DEM. Later in this chapter, we will build upon this scheme to provide a KEM/DEM framework for RRA-ATK PKE security.

Since KEM/DEM constructions yield public-key encryption schemes, it is clear that the results of the previous chapters may be applied to the KEM/DEM construction to trivially achieve RRA-ATK security. Why, then, do we bother to study related randomness for the KEM/DEM paradigm? It is possible that a study of RRA specifically for KEMs will lead to tighter security reductions, or security against stronger classes of functions. In this chapter we will encounter reasons to believe that achieving related randomness security for KEMs may be simpler than achieving the notion for PKE schemes.

7.1.1 Our contributions

We will begin by defining Related Randomness Attacks in the KEM setting and the relevant notions of security. Once we have established the necessary definitions, we will present a plausible argument to suggest that achieving RRA security in the KEM setting may be easier than for the PKE setting. Subsequently, we will see how an RRA-secure KEM can be combined with several other standard primitives to obtain an RRA-secure PKE scheme. Interestingly, this develops connections with the previous chapter since we show that Theorem 6.1.2 is essentially a special case of Theorem 7.3.1 proved in this chapter.

7.2 Related Randomness Security for KEMs

We begin by defining the RRA-CCA security game for KEMs, which can be seen in Figure 7.2 (standard KEM definitions are in the introductory chapter, Section 2.2.3). The notation $\mathcal{K}$ denotes the key space to which the KEM scheme `KEM` maps. That is, `KEM` maps to $\mathcal{C} \times \mathcal{K}$. Recall that in the PKE setting we were required to place restrictions upon adversaries in order to prevent trivial wins. We must similarly restrict KEM adversaries using an equality-pattern notion reminiscent of Definition 5.2.1, though the KEM restrictions will be much simpler to state than those for public-key encryption schemes. Informally, a KEM adversary in the RRA-ATK game may not query a pair (i, ϕ) to its real-or-random (**RoR**) oracle if the triple (ek^*, i, ϕ) was queried to the **Encap** oracle, and vice versa, where ek^* is the target encryption key. If an adversary were allowed to make these oracle calls, then the adversary would trivially win the game by simply comparing the outputs of the encapsulation and the real-or-random query. If the two keys are identical, then the the adversary knows with high probability that he is in the real world. The

proc. Initialise(λ):	**proc. RoR**(j, ϕ):	**proc. Decap**(c):
$b \leftarrow_\$ \{0,1\}$;	if CoinTab$[j] = \perp$	if $c \in \mathcal{S}$
CoinTab $\leftarrow \emptyset$;	CoinTab$[j] \leftarrow_\$$ Rnd	return $\perp$
$(ek^*, dk^*) \leftarrow_\$ \mathtt{KEM.K}(1^\lambda)$;	$r_j \leftarrow$ CoinTab$[j]$	else
$\mathcal{S} \leftarrow \emptyset$; return ek^*.	$(c, k_0) \leftarrow \mathtt{KEM.E}(ek^*; \phi(r_j))$	return $\mathtt{KEM.D}(dk^*, c)$.
	$k_1 \leftarrow_\$ \mathcal{K}$	
proc. Encap(ek, j, ϕ):	$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$	**proc. Finalise**(b'):
if CoinTab$[j] = \perp$	return (c, k_b).	if $b = b'$
CoinTab$[j] \leftarrow_\$$ Rnd		return 1
$r_j \leftarrow$ CoinTab$[j]$		else
$(c, k) \leftarrow \mathtt{KEM.E}(ek; \phi(r_j))$		return 0.
return (c, k).		

Figure 7.2: Game RRA-CCA for KEMs. The access to **proc. Decap** is removed for the RRA-CPA version of the game.

formalisation of the equality-pattern definition for KEMs is below.

Definition 7.2.1. Let $\mathcal{A}$ be a non-trivial KEM adversary that queries r different indices to its oracles and makes q_i **RoR** queries with index i. Let Φ_i be the set of functions ϕ such that $\mathcal{A}$ makes **Encap** query (ek^*, i, ϕ). Let $\phi_1^i, \phi_2^i, \ldots, \phi_{q_i}^i$ be $\mathcal{A}$'s **RoR** queries for index i. If for all $i \in \{1, \ldots, r\}$ and all $j \in \{1, \ldots, q_i\}$ we have

$$\phi_j^i \notin \Phi_i,$$

then we say $\mathcal{A}$ is an *equality-pattern respecting KEM adversary.*

Definition 7.2.2. We define the advantage of an equality-pattern respecting RRA-ATK adversary $\mathcal{A}$ against a KEM scheme KEM to be

$$\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{KEM},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\text{RRA-ATK}^{\mathcal{A}}_{\mathtt{KEM}}(\lambda) \Rightarrow 1] - 1.$$

A KEM scheme KEM is said to be Φ-RRA-ATK secure if the advantage of any Φ-restricted, equality-pattern respecting, RRA-ATK adversary against KEM that runs in polynomial time is negligible in the security parameter λ.

proc. Initialise(λ, ℓ):
$b \leftarrow_{\$} \{0,1\}$;
Keys $\leftarrow \emptyset$;
`target` $\leftarrow$ `false`;
CoinTab $\leftarrow \emptyset$;
$\mathcal{S} \leftarrow \emptyset$; $(ek^*, dk^*) \leftarrow \emptyset$
for $i = 1$ to ℓ
  $(ek_i, dk_i) \leftarrow_{\$}$ `KEM.K`(1^λ)
  Keys $\leftarrow$ Keys $\cup\, ek_i$
return Keys.

proc. Target(j):
if `target` $=$ `true`
  return $\perp$
else
  $(ek^*, dk^*) \leftarrow (ek_j, dk_j)$
`target` $\leftarrow$ `true`
return $\{dk_i\}_{i \neq j}$.

proc. Encap(ek, i, ϕ):
if `target` $=$ `false`
  return $\perp$
if $ek \notin$ Keys
  return $\perp$
if CoinTab$[i] = \perp$
  CoinTab$[i] \leftarrow_{\$}$ `Rnd`
$r_i \leftarrow$ CoinTab$[i]$
$(c, k) \leftarrow$ `KEM.E`$(ek; \phi(r_i))$
return (c, k).

proc. Decap(c):
if `target` $=$ `false`
  return $\perp$
if $c \in \mathcal{S}$
  return $\perp$
else
  return `KEM.D`(dk^*, c).

proc. RoR(i, ϕ):
if `target` $=$ `false`,
  return $\perp$
if CoinTab$[i] = \perp$
  CoinTab$[i] \leftarrow_{\$}$ `Rnd`
$r_i \leftarrow$ CoinTab$[i]$
$(c, k_0) \leftarrow$ `KEM.E`$(ek^*; \phi(r_i))$
$k_1 \leftarrow_{\$} \mathcal{K}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
return (c, k_b).

proc. Finalise(b'):
if $b = b'$
  return 1
else
  return 0.

Figure 7.3: Game ℓ-HK-RRA-ATK. (Note that if ATK = CPA, then the adversary's access to **proc. Decap** is removed.)

7.2.1 Alternative security notions

Similar to the PKE setting, we can define variants of the RRA game for the honest-key and the function-vector settings. The game for honest keys is in Figure 7.3, and security is defined in the natural way. For the RRA and HK-RRA games, we may define selective versions of these games. The games and security definitions will not be given here, but they are obvious extensions of the adaptive games.

For function-vector security, the game is in Figure 7.4, and the definition of security is below.

Definition 7.2.3. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$ be a vector of $q := q(\lambda)$ functions. We define the advantage of an equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-ATK adversary $\mathcal{A}$ against a KEM scheme `KEM` to be

$$\mathbf{Adv}^{\boldsymbol{\phi}\text{-fv-rra-atk}}_{\mathtt{KEM},\mathcal{A}}(\lambda) := 2 \cdot \mathbb{P}[\boldsymbol{\phi}\text{-FV-RRA-ATK}^{\mathcal{A}}_{\mathtt{KEM}}(\lambda) \Rightarrow 1] - 1.$$

proc. Initialise(λ):
$b \leftarrow_\$ \{0,1\}$;
$(ek^*, dk^*) \leftarrow_\$ \texttt{KEM.K}(1^\lambda)$;
CoinTab $\leftarrow \emptyset$; $\mathcal{S} \leftarrow \emptyset$;
return ek^*.

proc. Decap(c):
if $c \in \mathcal{S}$
    return $\perp$
else
    return $\texttt{KEM.D}(dk^*, c)$.

proc. RoR(i):
if CoinTab$[i] = \perp$
    CoinTab$[i] \leftarrow_\$ \texttt{Rnd}$
$r_i \leftarrow$ CoinTab$[i]$
$(c, k_0) \leftarrow \texttt{PKE.E}(ek^*; r_i)$
$k_1 \leftarrow_\$ \mathcal{K}$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
return (c, k_b).

proc. Encap(ek, i, j):
if CoinTab$[i] = \perp$
    CoinTab$[i] \leftarrow_\$ \texttt{Rnd}$
$r_i \leftarrow$ CoinTab$[i]$
$(c, k) \leftarrow \texttt{KEM.E}(ek; \phi_j(r_i))$
return c.

proc. Finalise(b'):
if $b = b'$
    return 1
else
    return 0.

Figure 7.4: Game $\boldsymbol{\phi}$-FV-RRA-ATK, where $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_q)$. (As usual, if ATK = CPA, then the adversary's access to **proc. Decap** is removed.)

If $\boldsymbol{\Phi}$ is a set of vectors of functions, then a KEM scheme KEM is said to be $\boldsymbol{\Phi}$-FV-RRA-ATK secure if, for all $\boldsymbol{\phi} \in \boldsymbol{\Phi}$, the advantage of any equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-ATK adversary against KEM that runs in polynomial time is negligible in the security parameter λ.

7.2.2 Simplifying lemmas

In this section we will prove some lemmas regarding the security of RRA-KEMs. Specifically, we will show that it suffices to prove that the KEM is secure when an adversary uses only one randomness index and one **RoR** query.

To prove that we only need to consider adversaries that use one randomness index, we proceed exactly as in the PKE setting. The KEM adversary with one randomness index simply selects his own randomness for the other indices and uses these to simulate queries for different randomness indices. The theorem statement is below, and the proof is omitted since it is nearly identical to the proof of Lemma 5.2.1.

Lemma 7.2.1. Consider an equality-pattern respecting, RRA-ATK adversary

$\mathcal{A}$ that queries q_r distinct randomness indices and makes at most q_{RoR} **RoR** queries. Then there exists an equality-pattern respecting, RRA-ATK adversary $\mathcal{B}$ that queries at most one randomness index and makes at most q_{RoR} **RoR** queries such that

$$\mathbf{Adv}^{\text{rra-atk}}_{\texttt{KEM},\mathcal{A}}(\lambda) \leq q_r \cdot \mathbf{Adv}^{\text{rra-atk}}_{\texttt{KEM},\mathcal{B}}(\lambda),$$

where $\mathcal{B}$ runs in approximately the same time as $\mathcal{A}$. In the CCA setting, $\mathcal{B}$ makes the same number of decapsulation queries as $\mathcal{A}$.

In the PKE setting we were unable to show that there is a security reduction from an adversary with multiple **LR** queries to an adversary with just one **LR** query. Instead, the reduction had to be made in the proof of the specific scheme we were considering. In the KEM setting, a result of this kind is in fact achievable. Using a standard hybrid argument, we can prove the following lemma, which shows it is sufficient to consider an adversary that makes only one **RoR** query and uses only one randomness index.

Lemma 7.2.2. Consider an equality-pattern respecting, RRA-ATK adversary $\mathcal{A}$ that queries one distinct randomness index and makes at most q_{RoR} **RoR** queries. Then there exists an equality-pattern respecting, RRA-ATK adversary $\mathcal{B}$ that queries at most one randomness index and makes at most one **RoR** query such that

$$\mathbf{Adv}^{\text{rra-atk}}_{\texttt{KEM},\mathcal{A}}(\lambda) \leq q_{RoR} \cdot \mathbf{Adv}^{\text{rra-atk}}_{\texttt{KEM},\mathcal{B}}(\lambda),$$

where $\mathcal{B}$ runs in approximately the same time as $\mathcal{A}$. In the CCA setting, $\mathcal{B}$ makes the same number of decapsulation queries as $\mathcal{A}$.

Proof. For $0 \leq j \leq q_{RoR}$, define Game G_j as in Figure 7.5. Note that G_0 is identical to the RRA-CCA game in Figure 7.2 with $b = 0$, whilst $G_{q_{RoR}}$ is identical to the RRA-CCA game in Figure 7.2 with $b = 1$. Since we are

now considering an adversary that uses only one randomness index, this index will be omitted from an adversary's queries. The advantage of $\mathcal{A}$ may be formulated as

$$\begin{aligned}
\mathbf{Adv}^{\text{rra-atk}}_{\mathtt{KEM},\mathcal{A}}(\lambda) &= \mathbb{P}[\mathcal{A}^{G_{q_{RoR}}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_0} \Rightarrow 1] \\
&= \sum_{j=0}^{q_{RoR}-1} \mathbb{P}[\mathcal{A}^{G_{j+1}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_j} \Rightarrow 1] \\
&\leq q_{RoR} \cdot (\mathbb{P}[\mathcal{A}^{G_{j^*+1}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_{j^*}} \Rightarrow 1]),
\end{aligned}$$

for some index $j^* \in \{1, \ldots, q_{RoR}\}$. The only difference between games j^* and $j^* + 1$ is how the game responds to the $(j^* + 1)$th **RoR** query. If $\mathcal{A}$ can distinguish between games j^* and j^*+1, then we may construct an adversary $\mathcal{B}$ that wins the RRA-CCA game and only uses one **RoR** query. The adversary $\mathcal{B}$ runs the following set up procedure. Adversary $\mathcal{B}$ sets ctr $= 1$, $\mathcal{S} = \emptyset$, and forwards his encapsulation key ek^* to $\mathcal{A}$. Adversary $\mathcal{B}$ then simulates the queries as follows:

Encap query (ek, ϕ)

$\mathcal{B}$ forwards (ek, ϕ) to his own **Encap** oracle

the answer is returned to $\mathcal{A}$.

RoR query ϕ

if ctr $\leq j^*$

$\mathcal{B}$ submits (ek^*, ϕ) to his **Encap** oracle

$\mathcal{B}$ receives (c, k_0) and chooses $k_1 \leftarrow_\$ \mathcal{K}$

$\mathcal{B}$ sets ctr $\leftarrow$ ctr $+ 1$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$, and returns (c, k_1)

else if ctr $= j^* + 1$

$\mathcal{B}$ submits ϕ to his **RoR** oracle

$\mathcal{B}$ sets ctr $\leftarrow$ ctr $+ 1$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$, returns the oracle output to $\mathcal{A}$

else

$\mathcal{B}$ submits ϕ to his **Encap** oracle

$\mathcal{B}$ sets ctr $\leftarrow$ ctr $+ 1$, $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$, returns the oracle output to $\mathcal{A}$.

proc. Initialise(λ):
$(ek^*, dk^*) \leftarrow_\$ \texttt{KEM.K}(1^\lambda)$;
$r \leftarrow_\$ \texttt{Rnd}$;
$\mathcal{S} \leftarrow \emptyset$; ctr $= 1$;
return ek^*.

proc. Encap(ek, ϕ):
$(c, k) \leftarrow \texttt{KEM.E}(ek; \phi(r))$
return (c, k).

proc. Decap(c):
if $c \in \mathcal{S}$
    return $\perp$
else
    return $\texttt{KEM.D}(c)$.

proc˙RoR(ϕ):
$(c, k_0) \leftarrow \texttt{KEM.E}(ek^*; \phi(r))$
if ctr $> j$
    return (c, k_0)
else
    $k_1 \leftarrow_\$ \mathcal{K}$
    return (c, k_1)
$\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$
return c.

proc. Finalise(b'):
if $b = b'$
    return 1
else
    return 0.

Figure 7.5: The game G_j for Lemma 7.2.2.

Decap query c

if $c \in \mathcal{S}$

$\mathcal{B}$ returns $\perp$

else

$\mathcal{B}$ forwards c to his **Decap** oracle

$\mathcal{B}$ returns the oracle output to $\mathcal{A}$.

When $\mathcal{A}$ halts and outputs b', $\mathcal{B}$ halts and outputs b'. When $\mathcal{B}$ is given real keys, he simulates game j^* perfectly. If the key is random, $\mathcal{B}$ simulates game $j^* + 1$ perfectly. The lemma now follows easily. Note, however, that when ATK=CPA, the adversary $\mathcal{B}$ will not simulate **Decap** queries since no **Decap** oracle exists in this version of the game. □

These two lemmas have established that we only need to consider adversaries using one randomness index and one **RoR** query for RRA security of KEMs. This gives us some indication as to why RRA-secure KEMs may be easier to obtain than RRA-secure PKE schemes. For PKE we need to show

that a scheme is secure when the adversary has multiple **LR** queries, whereas for KEMs we only need to prove security for one **RoR** query. If RRA security is indeed easier to achieve in this setting, then we would like to find a way of achieving RRA security for PKE via the RRA security of a KEM. The next section will show how to realise this goal.

7.3 Related Randomness for the KEM/DEM Paradigm

The CPA/CCA security proof of the scheme in Figure 7.1 (from [21]) only requires the DEM to be secure against adversaries that have only one challenge query, which is equivalent to one **LR** query in our setting (so called 'one-time security'). The reasoning behind this is straightforward. Since the KEM is randomised, the probability of the KEM outputting a pair (c, k) more than once is negligible (because otherwise we would be able to construct an algorithm with a non-negligible advantage against the scheme). Therefore, even though the PKE adversary will have multiple **LR** queries, each encryption by the DEM will (with high probability) be computed using different DEM keys, so the DEM need only be secure against one **LR** query. We will shortly see that we require stronger assumptions in the RRA setting.

We will attempt to extend the construction of Figure 7.1 to the related randomness setting. Specifically, we intend to combine an RRA-secure KEM with other components to achieve RRA PKE security. Extending the scheme of Figure 7.1 to the related randomness setting is not entirely straightforward. First, one-time security will no longer suffice, since if an adversary reuses the same KEM randomness in different oracle queries, then the DEM key output by the KEM will be identical. Hence, the DEM key will be used more than once and the scheme is no longer provably secure if we are using a one-time secure

Alg. $\texttt{KEM-PKE.K}(1^\lambda)$:	Alg. $\texttt{KEM-PKE.E}(ek, m)$:	Alg. $\texttt{KEM-PKE.D}(dk, c)$:
$(ek, dk) \leftarrow_\$ \texttt{KEM.K}(1^\lambda)$	$r_{kem} \leftarrow_\$ \texttt{Rnd}$	$(c_{kem}, c_{dem}) \leftarrow c$
return (ek, dk).	$(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek; r_{kem})$	$k_{kem} \leftarrow \texttt{KEM.D}(dk, c_{kem})$
	$(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$	$(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$
	$c_{dem} \leftarrow \texttt{DEM.E}(k_{dem}, m; F_{r_{dem}}(m))$	$m \leftarrow \texttt{DEM.D}(k_{dem}, c_{dem})$
	return $c = (c_{kem}, c_{dem})$.	return m.

Figure 7.6: Construction of a PKE scheme KEM-PKE from a KEM scheme KEM, a DEM scheme DEM, a KDF f, and a PRF, F.

DEM. A consequence of this is that we must necessarily use a randomised DEM. This then presents its own difficulties. In the related randomness setting the adversary can control the randomness, but we now have two randomness sources, both of which can be controlled by the adversary. How, then, do we overcome this issue? One approach is to borrow the techniques of the modified BHHO scheme in Figure 6.2. The key that is output by the KEM will be fed into a KDF, then the KDF will output two values. The first will be used as a DEM key and the second will be used as a key for a PRF. The input to the PRF will be the message that is to be encrypted, and the output of the PRF will be used as the randomness for the DEM. Our proposed construction formalising this idea can be found in Figure 7.6. Notice that we now have a randomised DEM, but we need not give the adversary control of this randomness since the algorithm never chooses the randomness with which to encrypt. Instead, the randomness used for DEM encryption is a deterministic function of the KEM randomness and the message.

The following theorem establishes the related randomness security of our scheme in Figure 7.6.

Theorem 7.3.1. Suppose $\mathcal{A}$ is a Φ-restricted, equality-pattern respecting adversary in the RRA-ATK game in Figure 5.3 against the PKE scheme KEM-PKE in Figure 7.6. Suppose $\mathcal{A}$ makes at most q_{LR} **LR** queries, at most s **Enc** queries,

queries q_r randomness indices, and requests oracle outputs for at most q_t distinct functions when using the target public key. Then there exists an equality-pattern respecting, Φ-restricted KEM adversary $\mathcal{B}$, a KDF adversary $\mathcal{C}$, a PRF adversary $\mathcal{D}$, and an IND-ATK adversary $\mathcal{E}$ such that

$$\begin{aligned}\mathbf{Adv}^{\text{rra-atk}}_{\texttt{KEM-PKE},\mathcal{A}}(\lambda) \;\leq\; & 2q_tq_r \cdot \mathbf{Adv}^{\text{rra-atk}}_{\texttt{KEM},\mathcal{B}}(\lambda) + 2q_tq_r \cdot \mathbf{Adv}^{\text{kdf}}_{f,\mathcal{C}}(\lambda) \\ & + 2q_tq_r \cdot \mathbf{Adv}^{\text{prf}}_{F,\mathcal{D}}(\lambda) + q_tq_r \cdot \mathbf{Adv}^{\text{ind-atk}}_{\texttt{DEM},\mathcal{E}}(\lambda).\end{aligned}$$

Adversary $\mathcal{B}$ makes one **RoR** query, uses only one randomness index, and makes at most s encapsulation queries. Adversary $\mathcal{C}$ makes just one oracle query, adversary $\mathcal{D}$ makes at most $q_{LR}+s$ oracle queries, and $\mathcal{E}$ makes at most $q_{LR}+s$ **Enc**/**LR** queries.

Proof. We first invoke Lemma 5.2.1, so that we may now consider an adversary that uses only one randomness index. We prove the theorem via a sequence of game hops. The changes in each game refer only to queries using the target public key. For oracle queries that do not use the target public key, the queries are answered as in the real game.

$\mathbf{G_0}$: This is the real game as defined in Figure 5.3.

$\mathbf{G_1}$: This is the same as G_0, except for encryptions under the target public key. Whenever the game computes an encryption (c_{kem}, c_{dem}), the game checks whether $(c_{kem}, \cdot)$ is in a look-up table. If not, then the game stores (c_{kem}, k_{kem}) in the look-up table, where k_{kem} was the key output by the KEM during encryption. Whenever the game receives a decryption query (c_{kem}, c_{dem}), the game checks whether $(c_{kem}, \cdot)$ is the look-up table. If so, the game decrypts with the corresponding k_{kem}. Otherwise, the game decrypts normally. Notice that this game is syntactically equivalent to G_0. Hence $\mathbb{P}[G_0^{\mathcal{A}} \Rightarrow 1] = \mathbb{P}[G_1^{\mathcal{A}} \Rightarrow 1]$.

$\mathbf{G_2}$: The same as G_1, except for queries under the target encryption key. Rather than return real KEM keys the encryption algorithm creates a uniformly random key for each value ϕ queried. If an adversary can distinguish G_2 from G_1, then we may construct an adversary that wins the RRA-ATK game for KEMs.

$\mathbf{G_3}$: The same as G_2, except for decryptions. Rather than use the outputs of the KDF to decrypt, the game stores the KDF outputs with the KEM keys (during encryption) in a look-up table, and uses these to decrypt. This game is syntactically equivalent to G_2. Hence, $\mathbb{P}[G_3^{\mathcal{A}} \Rightarrow 1] = \mathbb{P}[G_2^{\mathcal{A}} \Rightarrow 1]$.

$\mathbf{G_4}$: The same as G_3, except outputs of the KDF are replaced by uniformly random values. If an adversary can distinguish games G_4 and G_3, we may construct an adversary that wins the KDF game.

$\mathbf{G_5}$: The same as G_4, except outputs of the PRF are replaced with uniformly random values. If an adversary can distinguish games G_5 and G_4, we may construct an adversary that wins the PRF game.

The differences between these games will now be studied in more detail.

$\mathbf{G_1}$ –$\mathbf{G_2}$: We will prove the following:

Lemma 7.3.1. For any adversary $\mathcal{A}$, there exists a KEM adversary $\mathcal{B}$ that makes one **RoR** query such that

$$\mathbb{P}[G_0^{\mathcal{A}}] - \mathbb{P}[G_1^{\mathcal{A}}] \leq q_t \cdot \mathbf{Adv}^{\text{rra-atk}}_{\mathtt{KEM},\mathcal{B}}(\lambda).$$

Proof. We first consider an adversary $\mathcal{B}$ that is allowed multiple **LR** queries. The adversary $\mathcal{B}$ sets up the simulation by choosing $b \leftarrow_{\$} \{0,1\}$, setting $\mathcal{Q} \leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$, and forwarding his own encapsulation key ek^* to $\mathcal{A}$. Adversary $\mathcal{B}$ then simulates the oracle queries as follows:

Enc query (ek, m, ϕ)

if $ek = ek^*$

$\mathcal{B}$ submits ϕ to **RoR** oracle and receives (c_{kem}, k_{kem})

$\mathcal{B}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{B}$ computes $c_{dem} \leftarrow \mathtt{DEM.E}(k_{dem}, m; F_{r_{dem}}(m))$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{B}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, k_{kem})$

$\mathcal{B}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$

else

$\mathcal{B}$ submits (ek, ϕ) to his **Encap** oracle and obtains output (c_{kem}, k_{kem})

$\mathcal{B}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{B}$ computes $c_{dem} \leftarrow \mathtt{DEM.E}(k_{dem}, m; F_{r_{dem}}(m))$

$\mathcal{B}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

LR query (m_0, m_1, ϕ)

$\mathcal{B}$ submits ϕ to its **RoR** oracle and receives (c_{kem}, k_{kem})

$\mathcal{B}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{B}$ computes $c_{dem} \leftarrow \mathtt{DEM.E}(k_{dem}, m_b; F_{r_{dem}}(m_b))$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{B}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, k_{kem})$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(c_{kem}, c_{dem})\}$

$\mathcal{B}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

Dec query (c_{kem}, c_{dem})

if $(c_{kem}, c_{dem}) \in \mathcal{S}$

return $\perp$

if $(c_{kem}, \cdot) \in \mathcal{Q}$

$\mathcal{B}$ obtains corresponding k_{kem}

else

$\mathcal{B}$ submits c_{kem} to the decapsulation oracle, obtaining k_{kem}

$\mathcal{B}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{B}$ returns $m \leftarrow \texttt{DEM.D}(k_{dem}, c_{dem})$.

When $\mathcal{A}$ halts and outputs bit b', adversary $\mathcal{B}$ halts and outputs 1 if $b = b'$. If $\mathcal{B}$'s oracle provides real encryptions then G_1 is simulated perfectly. Otherwise $\mathcal{B}$ perfectly simulates G_2. Note that adversary $\mathcal{B}$ uses q_t **RoR** challenge queries, but the security can be reduced to a KEM adversary that has only one **RoR** query courtesy of Lemma 7.2.2. Hence, we obtain the desired result. □

G_3 –G_4: We will prove the following:

Lemma 7.3.2. For any adversary $\mathcal{A}$ there exists a KDF adversary $\mathcal{C}$ such that

$$\mathbb{P}[G_3^{\mathcal{A}}] - \mathbb{P}[G_4^{\mathcal{A}}] \leq q_t \cdot \mathbf{Adv}^{\text{kdf}}_{f,\mathcal{C}}(\lambda),$$

where adversary $\mathcal{C}$ makes only one query to its oracle.

Proof. The proof will first make use of a KDF adversary $\mathcal{C}$ that is allowed multiple calls to its KDF oracle. The adversary $\mathcal{C}$ chooses a bit $b \leftarrow_{\$} \{0,1\}$, a randomness value $r \leftarrow_{\$} \texttt{Rnd}$, sets $\mathcal{Q} \leftarrow \emptyset$, KDFTab $\leftarrow \emptyset$, and generates a KEM key pair $(ek^*, dk^*) \leftarrow_{\$} \texttt{KEM.K}(1^\lambda)$. Adversary $\mathcal{C}$ then runs $\mathcal{A}$, answering the oracle calls as follows:

Enc query (ek, m, ϕ)

if $ek = ek^*$

$\mathcal{C}$ computes $(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek^*; \phi(r))$

if KDFTab$[\phi] = \perp$

$\mathcal{C}$ invokes its oracle and sets the output $(k_{dem}, r_{dem}) =$ KDFTab$[\phi]$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{C}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, k_{dem})$

$\mathcal{C}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(k_{dem}, m; F_{r_{dem}}(m))$

$\mathcal{C}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$

else

$\mathcal{C}$ computes $(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek; \phi(r))$

$\mathcal{C}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{C}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(k_{dem}, m; F_{r_{dem}}(m))$

$\mathcal{C}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

LR query (m_0, m_1, ϕ)

$\mathcal{C}$ computes $(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek^*; \phi(r))$

if KDFTab$[\phi] = \perp$

$\mathcal{C}$ invokes its oracle and sets the output $(k_{dem}, r_{dem}) =$ KDFTab$[\phi]$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{C}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, k_{dem})$

$\mathcal{C}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(k_{dem}, m_b; F_{r_{dem}}(m))$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(c_{kem}, c_{dem})\}$

$\mathcal{C}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

Dec query (c_{kem}, c_{dem})

if $(c_{kem}, c_{dem}) \in \mathcal{S}$

return $\perp$

if $(c_{kem}, \cdot) \in \mathcal{Q}$

$\mathcal{C}$ obtains corresponding k_{dem}

else

$\mathcal{C}$ computes $k_{kem} \leftarrow kemdec(dk^*, c_{kem})$

$\mathcal{C}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{C}$ returns $m \leftarrow \texttt{DEM.D}(k_{dem}, c_{dem})$.

When $\mathcal{A}$ halts and outputs bit b', adversary $\mathcal{C}$ halts and outputs 1 if $b = b'$. If $\mathcal{C}$'s oracle provides real KDF outputs then G_3 is simulated perfectly. Otherwise $\mathcal{C}$ perfectly simulates G_4. The adversary $\mathcal{C}$ requires at most q_t oracle queries, but a standard security reduction shows the following relation to an adversary $\mathcal{C}'$ that makes one oracle query:

$$\mathbf{Adv}^{\text{kdf}}_{\mathcal{C},f}(\lambda) \leq q_t \cdot \mathbf{Adv}^{\text{kdf}}_{\mathcal{C}',f}(\lambda).$$

The lemma follows easily from this observation. □

$\mathbf{G_4}$ –$\mathbf{G_5}$: We will prove the following:

Lemma 7.3.3. For any adversary $\mathcal{A}$ there exists a PRF adversary $\mathcal{D}$ such that

$$\mathbb{P}[G_4^{\mathcal{A}}] - \mathbb{P}[G_5^{\mathcal{A}}] \leq q_t \cdot \mathbf{Adv}^{\mathrm{prf}}_{F,\mathcal{D}}(\lambda).$$

Proof. Let $G_{4/5,j}$ be the game in which, for distinct function $i \leq j$ (queried with the target public key), random values are output rather than using the PRF. For $i > j$ the PRF is used to generated the DEM randomness for encryption. If an adversary can distinguish games $G_{4/5,j}$ and $G_{4/5,j+1}$, then we may use this adversary to win the PRF game. The PRF distinguisher $\mathcal{D}$ sets up the simulation by choosing a bit $b \leftarrow_{\$} \{0,1\}$, $r \leftarrow_{\$} \mathtt{Rnd}$, sets $\mathcal{P} \leftarrow \emptyset$, $\mathcal{Q} \leftarrow \emptyset$ and ctr $\leftarrow 1$. Adversary $\mathcal{D}$ then generates a KEM key pair $(ek^*, dk^*) \leftarrow \mathtt{KEM.K}$, generates DEM keys $sk_1, \ldots, sk_{q_t}$ and generates PRF keys $K_{j+1}, K_{j+2}, \ldots, K_{q_t}$. Adversary $\mathcal{D}$ answers $\mathcal{A}$'s oracle queries as follows:

Enc query (ek, m, ϕ)

$\mathcal{D}$ computes $(c_{kem}, k_{kem}) \leftarrow \mathtt{KEM.E}(ek, \phi(r))$

if $ek = ek^*$

if $(\phi, \cdot) \in \mathcal{P}$

retrieve corresponding i

else

$\mathcal{P} \leftarrow \mathcal{P} \cup \{(\phi, \mathrm{ctr})\}$ and $i \leftarrow \mathrm{ctr}$

ctr $\leftarrow$ ctr $+ 1$

if $i < j + 1$

$\mathcal{D}$ chooses $r' \leftarrow_{\$} \mathtt{Rnd}_{dem}$

if $i = j + 1$

$\mathcal{D}$ submits m to its PRF oracle and receives response r'

else if $i > j + 1$

$\mathcal{D}$ computes $r' \leftarrow F_{K_i}(m)$

$\mathcal{D}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(sk_i, m; r')$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{D}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, i)$

$\mathcal{D}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$

else

$\mathcal{D}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{D}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(k_{dem}, m; F_{r_{dem}}(m))$

$\mathcal{D}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

LR query (m_0, m_1, ϕ)

$\mathcal{D}$ computes $(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek, \phi(r))$

if $(\phi, \cdot) \in \mathcal{P}$

retrieve corresponding i

else

$\mathcal{P} \leftarrow \mathcal{P} \cup \{(\phi, \text{ctr})\}$ and $i \leftarrow \text{ctr}$

$\text{ctr} \leftarrow \text{ctr} + 1$

if $i < j + 1$

$\mathcal{D}$ chooses $r' \leftarrow_{\$} \texttt{Rnd}_{dem}$

if $i = j + 1$

$\mathcal{D}$ submits m to its PRF oracle and receives response r'

else if $i > j + 1$

$\mathcal{D}$ computes $r' \leftarrow F_{K_i}(m)$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{D}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, i)$

$\mathcal{D}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(sk_i, m_b; r')$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(c_{kem}, c_{dem})\}$

$\mathcal{D}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

Dec query (c_{kem}, c_{dem})

if $(c_{kem}, c_{dem}) \in \mathcal{S}$

return $\perp$

if $(c_{kem}, \cdot) \in \mathcal{Q}$

$\mathcal{D}$ obtains corresponding i

$k_{dem} \leftarrow sk_i$

else

$\mathcal{C}$ computes $k_{kem} \leftarrow kemdec(dk^*, c_{kem})$

$\mathcal{D}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{D}$ returns $m \leftarrow \mathtt{DEM.D}(k_{dem}, c_{dem})$.

When $\mathcal{A}$ outputs a bit b', $\mathcal{D}$ outputs 1 if $b = b'$ and outputs 0 otherwise. If PRF adversary $\mathcal{D}$ is in the real world then $G_{4/5,j}$ is simulated perfectly. If $\mathcal{D}$ is in the random world then he provides a perfect simulation for $G_{4/5,j+1}$. Without loss of generality, we assume that hybrid games j and $j+1$ have the largest difference. Hence,

$$\begin{aligned}
\mathbb{P}[G_4^{\mathcal{A}}] - \mathbb{P}[G_5^{\mathcal{A}}] &= \mathbb{P}[G_{4/5,0}^{\mathcal{A}}] - \mathbb{P}[G_{4/5,q_t}^{\mathcal{A}}] \\
&= \sum_{k=0}^{q_t-1} \mathbb{P}[G_{4/5,k+1}^{\mathcal{A}}] - \mathbb{P}[G_{4/5,k}^{\mathcal{A}}] \\
&\leq q_t \cdot (\mathbb{P}[G_{4/5,j+1}^{\mathcal{A}}] - \mathbb{P}[G_{4/5,j}^{\mathcal{A}}]) \\
&= q_t \cdot (\mathbb{P}[\mathrm{REAL}_F^{\mathcal{A}}(\lambda) \Rightarrow 1] - \mathbb{P}[\mathrm{RAND}_F^{\mathcal{A}}(\lambda) \Rightarrow 1]) \\
&= q_t \cdot \mathbf{Adv}_{F,\mathcal{D}}^{\mathrm{prf}}(\lambda).
\end{aligned}$$

□

Finally, a hybrid game related to G_5 can be simulated by a standard IND-ATK symmetric encryption adversary. Specifically, we have:

Lemma 7.3.4. For any adversary $\mathcal{A}$ there exists an adversary $\mathcal{E}$ such that

$$2 \cdot \mathbb{P}[G_5^{\mathcal{A}}] - 1 \leq q_t \cdot \mathbf{Adv}_{\mathtt{DEM},\mathcal{E}}^{\text{ind-atk}}(\lambda).$$

Proof. The adversary $\mathcal{E}$ sets up the simulation by choosing $r \leftarrow_{\$} \mathtt{Rnd}$, setting $\mathcal{P} \leftarrow \emptyset$, $\mathcal{Q} \leftarrow \emptyset$ and ctr $\leftarrow 1$. Adversary $\mathcal{E}$ then generates a KEM key pair

$(ek^*, dk^*) \leftarrow_\$ \texttt{KEM.K}$, and generates DEM keys $sk_1, \ldots, sk_{j-1}, sk_{j+1}, \ldots, sk_q \leftarrow_\$ \texttt{DEM.K}$. Adversary $\mathcal{E}$ then answers $\mathcal{A}$'s queries as follows:

Enc query (ek, m, ϕ)

$\mathcal{E}$ computes $(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek, \phi(r))$

if $ek = ek^*$

if $\phi \in \mathcal{P}$

retrieve corresponding i

else

$\mathcal{P} \leftarrow \mathcal{P} \cup \{(\phi, \text{ctr})\}$ and $i \leftarrow \text{ctr}$

$\text{ctr} \leftarrow \text{ctr} + 1$

if $i \neq j$

generate $r \leftarrow_\$ \texttt{Rnd}_{dem}$ and compute $c_{dem} \leftarrow \texttt{DEM.E}(sk_i, m; r)$

else

$\mathcal{E}$ submits (m, m) to its **LR** oracle, receives $c_{dem} \leftarrow \texttt{DEM.E}(sk_j, m)$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{E}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, i)$

$\mathcal{E}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$

else

$\mathcal{E}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{E}$ computes $c_{dem} \leftarrow \texttt{DEM.E}(k_{dem}, m; r_{dem})$

$\mathcal{E}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

LR query (m_0, m_1, ϕ)

$\mathcal{E}$ computes $(c_{kem}, k_{kem}) \leftarrow \texttt{KEM.E}(ek; \phi(r))$

if $\phi \in \mathcal{P}$

retrieve corresponding i

else

$\mathcal{P} \leftarrow \mathcal{P} \cup \{(\phi, \text{ctr})\}$ and $i \leftarrow \text{ctr}$

$\text{ctr} \leftarrow \text{ctr} + 1$

$\mathcal{E}$ generates $r \leftarrow_{\$} \mathtt{Rnd}_{DEM}$

if $i < q_t - j$

$\mathcal{E}$ computes $c_{dem} \leftarrow \mathtt{DEM.E}(sk_i, m_0; r)$

else if $i = q_t - j$,

$\mathcal{E}$ submits (m_0, m_1) to his **LR** oracle, receives $c_{dem} \leftarrow \mathtt{DEM.E}(sk_j, m_b)$

else

$\mathcal{E}$ computes $c_{dem} \leftarrow \mathtt{DEM.E}(sk_i, m_1; r)$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(c_{kem}, c_{dem})\}$

if $(c_{kem}, \cdot) \notin \mathcal{Q}$

$\mathcal{E}$ sets $\mathcal{Q} \leftarrow \mathcal{Q} \cup (c_{kem}, i)$

$\mathcal{E}$ returns (c_{kem}, c_{dem}) to $\mathcal{A}$.

Dec query (c_{kem}, c_{dem})

if $(c_{kem}, c_{dem}) \in \mathcal{S}$

return $\perp$

if $(c_{kem}, \cdot) \in \mathcal{Q}$

$\mathcal{E}$ obtains corresponding i

if $i \neq j$

return $\mathtt{DEM.D}(sk_i, c_{dem})$

else

submit c_{dem} to decryption oracle and forward the answer to $\mathcal{A}$ else

$\mathcal{E}$ computes $k_{kem} \leftarrow \mathtt{KEM.D}(dk^*, c_{kem})$

$\mathcal{E}$ computes $(k_{dem}, r_{dem}) \leftarrow f(k_{kem})$

$\mathcal{E}$ returns $m \leftarrow \mathtt{DEM.D}(k_{dem}, c_{dem})$.

When $\mathcal{A}$ halts and outputs bit b, adversary $\mathcal{E}$ halts and outputs the same bit b. If we let $G_{5,j}$ denote the game in which **LR** queries for key $i \leq q_t - j$ return an encryption of m_0 and **LR** queries for key $i > q_t - j$ return an

encryption of m_1, then

$$\begin{aligned} 2 \cdot \mathbb{P}[G_5^{\mathcal{A}} \Rightarrow 1] - 1 &= \mathbb{P}[\mathcal{A}^{G_{5,q_t}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_{5,0}} \Rightarrow 1] \\ &\leq q_t \cdot (\mathbb{P}[\mathcal{A}^{G_{5,j+1}} \Rightarrow 1] - \mathbb{P}[\mathcal{A}^{G_{5,j}} \Rightarrow 1]) \\ &= q_t \cdot (\mathbb{P}[\mathcal{E} \Rightarrow 1 \mid b = 1] - \mathbb{P}[\mathcal{E} \Rightarrow 1 \mid b = 0]) \\ &= q_t \cdot \mathbf{Adv}^{\text{ind-atk}}_{\mathtt{DEM},\mathcal{E}}(\lambda). \end{aligned}$$

□

The theorem follows by combining all of the previous inequalities. □

7.4 Instantiations

In this section we investigate how the components needed for the construction of KEM-PKE in Figure 7.6 can be efficiently instantiated.

Recall that in Theorem 6.1.2 of Section 6.1 we used a very similar scheme that also made use of a KDF, a DEM, and a PRF as building blocks. It should therefore come as no surprise that Theorem 6.1.2 is a special case of the theorem we just proved. Theorem 6.1.2 made no mention of the KEM/DEM framework, but the theorem implicitly used a KEM that is secure in the FV-RRA-CPA game (Figure 7.4). The description of the KEM, which we call DDH-KEM, can be seen in Figure 7.7, and the next theorem establishes its security in the function-vector model. The proof follows easily by using the relevant game hops that appear in the proof of Theorem 6.1.2.

Theorem 7.4.1. Let $\boldsymbol{\Phi}$ be the set of δ-hard-to-invert vectors of functions on $\{0,1\}^k$. Consider any polynomial-size vector of functions $\boldsymbol{\phi} \in \boldsymbol{\Phi}$ and any equality-pattern respecting, $\boldsymbol{\phi}$-FV-RRA-CPA adversary $\mathcal{A}$ against DDH-KEM in Figure 7.7. If $\mathcal{A}$ uses only one randomness index, then there exists an adversary

Alg. DDH-KEM.K(1^λ):	Alg. DDH-KEM.E(ek):	Alg. DDH-KEM.D(dk, c):
$g_1, \ldots, g_k \leftarrow_\$ \mathbb{G}$	$r \leftarrow_\$ \{0,1\}^k$	$k \leftarrow c^x$
$x \leftarrow_\$ \mathbb{Z}_p$	$c = \prod_{i=1}^k g_i^{r_i}$	return c_2.
$ek = (g_1, \ldots, g_k, g_1^x \ldots, g_k^x)$	$k = \prod_{i=1}^k (g_i^x)^{r_i}$	
$dk = x$.	return (c_1, c_2) .	

Figure 7.7: The KEM DDH-KEM implicitly used in Theorem 6.1.2.

$\mathcal{B}$ such that

$$\mathbf{Adv}^{\boldsymbol{\phi}\text{-fv-rra-cpa}}_{\text{DDH-KEM},\mathcal{A}}(\lambda) \leq 2 \cdot \mathbf{Adv}^{k\text{-ddh}}_{\mathbb{G},\mathcal{B}}(\lambda) + \sqrt[3]{512\delta k p^4}.$$

If we combine this previous theorem with Theorem 7.3.1, then the bound on the advantage of the PKE adversary will be slightly different to the bound obtained in Theorem 6.1.2. We will explain these discrepancies forthwith.

Firstly, in the first game hop of the proof of Theorem 7.3.1, the keys output by KEM.E($ek^*; \phi(r)$) are replaced with uniformly random keys for all ϕ. It is in fact only necessary to replace the outputs of the KEM for functions ϕ that appear in the **LR** queries, but since the function ϕ can appear in both **Enc** and **LR** queries, the proof required us to replace all outputs encrypted with the target public key. In the FV-RRA-ATK, the restrictions mean that only the identity function can be queried with the **LR** oracle, and it may not appear in an **Enc** query. Hence, for the function-vector game we always have $q_t = 1$. Secondly, the proof of Theorem 6.1.2 modifies the public key and the later game hops use the modified version of the public key. When combining the proofs of Theorems 7.3.1 and 7.4.1, the KEM key is modified in order to then replace the KEM key with a random value, but then the KEM key is switched back to its original form before proceeding with the other game hops. Hence, the general theorem (Theorem 7.3.1) requires twice as many DDH game hops, resulting in a looser bound.

In addition to achieving RRA-KEM security via the method above, there

are also many alternatives that would allow us to construct secure RRA-KEM schemes. In the previous chapters we have considered how PKE schemes may be modified by RKA-PRFs, CIS hash functions and auxiliary-input reconstructive extractors. Each of these primitives can also be applied to KEM schemes in order to achieve security against related randomness attacks, though not necessarily the strongest type. For example, public-keys may need to be restricted, or functions may only be selectively chosen. Since these techniques are very similar to what we have seen in previous chapters, and also because they will not provide us with PKE schemes that are secure against alternative classes of functions, we will not discuss these variations further.

7.5 Conclusions

In this chapter we have generalised the approach used in constructing the modified BHHO scheme (Section 6.1.1, Figure 6.2) by showing how to use the KEM/DEM paradigm in order to achieve related randomness security for PKE schemes. This main result of this chapter (Theorem 7.3.1) provides alternatives to the transforms that appeared in Chapters 5 and 6, and the result of Lemma 7.2.2 gives some indication that achieving related randomness security via this route may be easier than using the corresponding results from the previous two chapters. However, whilst it may be easier to prove results in this setting, the security bounds are likely to be somewhat looser, as can be seen by comparing Theorem 5.4.1 and Theorem 7.3.1.

Chapter 8

Conclusions

In this thesis we have analysed public-key encryption schemes in extended attack models. These extended attack models can be categorised into two distinct types, with each type corresponding to a distinct part of the thesis. In the first part of this thesis we studied cold boot attacks, which are practical attacks that are not covered by standard security games, such as the indistinguishability notion. Observing matters from the attacker's perspective, we showed how to recover private keys from a cold boot attack. In particular, our attacks targeted RSA private keys, and elliptic curve implementations in OpenSSL and PolarSSL. We provided two algorithms for recovering private keys. The first was a maximum-likelihood based approach, which was very successful in practice, but lacked a rigorous theoretical analysis of its success rate. The alternative algorithm used the multinomial test as a way of filtering potential solutions. This algorithm has a simple theoretical analysis of its success rate, but we were unable to provide an analysis of the running-time. An area for further research would be to bound the success of the maximum-likelihood approach, or to bound the running-time of the multinomial algorithm. Additionally, it would be interesting to explore whether there are any practical counter-measures to prevent such attacks. None so far has appeared in the

literature as far as we are aware.

In the second part of this thesis, when considering extended attack models, we studied how to protect against these types of attacks rather than how to exploit them. The specific attack under review was the related randomness attack, which was introduced in this thesis. In this type of attack, an adversary is allowed to see encryptions under related randomness values, rather than having uniform randomness for every encryption. This type of attack is designed to simulate events such as failures of random number generators. We formalised several variants of this model, which restricted adversaries in certain ways. For example, in our weaker variants we restricted either the public keys or the functions that could be used in an adversary's oracle queries. For each variant of the security game, we were able to provide a generic transform that converts a standard, secure PKE scheme (in the indistinguishability sense) into a PKE scheme that is secure with respect to our related randomness notion. In the final chapter we extended these notions to Key Encapsulation Mechanisms, and developed connections with the previous chapters. Open problems include the design of stronger RKA-PRFs, CIS hash functions, and auxiliary-input reconstructive extractors. As we have seen, each of these primitives can be combined with standard PKE schemes in order to protect against certain types of related randomness attack. Therefore, further advances with regards to each of these primitives will immediately yield stronger results in the related randomness attack setting. Furthermore, it may be interesting to increase the power of an adversary by allowing him to tamper with private keys as well as the randomness. As previously discussed, there is already some work beginning to emerge in this direction.

Bibliography

[1] Michel Abdalla, Fabrice Benhamouda, Alain Passelègue, and Kenneth G. Paterson, *Related-key security for pseudorandom functions beyond the linear barrier*, Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I (Juan A. Garay and Rosario Gennaro, eds.), Lecture Notes in Computer Science, vol. 8616, Springer, 2014, pp. 77–94.

[2] Martin R. Albrecht and Carlos Cid, *Cold boot key recovery by solving polynomial systems with noise*, ACNS (Javier Lopez and Gene Tsudik, eds.), Lecture Notes in Computer Science, vol. 6715, 2011, pp. 57–72.

[3] Andrew Becherer, Alex Stamos, and Nathan Wilcox, *Cloud computing security: Raining on the trendy new parade*, BlackHat USA (2009).

[4] Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, Hovav Shacham, and Scott Yilek, *Hedged public-key encryption: How to protect against bad randomness*, ASIACRYPT (Mitsuru Matsui, ed.), Lecture Notes in Computer Science, vol. 5912, Springer, 2009, pp. 232–249.

[5] Mihir Bellare and David Cash, *Pseudorandom functions and permutations provably secure against related-key attacks*, in Rabin [67], pp. 666–684.

[6] Mihir Bellare, David Cash, and Rachel Miller, *Cryptography secure against related-key attacks and tampering*, ASIACRYPT (Dong Hoon Lee and Xiaoyun Wang, eds.), Lecture Notes in Computer Science, vol. 7073, Springer, 2011, pp. 486–503.

[7] Mihir Bellare and Tadayoshi Kohno, *A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications*, EUROCRYPT (Eli Biham, ed.), Lecture Notes in Computer Science, vol. 2656, Springer, 2003, pp. 491–506.

[8] Mihir Bellare, Kenneth G. Paterson, and Susan Thomson, *RKA security beyond the linear barrier: IBE, encryption and signatures*, in Wang and Sako [78], pp. 331–348.

[9] Mihir Bellare and Phillip Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993. (Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, eds.), ACM, 1993, pp. 62–73.

[10] ———, *Optimal asymmetric encryption*, EUROCRYPT (Alfredo De Santis, ed.), Lecture Notes in Computer Science, vol. 950, Springer, 1994, pp. 92–111.

[11] ———, *The security of triple encryption and a framework for code-based game-playing proofs*, in Vaudenay [77], pp. 409–426.

[12] Mike Bendel, *Hackers describe PS3 security as epic fail, gain unrestricted access*, 2011, `http://www.exophase.com/20540/hackers-describe-ps3-security-as-epic-fail-gain-unrestricted-access/`.

[13] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren, *Factoring RSA keys from certified smart cards: Coppersmith in the wild*, Cryptology ePrint Archive, Report 2013/599, 2013, `http://eprint.iacr.org/`.

[14] Eli Biham, *New types of cryptoanalytic attacks using related keys (extended abstract)*, Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings (Tor Helleseth, ed.), Lecture Notes in Computer Science, vol. 765, Springer, 1993, pp. 398–409.

[15] Bitcoin.org, *Android security vulnerability*, 2013, `http://bitcoin.org/en/alert/2013-08-11-android`.

[16] Dan Boneh, *Twenty years of attacks on the RSA cryptosystem*, NOTICES OF THE AMS **46** (1999), 203–213.

[17] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky, *Circular-secure encryption from decision Diffie-Hellman*, CRYPTO (David Wagner, ed.), Lecture Notes in Computer Science, vol. 5157, Springer, 2008, pp. 108–125.

[18] Joppe W. Bos, Craig Costello, Patrick Longa, and Michael Naehrig, *Selecting elliptic curves for cryptography: An efficiency and security analysis*, Cryptology ePrint Archive, Report 2014/130, 2014, `http://eprint.iacr.org/2014/130`.

[19] David Brumley and Dan Boneh, *Remote timing attacks are practical*, Computer Networks **48** (2005), no. 5, 701–716.

[20] Don Coppersmith, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, J. Cryptology **10** (1997), no. 4, 233–260.

[21] Ronald Cramer and Victor Shoup, *Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack*, SIAM J. Comput. **33** (2003), no. 1, 167–226.

[22] Debian, *Debian Security Advisory DSA-1571-1: OpenSSL – predictable random number generator*, 2008, `http://www.debian.org/security/2008/dsa-1571`.

[23] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan, *Public-key encryption schemes with auxiliary inputs*, TCC (Daniele Micciancio, ed.), Lecture Notes in Computer Science, vol. 5978, Springer, 2010, pp. 361–381.

[24] Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai, *On the (im)possibility of cryptography with imperfect randomness*, FOCS, IEEE Computer Society, 2004, pp. 196–205.

[25] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs, *Security analysis of pseudo-random number generators with input: /dev/random is not robust*, IACR Cryptology ePrint Archive **2013** (2013), 338.

[26] Leo Dorrendorf, Zvi Gutterman, and Benny Pinkas, *Cryptanalysis of the random number generator of the Windows operating system*, ACM Trans. Inf. Syst. Secur. **13** (2009), no. 1.

[27] Peter Elias, *List decoding for noisy channels*, Technical Report 335, Research Laboratory of Electronics, MIT, 1957.

[28] ———, *Error-correcting codes for list decoding*, IEEE Transactions on Information Theory **37** (1991), no. 1, 5–12.

[29] Eiichiro Fujisaki and Tatsuaki Okamoto, *Secure integration of asymmetric and symmetric encryption schemes*, CRYPTO (Michael J. Wiener, ed.), Lecture Notes in Computer Science, vol. 1666, Springer, 1999, pp. 537–554.

[30] Taher El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), no. 4, 469–472.

[31] Ian Goldberg and David Wagner, *Randomness and the Netscape browser*, 1996, `http://www.drdobbs.com/windows/184409807`.

[32] Vipul Goyal, Adam O'Neill, and Vanishree Rao, *Correlated-input secure hash functions*, TCC (Yuval Ishai, ed.), Lecture Notes in Computer Science, vol. 6597, Springer, 2011, pp. 182–200.

[33] Venkatesan Guruswami, *Algorithmic results in list decoding*, Foundations and Trends in Theoretical Computer Science **2** (2006), no. 2.

[34] Zvi Gutterman and Dahlia Malkhi, *Hold your sessions: An attack on java session-id generation*, CT-RSA (Alfred Menezes, ed.), Lecture Notes in Computer Science, vol. 3376, Springer, 2005, pp. 44–57.

[35] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman, *Analysis of the linux random number generator*, IEEE Symposium on Security and Privacy, IEEE Computer Society, 2006, pp. 371–385.

[36] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten, *Lest we remember: Cold boot attacks on encryption keys*, USENIX Security Symposium (Paul C. van Oorschot, ed.), USENIX Association, 2008, pp. 45–60.

[37] ———, *Lest we remember: cold-boot attacks on encryption keys*, Commun. ACM **52** (2009), no. 5, 91–98.

[38] Darrel Hankerson, Alfred Menezes, and Scott Vanstone, *Guide to Elliptic Curve Cryptography*, Springer, 2004.

[39] Mustapha Hedabou, Pierre Pinel, and Lucien Bénéteau, *A comb method to render ECC resistant against side channel attacks*, Cryptology ePrint Archive, Report 2004/342, 2004, `http://eprint.iacr.org/2004/342`.

[40] Mustapha Hedabou, Pierre Pinel, and Lucien Bénéteau, *Countermeasures for preventing comb method against SCA attacks*, Information Security Practice and Experience, First International Conference, ISPEC 2005, Singapore, April 11-14, 2005, Proceedings (Robert H. Deng, Feng Bao, HweeHwa Pang, and Jianying Zhou, eds.), LNCS, vol. 3439, Springer, 2005, pp. 85–96.

[41] Rafi Heiman, *A note on discrete logarithms with special structure*, Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28, 1992, Proceedings (Rainer A. Rueppel, ed.), Lecture Notes in Computer Science, vol. 658, Springer, 1992, pp. 454–457.

[42] Wilko Henecka, Alexander May, and Alexander Meurer, *Correcting errors in RSA private keys*, in Rabin [67], pp. 351–369.

[43] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman, *Mining your Ps and Qs: Detection of widespread weak keys in network devices*, Proceedings of the 21st USENIX Security Symposium, August 2012.

[44] Nadia Heninger and Hovav Shacham, *Reconstructing RSA private keys from random key bits*, CRYPTO (Shai Halevi, ed.), Lecture Notes in Computer Science, vol. 5677, Springer, 2009, pp. 1–17.

[45] Mathias Herrmann and Alexander May, *Solving linear equations modulo divisors: On factoring given any bits*, Advances in Cryptology - ASIACRYPT 2008, 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia,

December 7-11, 2008. Proceedings (Josef Pieprzyk, ed.), Lecture Notes in Computer Science, vol. 5350, Springer, 2008, pp. 406–424.

[46] Jakob Jonsson and Burton S. Kaliski, *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1 (RFC 3447)*, 2003, `https://www.ietf.org/rfc/rfc3447.txt`.

[47] Abdel Alim Kamal and Amr M. Youssef, *Applications of SAT solvers to AES key recovery from decayed key schedule images*, IACR Cryptology ePrint Archive **2010** (2010), 324.

[48] Seny Kamara and Jonathan Katz, *How to encrypt with a malicious random number generator*, FSE (Kaisa Nyberg, ed.), Lecture Notes in Computer Science, vol. 5086, Springer, 2008, pp. 303–315.

[49] Lars R. Knudsen, *Cryptanalysis of LOKI91*, Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings (Jennifer Seberry and Yuliang Zheng, eds.), Lecture Notes in Computer Science, vol. 718, Springer, 1992, pp. 196–208.

[50] Paul C. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings (Neal Koblitz, ed.), Lecture Notes in Computer Science, vol. 1109, Springer, 1996, pp. 104–113.

[51] Noboru Kunihiro and Junya Honda, *RSA meets DPA: recovering RSA secret keys from noisy analog data*, Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings (Lejla Batina and Matthew Robshaw, eds.), Lecture Notes in Computer Science, vol. 8731, Springer, 2014, pp. 261–278.

[52] Noboru Kunihiro, Naoyuki Shinohara, and Tetsuya Izu, *Recovering RSA secret keys from noisy key bits with erasures and errors*, Public Key Cryptography (Kaoru Kurosawa and Goichiro Hanaoka, eds.), Lecture Notes in Computer Science, vol. 7778, Springer, 2013, pp. 180–197.

[53] Kaoru Kurosawa and Yvo Desmedt, *A new paradigm of hybrid encryption scheme*, Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings (Matthew K. Franklin, ed.), Lecture Notes in Computer Science, vol. 3152, Springer, 2004, pp. 426–442.

[54] D. N. Lawley, *A general method for approximating to the distribution of likelihood ratio criteria*, Biometrika **43** (1956), no. 3/4, pp. 295–303 (English).

[55] Hyung Tae Lee, HongTae Kim, Yoo-Jin Baek, and Jung Hee Cheon, *Correcting errors in private keys obtained from cold boot attacks*, Information Security and Cryptology - ICISC 2011 - 14th International Conference, Seoul, Korea, November 30 - December 2, 2011. Revised Selected Papers (Howon Kim, ed.), Lecture Notes in Computer Science, vol. 7259, Springer, 2011, pp. 74–87.

[56] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter, *Public keys*, CRYPTO (Reihaneh Safavi-Naini and Ran Canetti, eds.), Lecture Notes in Computer Science, vol. 7417, Springer, 2012, pp. 626–642.

[57] Chae Hoon Lim and Pil Joong Lee, *More flexible exponentiation with precomputation*, Advances in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings (Yvo Desmedt, ed.), Lecture Notes in Computer Science, vol. 839, Springer, 1994, pp. 95–107.

[58] Stefan Lucks, *Ciphers secure against related-key attacks*, in Roy and Meier [71], pp. 359–370.

[59] Alexander May, *Using lll-reduction for solving RSA and factorization problems*, The LLL Algorithm - Survey and Applications (Phong Q. Nguyen and Brigitte Vallée, eds.), Information Security and Cryptography, Springer, 2010, pp. 315–348.

[60] Kai Michaelis, Christopher Meyer, and Jörg Schwenk, *Randomly failed! the state of randomness in current java implementations*, CT-RSA (Ed Dawson, ed.), Lecture Notes in Computer Science, vol. 7779, Springer, 2013, pp. 129–144.

[61] Bodo Möller, *Improved techniques for fast exponentiation*, Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers (Pil Joong Lee and Chae Hoon Lim, eds.), Lecture Notes in Computer Science, vol. 2587, Springer, 2002, pp. 298–312.

[62] David M'Raïhi, David Naccache, David Pointcheval, and Serge Vaudenay, *Computational alternatives to random number generators*, Selected Areas in Cryptography '98, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings (Stafford E. Tavares and Henk Meijer, eds.), Lecture Notes in Computer Science, vol. 1556, Springer, 1998, pp. 72–80.

[63] Moni Naor and Omer Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, J. ACM **51** (2004), no. 2, 231–262.

[64] Kenneth G. Paterson, Antigoni Polychroniadou, and Dale L. Sibborn, *A coding-theoretic approach to recovering noisy RSA keys*, in Wang and Sako [78], pp. 386–403.

[65] Kenneth G. Paterson, Jacob C. N. Schuldt, and Dale L. Sibborn, *Related randomness attacks for public key encryption*, IACR Cryptology ePrint Archive **2014** (2014), 337.

[66] T. Pornin, *Deterministic usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)*, Internet Requests for Comments, August 2013, `http://www.rfc-editor.org/rfc/rfc6979.txt`.

[67] Tal Rabin (ed.), *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, Lecture Notes in Computer Science, vol. 6223, Springer, 2010.

[68] Thomas Ristenpart and Scott Yilek, *When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography*, NDSS, The Internet Society, 2010.

[69] Phillip Rogaway, *Nonce-based symmetric encryption*, in Roy and Meier [71], pp. 348–359.

[70] Phillip Rogaway and Thomas Shrimpton, *A provable-security treatment of the key-wrap problem*, in Vaudenay [77], pp. 373–390.

[71] Bimal K. Roy and Willi Meier (eds.), *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised papers*, Lecture Notes in Computer Science, vol. 3017, Springer, 2004.

[72] Santanu Sarkar, Sourav Sen Gupta, and Subhamoy Maitra, *Reconstruction and error correction of RSA secret parameters from the MSB side*, WCC 2011 - Workshop on coding and cryptography (Paris, France), April 2011, pp. 7–16.

[73] Santanu Sarkar and Subhamoy Maitra, *Side channel attack to actual cryptanalysis: Breaking CRT-RSA with low weight decryption exponents*, Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings (Emmanuel Prouff and Patrick Schaumont, eds.), Lecture Notes in Computer Science, vol. 7428, Springer, 2012, pp. 476–493.

[74] Claude E. Shannon, *A mathematical theory of communication*, Bell System Technical Journal **27** (1948), 379–423 and 623–656.

[75] Paul J. Smith, Donald S. Rae, Ronald W. Manderscheid, and Sam Silbergeld, *Approximating the moments and distribution of the likelihood ratio statistic for multinomial goodness of fit*, Journal of the American Statistical Association **76** (1981), no. 375, pp. 737–740 (English).

[76] Alex Tsow, *An improved recovery algorithm for decayed AES key schedule images*, Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers (Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, eds.), Lecture Notes in Computer Science, vol. 5867, Springer, 2009, pp. 215–230.

[77] Serge Vaudenay (ed.), *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, Lecture Notes in Computer Science, vol. 4004, Springer, 2006.

[78] Xiaoyun Wang and Kazue Sako (eds.), *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, Lecture Notes in Computer Science, vol. 7658, Springer, 2012.

[79] Hoeteck Wee, *Public key encryption against related key attacks*, Public Key Cryptography (Marc Fischlin, Johannes Buchmann, and Mark Manulis, eds.), Lecture Notes in Computer Science, vol. 7293, Springer, 2012, pp. 262–279.

[80] D. A. Williams, *Improved likelihood ratio tests for complete contingency tables*, Biometrika **63** (1976), no. 1, pp. 33–37 (English).

[81] Scott Yilek, *Resettable public-key encryption: How to encrypt on a virtual machine*, CT-RSA (Josef Pieprzyk, ed.), Lecture Notes in Computer Science, vol. 5985, Springer, 2010, pp. 41–56.

[82] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage, *When private keys are public: results from the 2008 Debian OpenSSL vulnerability*, Internet Measurement Conference (Anja Feldmann and Laurent Mathy, eds.), ACM, 2009, pp. 15–27.

[83] Tsz Hon Yuen, Cong Zhang, Sherman S.M. Chow, and Siu Ming Yiu, *Related randomness attacks for public key cryptosystems*, Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (New York, NY, USA), ASIA CCS '15, ACM, 2015, pp. 215–223.