UNIVERSITY OF CALIFORNIA, SAN DIEGO

Public-Key Encryption Secure in the Presence of Randomness Failures

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Scott Christopher Yilek

Committee in charge:

Professor Daniele Micciancio, Chair
Professor Mihir Bellare, Co-Chair
Professor Samuel Buss
Professor Adriano Garsia
Professor Hovav Shacham

2010

DEDICATION

To Jess.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Graduate school is difficult, and would be impossible without the help and support of many mentors, colleagues, friends, and family. I have many to thank.

First, I thank my advisor, Daniele Micciancio. He always allowed me to find my own way and never discouraged me from pursuing the projects I was most interested in. For this I will always be grateful. Additionally, Daniele was always there to give valuable advice. I knew that I could talk to him about any issue and get an honest, thoughtful, and insightful opinion on how to proceed. Thank you, Daniele.

Next, I thank Mihir Bellare, who has been a fantastic mentor. Mihir has always stressed the importance of going against the grain, stepping back and asking if your work is truly important, and trying to find projects that will have high impact. I know that I will be a better researcher because of the influence he has had on me.

I also thank Hovav Shacham. I was frustrated with graduate school in the Summer of 2008 until Hovav proposed a security project that we could work on together. The project reinvigorated me and helped launch one of my most productive and enjoyable periods of grad school. Additionally, Hovav has always been someone to whom I can send emails with random ideas and always receive a thoughtful response.

In addition to those above, I thank my close friend and collaborator, Tom Ristenpart, for making my time at UCSD both more productive and more enjoyable. I thank Sam Buss and Adriano Garsia for overseeing my dissertation. I thank Julie Conner and the rest of the UCSD CSE staff for all of their assistance during the last five years. I thank the other members of the UCSD Cryptography Group during my time at UC San Diego for creating such a vibrant environment in which to do research: David Cash, Shanshan Duan, Tadayoshi Kohno (Yoshi), Vadim Lyubashevsky, Sarah Meiklejohn, Anton Mityagin, Petros Mol, Adriana Palacio, Saurabh Panjwani (Panju), Sriram Ravinarayanan, Todor Ristov, Sarah Shoup, and Panagiotis Voulgaris (Panos). I also thank my close friends, Sat Garcia and Natalie Castellana, for making my years at UCSD so enjoyable.

I thank my parents, John and Chris, for providing endless support and advice. Most importantly, I thank my wife, Jess. It's not easy being married to someone working on a Ph.D., but Jess has always been there for me, supplying an enormous amount of support, encouragement, and love. Thank you, Jess; I love you.

Chapter 3 is an expanded and updated version of parts of an ASIACRYPT 2009 paper "Hedged Public-Key Encryption: How to Protect against Bad Randomness", copyright IACR, that I co-authored with Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, and Hovav Shacham. I was a primary researcher on this paper.

Chapter 4 combines modified versions of parts of two papers: a single-authored paper "Resettable Public-Key Encryption: How to Encrypt on a Virtual Machine" appearing at CT-RSA 2010; and "When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography", appearing at NDSS 2010, copyright the Internet Society, and co-authored with Thomas Ristenpart. I was a primary researcher on both papers.

Chapter 5 is a modified and expanded version of parts of a EUROCRYPT 2009 paper "Possibility and Impossibility Results for Encryption and Commitment Secure under Selective Opening", copyright IACR, that I co-authored with Mihir Bellare and Dennis Hofheinz. I was a primary researcher on this paper.

VITA

2005	Bachelor of Science in Computer Science, *summa cum laude*, University of Minnesota
2007	Master of Science in Computer Science, University of California, San Diego
2010	Doctor of Philosophy in Computer Science, University of California, San Diego

ABSTRACT OF THE DISSERTATION

Public-Key Encryption Secure in the Presence of Randomness Failures

by

Scott Christopher Yilek

Doctor of Philosophy in Computer Science

University of California, San Diego, 2010

Professor Daniele Micciancio, Chair
Professor Mihir Bellare, Co-Chair

Public-key encryption (PKE) is a central tool for protecting the privacy of digital information. To achieve desirable strong notions of security like indistinguishability under chosen-plaintext attack (IND-CPA), it is essential for an encryption algorithm to have access to a source of fresh, uniform random bits. Further, these bits should never be revealed and never reused. In practice, our machines typically generate these random bits with software random number generators (RNGs). Unfortunately, RNGs are prone to problems. The resulting randomness failures can have disastrous consequences for the security of existing PKE schemes that rely on good randomness.

In this dissertation we focus PKE security in the presence of three types of randomness failures: predictable randomness, repeated randomness, and revealed randomness. For predictable randomness, where the encryption algorithm is given random inputs that are predictable to an adversary, we argue that we want PKE schemes that are hedged against bad randomness: if the encryption scheme is given good randomness it provably meets traditional notions like IND-CPA, while if it is given poor randomness, it still provably provides some security. We formalize this security notion and give provably-secure constructions of hedged public-key encryption.

Next, we show how repeated randomness failures, where the encryption algorithm is given random inputs that it was given previously, can occur in practice due to virtual machine snapshots. In particular, we show how many popular web browsers are vulnerable to these failures. We then turn to building PKE schemes that still provide provable security when given repeated randomness. We develop new models of security to capture this situation and prove that a simple and efficient modification to any existing secure scheme gives security under our new models.

Finally, we study the strange effects revealed randomness failures, where the random inputs used for encryption are later revealed to an adversary, can have on public-key encryption security. Specifically, we focus on selective opening attacks. We show that a large class of PKE schemes, called lossy encryption schemes, provably resists selective opening attacks.

Chapter 1

Introduction

If you have ever bought a book or checked your bank account balance online, then you have relied on *public-key cryptography* to keep your sensitive data, such as credit card numbers and passwords, out of the hands of criminals. Public-key cryptography allows parties to communicate securely over an insecure channel (e.g., the Internet) without first sharing any secrets. While public-key techniques can be used to achieve many important security goals such as integrity and authenticity, this thesis will focus on public-key techniques to achieve message *privacy*, namely public-key *encryption*.

1.0.1 Defining Privacy, Provable Security, and the Need for Randomness

For the time being, think of a public-key encryption (PKE) scheme as three programs: key generation, denoted $\mathcal{K}$, outputs a public key pk and a secret key sk; encryption, denoted $\mathcal{E}$, takes input a public key pk and a message m and outputs a ciphertext c; and decryption, denoted $\mathcal{D}$, takes a secret key sk and a ciphertext c and outputs a message m. Decryption should undo encryption, meaning that if we give the decryption program a ciphertext $c = \mathcal{E}(pk, m)$, it should output the original message m.

As we said above, public-key encryption is a tool for keeping messages private. This, of course, leaves the question what precisely does it mean for a PKE scheme to be secure? This was the question Goldwasser and Micali examined in their seminal paper [41] and, in doing so, invented *provable security.*

Provable security is inspired by NP-completeness reductions and gives a way to relate the security of cryptographic protocols to well-studied problems in algorithms and complexity theory. To prove that a given protocol is secure, one first develops precise definitions for the security goal he or she is trying to achieve. Then, one proves that the security of the cryptographic protocol in question reduces to the hardness of some other well-studied problem, for example factoring large numbers. The result is that breaking the security of the protocol leads to solving the hard problem.

Thus, to provide provable security for public-key encryption, Goldwasser and Micali had to first precisely define what it means for a PKE scheme to be secure. One thing that came out of GM is the idea that to meet strong security goals and be useful in a variety of applications, encryption should be *randomized.* By randomized we mean that the encryption algorithm $\mathcal{E}$, when taking input pk and a message m, flips coins and uses the resulting random bits (either 1 for heads or 0 for tails) to aid in the creation of a ciphertext.

To see why we might want randomized encryption, consider the following scenario. A user wants to buy and sell stocks online at a website called `tradenow.com`. Say there are only two possible messages, "buy" and "sell", and the adversary knows that these are the only two possible messages. When the user wants to purchase a stock, he encrypts his choice, "buy", under `tradenow.com`'s public key pk. If encryption is deterministic, the result will be $c = \mathcal{E}(pk, ``buy'')$. The adversary, upon seeing the ciphertext c, wants to determine whether the user is buying or selling. Notice that the adversary knows the public-key pk (since it is public), and also knows the two possible messages the user is sending. The adversary can simply compute $c_1 = \mathcal{E}(pk, ``buy'')$ and $c_2 = \mathcal{E}(pk, ``sell'')$ and compare c_1 and c_2 to the observed ciphertext c. Since encryption is deterministic, c will equal either c_1 or c_2, and the adversary will learn the message.

Now let us look at how randomizing the encryption algorithm can help stop this attack. To encrypt "buy" with a randomized encryption scheme, the user will compute $c = \mathcal{E}(pk, \text{“}buy''; r)$, where r is a random string of, for example, 100 bits. (We will always separate the coin input from the rest of the inputs with a semi-colon.) As above, the adversary knows pk and also knows the two possible messages “buy” and “sell”. However, now the adversary does not know the random input to the encryption routine. If for a fixed pk and message m each different value of r leads to a unique ciphertext (which will usually be the case in the schemes we see later), then to perform an attack similar to above, the adversary would have to enumerate all possible values of r. But, there are 2^{100} possibilities, well more than any reasonable adversary can be expected to try.

From this example, we can see that randomized encryption is necessary to get security in some scenarios, but we still have not said what the right notion of PKE security to target is. Informally, Goldwasser and Micali said that public-key encryption should hide all partial information about the message. They called their security notion semantic security. Nowadays, if we want to show a PKE scheme is semantically secure, we show it meets an equivalent notion of security called indistinguishability under chosen-plaintext attack (IND-CPA). IND-CPA security is formalized by considering an adversary playing a game with a challenger. At the start of the game, the challenger gives the adversary a public key pk generated by the key generation algorithm $\mathcal{K}$. The adversary then chooses two equal-length messages m_0 and m_1 and gives them to the challenger. The challenge randomly chooses a bit b to be either 0 or 1 and returns to the adversary the encryption of m_b under pk. The adversary then tries to guess the bit b, outputting a guess b'. We say the adversary wins the IND-CPA game if it correctly guesses the bit b, i.e., $b = b'$.

A PKE scheme is considered to be IND-CPA secure if all adversaries running in a reasonable amount of time fail to win the IND-CPA game with probability much better than 1/2. (An adversary that randomly guesses the bit will win the game with probability exactly 1/2.) Notice the similarity between this security game and the stock example described above.

Later work strengthened IND-CPA by allowing the adversary the ability to decrypt messages other than the challenge message. This latter notion, called

IND-CCA [60] for indistinguishability under chosen-ciphertext attack, is the typical target for PKE security today. We emphasize that good randomness is similarly crucial for IND-CCA security.

IND-CPA and IND-CCA are strong notions of security for PKE, and provable security allows us to formally argue about whether our schemes meet them. However, all of this relies on the implicit assumption that users who encrypt have access to a supply of uniform random bits.

1.0.2 Randomness Generation in Practice

Since cryptographic protocols like public-key encryption rely on the availability of uniform random bits for each operation, and since we want our computers to perform these operations, we need a way for computers to simulate coin flipping. The most common way this is done in practice is to use what is called a software random number generator (RNG).

Implementations of software RNGs are present in many popular programs, including web browsers, SSH clients and servers, and web (HTTPS) servers. Software RNGs work under the assumption that from the point of view of an adversary outside of the system, there are some measurable system events that are highly unpredictable. For example, the exact coordinates of the mouse pointer on the screen at any given time should be unpredictable to an adversary who is not in control of the system or observing the user. Other examples of events that are believed to be somewhat unpredictable are keyboard presses and timings between system interrupts. A software RNG attempts to take this unpredictability and extract randomness from it.

To convert unpredictable system events in to random bits, a software RNG periodically takes digital representations of these events (sometimes referred to as entropy) and stores them in a pool in memory. If some cryptographic operation (e.g., encryption) requests random bits from the RNG, the RNG uses repeated applications of a hash function like SHA-1 to mix together the unpredictable bits in the pool. The result of this mixing is given to the cryptographic operation and is hopefully random. This process of extracting random bits from unpredictable data using hash functions is closely related to the well-studied notion of randomness extractors (c.f., [8]).

One important fact about software RNGs found in current cryptographic libraries is that they are stateful. At any given time the RNG has state consisting of a large pool of unpredictable data and some other variables such as counters. The output of the RNG is then a deterministic function of this state. The state changes every time the RNG outputs a value or new unpredictable data is mixed in to the pool. However, unpredictable data is typically only mixed in periodically, so there are often long periods of time when the state of the RNG is static. Later, we will see later the problems this can cause.

Though software RNGs are extremely useful and mostly work well enough in practice to allow secure use of cryptography, they are not perfect. As with any complex piece of software, RNGs periodically fail in unanticipated ways. In fact, RNGs have a long history of problems [6, 40, 44, 43, 35, 65] leading to spectacular failures.

1.1 This Thesis: PKE in the Presence of Randomness Failures

As we said earlier, long-accepted models of security for public-key encryption implicitly rely on systems having access to a source of unbiased, private random bits. On real systems, these random bits typically come from software random number generators. Unfortunately, real systems (and thus software RNGs) have bugs, fail in unexpected ways, and are used in unanticipated environments and unforeseen ways. The resulting randomness failures can lead to damaging attacks on our existing PKE schemes.

In short, this thesis is about making public-key encryption more secure on real systems. Specifically, we focus on developing a rigorous theory of PKE under three different types of randomness failures: predictable randomness, repeated randomness, and revealed randomness.

To illustrate the problems each randomness failure presents for PKE, we will focus on *hybrid encryption* [32]. Hybrid encryption is the most common way PKE is used in practice, as it is the basis for the key transport used in the SSL/TLS protocol and is also used in numerous email encryption programs. As the name suggests, a hybrid encryption scheme is a hybrid of public-key encryption and symmetric encryption. In short, the public-key encryption scheme is used to encrypt a random symmetric key, while the symmetric encryption algorithm uses this random key to encrypt the actual message. To be more precise, given any public key encryption algorithm $\mathcal{E}$ and any symmetric encryption algorithm SE, we can construct a hybrid encryption algorithm $\bar{\mathcal{E}}$ that combines $\mathcal{E}$ and SE. Algorithm $\bar{\mathcal{E}}$ chooses a random symmetric key K for SE, encrypts K using public-key encryption $\mathcal{E}$, and then encrypts the actual message M using the symmetric key K and the symmetric encryption algorithm SE. The ciphertext generated by $\bar{\mathcal{E}}$ is the pair $\mathcal{E}(pk, K), \mathsf{SE}(K, M)$. To decrypt, one first uses the PKE decryption algorithm $\mathcal{D}$ to retrieve K and then uses that key with the symmetric decryption algorithm SD to retrieve the message.

We note that this closely mirrors what happens essentially every time one connects to a secure (HTTPS) website[1]: the web browser picks a random value R (called the premaster secret) and sends $\mathcal{E}(pk, R)$, where pk is the public key of the secure website and $\mathcal{E}$ is usually RSA encryption with PKCS#1 v1.5 padding. The actual messages one wants to send to the secure website (e.g., passwords) are then sent using a symmetric encryption algorithm with keys derived from R and other public values. One useful way to think of this is that the HTTPS session is like an online[2] version of hybrid encryption.

It is easy to see that for secure hybrid encryption we need good randomness, since the symmetric key K should be chosen freshly and uniformly at random for each encryption. Additionally, the algorithms $\mathcal{E}$ and SE may also require fresh randomness to operate securely, but we will focus on the randomness used to pick the key K for the remainder of this discussion.

[1]We say *essentially every time* because in some cases Diffie-Hellman key agreement is used instead of RSA key transport.

[2]We mean “online” as in “online algorithm”, i.e., an algorithm that operates on input that is not all available immediately upon execution.

We now examine each of the three randomness failures, motivating their study, explaining their effect on PKE, and discussing our central contributions.

1.1.1 Predictable Randomness

The first randomness failure we explore in this dissertation is *predictable randomness*. This randomness failure occurs when the encryption algorithm is given coins that an adversary can predict. Since outputting predictable values is the exact opposite of what a software random number generator is supposed to do, one might think that such failures are uncommon. Unfortunately, they are quite common.

The most recent notable example of a predictable randomness failure was the Debian OpenSSL vulnerability. In 2006, a programmer mistakenly commented out an important line of code in the Debian Linux version of the OpenSSL cryptographic library. Without this important line of code, the only unpredictable data the OpenSSL software random number generator had access to was the process ID. Since the process ID only takes on about 32,000 values, the RNG did not have sufficient entropy and thus the resulting values outputted by the RNG were predictable to an adversary.

The bug was eventually discovered in May 2008 by Luciano Bello [6], but by then the damage had been done. HTTPS web servers with keys generated on affected Debian machines had predictable private keys, and were forced to generate new keys, get new digital certificates, and have CAs revoke the bad certificates. (See [67] for more details on the effects of the bug on SSL/TLS certificates.) Perhaps more frightening, even servers running the buggy Debian that had generated their keys on an unaffected machine were in danger, since routine cryptographic operations like encryption and signing often require good per-operation randomness and fail spectacularly when given predictable randomness.

This is the most recent notable example of how a software RNG in popular software ended up failing and providing predictable values. Unfortunately, this is not an isolated incident. There are also numerous past examples of RNG failures [40, 44, 43, 35, 65], and it is reasonable to expect that more will occur in the future.

Even though unpredictable randomness is necessary to achieve strong notions of security such as IND-CPA, we might hope that existing PKE schemes, even when given predictable coins, still provide *some* level of security. Unfortunately, this is not the case.

To see the effect predictable randomness has on public-key encryption security, let us look at its effect on the hybrid encryption scheme we described earlier. Recall that in hybrid encryption a ciphertext is a pair $\mathcal{E}(pk, K), \mathsf{SE}(K, M)$, where K is chosen by the encryptor uniformly at random. If this encryptor has a random number generator that outputs predictable values, then the key K will be predictable to an adversary. Once the adversary predicts K, it can simply use it to decrypt the second part of the ciphertext pair and recover the message M. Notice that this is true irrespective of the message encrypted. Even if the message is itself completely random (perhaps it is a key generated on another machine), then using hybrid encryption with bad coins still leads to its immediate revelation.

Since hybrid encryption closely mirrors how PKE is used in practice, this problem can be particularly damaging. We might hope that switching to other known PKE schemes would help, but unfortunately, similar attacks can be exhibited on common schemes such as El Gamal [38], Rabin SAEP [56], and RSA-OAEP [25].

We would like to build encryption schemes that still provide some level of security when given bad coins. To motivate how we do this, let us review some of what we know about PKE. On one hand, we know numerous *deterministic* PKE schemes (often called families of injective trapdoor functions). These deterministic schemes are believed to meet some reasonable security notions, the strongest being PRIV security [12], when messages are random or even just unpredictable to an adversary. Alternatively, we know numerous randomized encryption schemes that meet very strong notions of security regardless of the messages being encrypted, yet rely on fresh, unbiased random bits for every encryption. As we saw previously with hybrid encryption, if this per-operation randomness is predictable, the encrypted message is revealed immediately to an adversary.

Notice that if we take a randomized encryption algorithm and fix the coins to some predictable value such as a string of all 0s, then the scheme becomes deterministic. One might reasonably expect that in this case a well-designed randomized PKE scheme should act like good deterministic scheme, i.e., still provide security if the message being encrypted is random or unpredictable. This is the intuition for our improved encryption schemes.

We combine techniques from the state-of-the-art in deterministic PKE (i.e., PRIV security [12] and its variants [16, 23]) with techniques from our existing strong randomized PKE schemes and end up with PKE schemes that are hedged against bad randomness. Such hedged PKE schemes meet standard strong notions of security for randomized encryption, namely IND-CPA, when the per-operation randomness is good. In the case that the randomness is not perfect, a hedged PKE scheme still provably provides security as long as the message and coins together are very unpredictable. We formalize this latter notion of security by extending the PRIV security definition for deterministic encryption, calling the resulting notion IND-CDA for indistinguishability under chosen-distribution attack. We are able to construct hedged PKE schemes in the standard model. More details can be found in Chapter 3

1.1.2 Repeated Randomness

The second randomness failure we explore in this dissertation is repeated randomness. This failure occurs when the encryption algorithm is given the same coins that it was given at some earlier time. As we said above, common software RNGs are *stateful*: their state changes over time as either entropy is added or values are outputted, and the output of the RNG is a deterministic function of its state. Thus, one way a repeated randomness failure can potentially occur is if for some reason the RNG returns to a previous state. We will see how this can happen because of *virtual machines*.

A virtual machine (VM) is a software implementation of a real, physical machine. On a single physical machine one can run many virtual machines that share the resources of the physical machine. To applications running on a virtual machine, it appears as if they are running on a real machine. The use of virtual machines on web servers is in part fueling the cloud computing boom. In this thesis we will mainly be concerned with the use of virtual machines on desktop computers. Desktop virtualization has been popularized by products like VMWare [5] and VirtualBox [4], which allow users to easily set up VMs on their desktop machines.

One useful feature of these VMs is the ability to save the state of a virtual machine. If at any time something goes wrong with the virtual machine (e.g., a virus infection), then the user can simply revert back to a previously saved copy of the machine. This acts much like the save document feature in popular word processing programs: if one makes unwanted changes to an important document he can always revert back to a saved copy.

While saved virtual machine states, called snapshots, are extremely useful, they cause some security concerns. Garfinkel and Rosenblum [39] theorized that the use of VM snapshots could potentially lead to cryptographic insecurities. We will show that the types of problems they described are not just hypothetical by showing real attacks on public-key encryption as used by TLS/SSL clients such as web browsers. Our attacks exploit repeated randomness failures caused by executing multiple times from the same virtual machine snapshot. In short, the state of the stateful software random number generators will be captured in a snapshot, leading to the same values being outputted by the RNG every time execution begins from the snapshot.

Before getting to the details of our attacks, let us see how repeated randomness can have a disastrous effect on PKE security. Consider again a hybrid encryption scheme. Now, say that Alice sends a sensitive message m^* (e.g., her password) to her bank over the Internet using hybrid encryption. The ciphertext will be $\mathcal{E}(pk_B, K), \mathsf{SE}(K, m^*)$, where pk_B is the public key of the bank. Then, at some later time, Alice sends a message m to a malicious website. If Alice has a bad random number generator that repeats values, and the same coins are used in the two encryptions, then the ciphertext sent to the malicious site will be $\mathcal{E}(pk_E, K), \mathsf{SE}(K, m)$, where pk_E is the public key of the malicious site ("E" stands for "evil"). The owner of the malicious site can then use its secret key sk_E

to decrypt $\mathcal{E}(pk_E, K)$, learning the symmetric key K. This, however, was the same symmetric key Alice used to send her password to her bank! Thus, the owner of the malicious site can use K to decrypt $\mathsf{SE}(K, m^*)$, learn the password m^*, and compromise Alice's bank account.[3]

Based on the above problems with PKE and repeated randomness, we show VM reset attacks on many popular web browsers. We show that if a user starts up a web browser and saves the state of her virtual machine, then every time she reloads this saved state and visits a website her browser will send the same secret random value. This is particularly damaging if a user visits a sensitive site like a bank and then later reloads the saved state and visits a malicious website, since it means the malicious website owner knows the secret random value that helps secure the user's banking session.

Since existing PKE schemes are vulnerable to attack when repeated coins are used, and since we show these problems cause serious vulnerabilities in practice, we would like new schemes that perform better and provably still provide as much security as possible. Though researchers have examined the effect state resets have on zero-knowledge protocols [27, 53, 7], identification schemes [15], and multiparty computation protocols [42], no one appears to have looked at deployed cryptographic primitives like PKE in such a setting.

We cannot hope for a perfect solution; as with the predictable randomness case, there are limitations to how much security we can get when randomness is imperfect. In the case of repeated randomness, the inherent limitation is that encrypting the same message to the same public key and with the same randomness results in the same ciphertext.

Given this limitation, our high-level goal will be as follows: if randomness is repeated but anything else changes (i.e.g, the message or the public key), then we still get security. We formalize the details of this security notion in Chapter 4.

Achieving security against resets turns out to be relatively straightforward. We can take any IND-CPA secure PKE scheme and slightly modify it using a pseudorandom function. The resulting scheme will be secure against resets. The

[3]This attack requires that the owner of the malicious site also observe Alice's session with her bank. This might be possible if, for example, Alice is using an unsecured wireless network.

high-level intuition is that while coins are meant to be used only once, many cryptographic objects are intended to be repeatedly used. One such object is the key of a pseudorandom function. Thus, we use the repeated coins as a PRF key to derive new coins. We point out that applying these ideas to key transport in SSL/TLS prevents our attacks.

1.1.3 Revealed Randomness

The last randomness failure we explore is revealed randomness. This failure occurs when the coins use for encryption are later revealed to an adversary. In this thesis we will focus on a particular type of attack that is related to revealed randomness failures called a selective opening attack.

In the strongest selective opening attacks we consider, a large number of users send possibly-related messages to a single server, encrypting the messages with the server's public key. One example of possibly related messages are passwords; many users pick similar passwords, so many users' passwords might be related. Upon seeing the resulting ciphertexts, an adversary can select some subset of the users to corrupt. Upon corrupting a user's computer, the adversary learns the message sent by the user and also the coins used to create the ciphertext. Clearly the messages sent by corrupted users can no longer be private, since the adversary learns them upon corruption. The question we will investigate is whether the messages sent by the *uncorrupted* users are still secure in this scenario.

To those unfamiliar with selective opening attacks, the answer may seem obvious. However, it turns out to be difficult to prove the selective opening security of common encryption schemes that meet traditional notions like IND-CPA. Similar problems have been studied for years under the name adaptively-secure encryption. There are known solutions [26, 28], but they all have significant drawbacks.

We categorize selective opening attacks as a problem related to a randomness failure because the fact that the coins are revealed to an adversary upon corruption appears to cause the main technical difficulties. If users are able to erase the randomness used during encryption and the corrupting adversary only learns the messages sent, then IND-CPA does ensure the security of the uncorrupted ciphertexts.

We point out that no specific attacks are known on IND-CPA secure schemes. In other words, though we cannot prove that IND-CPA secure PKE schemes are also secure against selective opening attacks, we also cannot show that they are *in*secure. Designing any IND-CPA secure scheme that is vulnerable to a selective opening attack is a major open problem in the area. Despite the lack of known attacks, since Goldwasser and Micali's seminal paper our goal has been to prove our PKE schemes secure under all reasonable attacks. Thus, the current state of affairs for selective opening attacks is disquieting.

In Chapter 5 we take a major step to resolving this problem by showing how to prove many known PKE schemes secure under selective opening attacks. In short, we show that any PKE scheme that is lossy is also secure under selective opening attacks. A lossy randomized PKE scheme is one in which the public keys are (computationally) indistinguishable for other, fake public keys, and encryption under these fake keys statistically loses nearly all information about the message being encrypted. Surprisingly, the original PKE scheme from Goldwasser and Micali turns out to have this property!

1.2 Future Directions

We briefly mention some extensions and open problems. We will also discuss open problems in more detail in each individual chapter.

For much of the dissertation we focus on chosen plaintext attacks. Extending the results in each section to a setting with chosen-ciphertext attacks is important and interesting. Some work on this has already been done. As we show in Chapter 4, our results on resettable PKE apply to the CCA setting. In the selective opening attack setting, Fehr, Hofheinz, Kiltz, and Wee [37] recently found a way to achieve full chosen-ciphertext security, while earlier work by Hemenway, Libert, Ostrovsky, and Vergnaud achieved a weaker form of selective opening CCA security [46].

Another interesting future direction is creating PKE schemes that simultaneously meet all of the security notions we examine separately in this dissertation. Specifically, we can ask whether there exist PKE schemes that are simultaneously hedged against bad randomness, resettably secure, and secure against selective opening attacks. This is likely easy in the random oracle model [20], but more difficult in the standard model.

Chapter 2

Background

Before examining randomness failures and their effects on PKE, we give some background information that will be useful for understanding our results in subsequent chapters.

2.1 Notation.

For an integer $n \in \mathbb{N}$, we let $[n]$ denote the set $\{1, \ldots, n\}$. Let $k \in \mathbb{N}$ denote a security parameter and 1^k its unary encoding. Unless stated otherwise, all algorithms in this paper are randomized. We use "PT" for polynomial-time. We call an algorithm efficient if it is PT. We let $\mathbf{m}$ denote a sequence of messages $(m_1, \ldots, m_\ell)$, let $|\mathbf{m}|$ denote the number of messages in the vector, and let $\mathbf{m}[i]$ be the ith message in the vector. For bitstring m we let $|m|$ denote the length of m, and m_i denote the ith bit of m. We let $[A(x_1, \ldots, x_t)]$ be the set of possible outputs of an algorithm A on inputs $x_1, \ldots, x_t$. If A outputs a sequence, then $[A(x_1, \ldots, x_t)]_i$ denotes the set of ith components of the outputs of algorithm A on inputs $x_1, \ldots, x_t$.

2.2 Code-Based Games.

Our security definitions use code-based games [21]. With code-based games, security definitions are formulated by considering a game played with an adversary. Such a game consists of procedures **Initialize** and **Finalize**, as well as procedures for handling oracle calls the adversary can make. At the start of the game, **Initialize** is run and its output is given to the adversary. The adversary then runs and may make oracle calls that are answered by the corresponding game procedures. When the adversary halts with output w, that becomes the input to the **Finalize** procedure and the resulting output of **Finalize** is called the output of the game. We denote by $G^A \Rightarrow y$ the event that game G, when run with adversary A, outputs y. Sometimes we let G^A denote the event $G^A \Rightarrow \mathsf{true}$. We say the running time of an adversary playing a game is the time it takes to run the adversary with the game, so in particular, the time it takes to run game procedures is accounted for in the adversary's running time.

We will often consider adversaries that query the description of a Turing machine. For example, we may have a game with procedure **LR** that, on input the description of an algorithm $\mathcal{M}$, executes $\mathcal{M}$ within the procedure. Since the time it takes to run the game with an adversary counts towards the adversary's running time, a polynomial time adversary must query only descriptions of algorithms that also run in polynomial time. Equivalently, we could consider adversaries that query descriptions of circuits; a polynomial time adversary would only be able to write out the description of a polynomial size circuit, guaranteeing that running the circuit is also polynomial time.

2.3 Hashing

A family of hash functions is a tuple of PT algorithms $\mathcal{H} = (\mathsf{P}_{\mathrm{h}}, \mathsf{K}_{\mathrm{h}}, \mathsf{H})$ with message length $n(\cdot)$ and with the following properties. The randomized parameter generation algorithm P_{h} takes as input a security parameter 1^k and outputs a parameter string π_{h}. The randomized key generation algorithm K_{h} takes as input a parameter string π_{h} and outputs a hash key κ. The deterministic hash evaluation algorithm H takes as input hash key κ and input $x \in \{0,1\}^{n(k)}$ and outputs y. For every k and all $\pi_{\mathrm{h}} \in [\mathsf{P}_{\mathrm{h}}(1^k)]$, let $R(\pi_{\mathrm{h}}) = \{\mathsf{H}(\kappa, x) \ : \ \kappa \in [\mathsf{K}_{\mathrm{h}}(\pi_{\mathrm{h}})] \wedge x \in \{0,1\}^n\}$. We say a hash family has $2^{t(k)}$ bounded range if for all k and all $\pi_{\mathrm{h}} \in [\mathsf{P}_{\mathrm{h}}(1^k]$ it is the case that $|R(\pi_{\mathrm{h}})| \leq 2^{t(k)}$. We say $\mathcal{H}$ has output length ℓ if for all k and all $\pi_{\mathrm{h}} \in [\mathsf{P}_{\mathrm{h}}(1^k)]$, the range $R(\mathit{pars}) = \{0,1\}^{\ell(k)}$.

We say that $\mathcal{H}$ is *universal* if for all k, all $\pi_{\mathrm{h}} \in [\mathsf{P}_{\mathrm{h}}(1^k)]$, and all distinct inputs $x, x' \in \{0,1\}^{n(k)}$, it is the case that

$$\Pr\left[\, \mathsf{H}(\kappa, x) = \mathsf{H}(\kappa, x') \,\right] \leq \frac{1}{|R(\pi_{\mathrm{h}})|} \ ,$$

where the probability is taken over $\kappa \leftarrow_{\$} \mathsf{K}_{\mathrm{h}}(\pi_{\mathrm{h}})$.

We say that $\mathcal{H}$ is *pairwise-independent* if for all k, all $\pi_{\mathrm{h}} \in [\mathsf{P}_{\mathrm{h}}(1^k)]$, all distinct inputs $x, x' \in \{0,1\}^{n(k)}$, and all $y, y' \in R(\pi_{\mathrm{h}})$, it is the case that

$$\Pr\left[\, \mathsf{H}(\kappa, x) = y \wedge \mathsf{H}(\kappa, x') = y' \,\right] \leq \frac{1}{|R(\pi_{\mathrm{h}})|^2} \ ,$$

where the probability is taken over $\kappa \leftarrow_{\$} \mathsf{K}_{\mathrm{h}}(\pi_{\mathrm{h}})$.

2.4 Public-Key Encryption and Hiding Schemes

A public-key encryption (PKE) scheme $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is a tuple of PT algorithms. All algorithms either explicitly or implicitly take as input the security parameter 1^k in unary and are PT in k. The randomized parameter generation algorithm $\mathcal{P}$, on input unary security parameter 1^k, outputs scheme parameters pars. The randomized key generation algorithm $\mathcal{K}$, on input parameters pars, outputs a pair of keys $(\mathit{pk}, \mathit{sk})$. The randomized encryption algorithm $\mathcal{E}$, on input public key pk and message $m \in \mathsf{MsgSpace}_{\mathit{pars}}$, outputs a ciphertext c. Let the set of coins $\mathcal{E}$ uses on input pars and pk be denoted $\mathsf{Coins}_{\mathcal{E}}(\mathit{pars}, \mathit{pk})$ and let $\mathcal{E}(\mathit{pk}, m; r)$ denote encryption of m under public key pk and coins r. Finally, the deterministic decryption algorithm $\mathcal{D}$, on input a secret key sk and ciphertext c, outputs either $\perp$ in the case of failure, or $m \in \mathsf{MsgSpace}_{\mathit{pars}}$. We require that for all $k \in \mathbb{N}$, all $\mathit{pars} \in [\mathcal{P}(1^k)]$, all $(\mathit{pk}, \mathit{sk}) \in [\mathcal{K}(\mathit{pars})]$ and for all $m \in \mathsf{MsgSpace}_{\mathit{pars}}$, it is true that $\mathcal{D}(\mathit{sk}, \mathcal{E}(\mathit{pk}, m)) = m$. We note that many PKE schemes do not have a parameter generation algorithm; for those schemes $\mathcal{P}$ simply outputs the security parameter, i.e., $\mathit{pars} = 1^k$.

If for all k and all $pars \in [\mathcal{P}(1^k)]$ it is the case that $\mathsf{Coins}_{pars} = \emptyset$, then $\mathcal{E}$ is always deterministic and we say that $\mathcal{AE}$ is a deterministic encryption scheme. This is syntactically the same as a family of injective trapdoor functions, so we will use those terms interchangeably.

We will also consider what we call *hiding schemes*. A hiding scheme removes the efficient decryption and correctness requirements from the definition of PKE; the scheme is simply used to hide messages, not necessarily transport them. More formally, a hiding scheme is a tuple of PT algorithms $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}, \mathcal{E})$ with the same properties as the corresponding PKE algorithms described above.

If $\mathbf{m} = (m_1, \ldots, m_\ell)$ and $\mathbf{r} = (r_1, \ldots, r_\ell)$, then when we say $\mathcal{E}(pk, \mathbf{m}\,;\,\mathbf{r})$ we mean component-wise encryption. In other words,

$$\mathcal{E}(pk, \mathbf{m}\,;\,\mathbf{r}) = (\mathcal{E}(pk, \mathbf{m}[1]\,;\,\mathbf{r}[1]), \ldots, \mathcal{E}(pk, \mathbf{m}[\ell]\,;\,\mathbf{r}[\ell]))\ .$$

Similarly, if $\mathbf{c} = \mathcal{E}(pk, \mathbf{m}\,;\,\mathbf{r})$, then we denote by $\mathcal{D}(sk, \mathbf{c})$ component-wise decryption, i.e.,

$$\mathcal{D}(sk, \mathbf{c}) = (\mathcal{D}(sk, \mathbf{c}[1]), \ldots, \mathcal{D}(sk, \mathbf{c}[\ell]))\ .$$

2.4.1 Security Notions

We will consider numerous security notions for PKE and hiding schemes. In later chapters we will also develop new security notions.

Key Indistinguishability. Consider a PKE (or hiding) scheme Π with key generation algorithm $\mathcal{K}$. For many of the proofs in this thesis we will want to instead use an alternate key generation algorithm $\mathcal{K}'$. However, we do not want an adversary who is given the public key to know whether the public key came from the real key generation algorithm $\mathcal{K}$ or the alternate key generation algorithm $\mathcal{K}'$. To capture this property, consider game KEYIND (not shown) parameterized by scheme Π with key generation algorithm $\mathcal{K}$, alternate key generation algorithm $\mathcal{K}'$, and security parameter k. The **Initialize** procedure runs the parameter generation $\mathcal{P}$ from Π to get $pars$ and then flips a bit b. Let $\mathcal{K}_0 = \mathcal{K}$ and $\mathcal{K}_1 = \mathcal{K}'$.

Then, **Initialize** runs $\mathcal{K}_b(pars)$ and returns the public key pk to the adversary. The adversary halts with output a bit b' and the output of **Finalize** is true if $b = b'$ and false otherwise. The key-ind advantage of an adversary A is then

$$\mathbf{Adv}^{\text{key-ind}}_{A,\Pi,\mathcal{K}'}(k) = 2 \cdot \Pr\left[\,\text{KEYIND}^{A}_{\Pi,\mathcal{K}',k} \Rightarrow \mathsf{true}\,\right] - 1 \;.$$

IND-CPA. Let Π be either a PKE scheme or a hiding scheme. An INDCPA adversary is one that plays game INDCPA, found in Figure 2.1, and always queries **LR** with two equal-length messages. We say the ind-cpa advantage of an INDCPA adversary A against Π is

$$\mathbf{Adv}^{\text{ind-cpa}}_{\Pi,A}(k) = 2 \cdot \Pr\left[\,\text{INDCPA}^{A}_{\Pi,k} \Rightarrow \mathsf{true}\,\right] - 1 \;,$$

where the security game is found in Figure 2.1. We say scheme Π is IND-CPA secure if for all PPT INDCPA adversaries A making one **LR** query, the ind-cpa advantage of A against Π is negligible in k. We only need to consider adversaries making one **LR** query because of the following proposition:

Proposition 2.4.1 Let k be a security parameter and Π a hiding scheme. Let A be an INDCPA adversary making at most q queries to the **LR** oracle. Then there exists an INDCPA adversary B making one **LR** query such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\Pi,A}(k) \le q \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\Pi,B}(k) \;.$$

▮

The proposition can be proved with a standard hybrid argument. See [11] for a proof in the more general case of multiple receivers. (Restricting the proof in [11] to one receiver results in the above proposition.)

IND-CCA. Let $\mathcal{AE}$ be a PKE scheme. An INDCCA adversary is one that plays game INDCCA, found in Figure 2.2, and always queries **LR** with two equal-length messages. We say the ind-cca advantage of an INDCCA adversary A against $\mathcal{AE}$

```
procedure Initialize:
$b \leftarrow_{\$} \{0,1\}$
$pars \leftarrow_{\$} \mathcal{P}(1^k)$
$(pk, sk) \leftarrow_{\$} \mathcal{K}(pars)$
Ret $pk$

procedure LR$(m_0, m_1)$:
$c \leftarrow_{\$} \mathcal{E}(pk, m_b)$
Return $c$

procedure Finalize$(b')$:
Ret $(b = b')$
```

Figure 2.1: Security game $\mathrm{INDCPA}_{\Pi,k}$.

```
procedure Initialize:
$b \leftarrow_{\$} \{0,1\}$
$pars \leftarrow_{\$} \mathcal{P}(1^k)$
$(pk, sk) \leftarrow_{\$} \mathcal{K}(pars)$
$S \leftarrow \emptyset$
Ret $pk$

procedure Dec$(c)$:
If $c \in S$ then return $\perp$
$m \leftarrow \mathcal{D}(sk, c)$
Return $m$

procedure LR$(m_0, m_1)$:
$c \leftarrow_{\$} \mathcal{E}(pk, m_b)$
$S \leftarrow S \cup \{c\}$
Return $c$

procedure Finalize$(b')$:
Ret $(b = b')$
```

Figure 2.2: Security game $\mathrm{INDCCA}_{\mathcal{AE},k}$.

is

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\,\mathrm{INDCCA}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true}\,\right] - 1 ,$$

where the security game is found in Figure 2.2. We say scheme $\mathcal{AE}$ is IND-CCA secure if for all PPT INDCCA adversaries A making one **LR** query, the ind-cca advantage of A against $\mathcal{AE}$ is negligible in k. As above, we only need to consider adversaries making one **LR** query, since the proof of the above proposition applies equally well here.

2.5 Trapdoor Functions

We will use families of injective trapdoor functions (TDFs). As noted above, these are syntactically identical to deterministic PKE schemes, but we will sometimes use different notation to avoid confusion. Other times we will use the PKE notation. Formally, a family of injective trapdoor functions with message length $n(\cdot)$ is a tuple of PT algorithms $\mathcal{F} = (\mathsf{P}, \mathsf{K}, \mathsf{F}, \mathsf{F}^{-1})$ with the following properties. The randomized parameter generation algorithm P takes as input security parameter 1^k and outputs parameter string π. The randomized key generation algorithm K takes as input parameters π and outputs a function index σ and trapdoor τ. The deterministic function evaluation algorithm F takes as input a function index σ and $x \in \{0,1\}^{n(k)}$ and outputs a point y. The function inverse algorithm F^{-1} takes as input a trapdoor τ and a point y and outputs a point x. We require the correctness condition that for all 1^k, all $\pi \in [\mathsf{P}(1^k)]$, all $(\sigma, \tau) \in [\mathsf{K}(\pi)]$, and all $x \in \{0,1\}^{n(k)}$, it is the case that $\mathsf{F}^{-1}(\tau, \mathsf{F}(\sigma, x)) = x$.

Lossiness. Lossy trapdoor functions were introduced by Peikert and Waters [58] and have been useful for solving numerous open problems in cryptography. We will rely on them heavily in this dissertation.

We say K_ℓ is an (n, L)-lossy key generation algorithm for $\mathcal{F}$ with message length $n(\cdot)$ if for all k, all $\pi \in [\mathsf{P}(1^k)]$, and all $(\sigma_\ell, \tau_\ell) \in [\mathsf{K}_\ell(\pi)]$ the map $\mathsf{F}(\sigma_\ell, \cdot)$ has image size at most $2^{n(k)-L(k)}$. We say K_ℓ is universal-inducing if $\mathcal{H} = (\mathsf{P}, \mathsf{K}_\ell, \mathsf{F})$ is a family of universal hash functions.

Then, we say a family $\mathcal{F}$ of injective trapdoor functions with message length $n(\cdot)$ is (n, L)-lossy if there exists a PT (n, L)-lossy key generation algorithm K_ℓ (called the lossy key generation algorithm) such that For all PT adversaries A, the key-ind advantage $\mathbf{Adv}^{\text{key-ind}}_{A,\mathcal{F},\mathsf{K}_\ell}(k)$ is negligible in k. If K_ℓ is universal-inducing we say that $\mathcal{F}$ is a family of universal LTDFs (u-LTDFs).

2.6 Pseudorandom Functions.

We will need families of pseudorandom functions (PRFs) for some of our results. Let $\mathsf{Fun} : \mathsf{Keys}_k \times \mathsf{Dom}_k \to \mathsf{Rng}_k$ be a family of functions indexed by a

```
procedure Initialize:                                   Game REAL_{F,k}
$K \leftarrow_{\$} \mathsf{Keys}_k$
Ret $1^k$                                 procedure Finalize($a$):
                                          Ret $a$
procedure Fun($x$):
Return $\mathsf{Fun}(K, x)$
```

```
procedure Initialize:                                   Game RAND_{F,k}
$\mathtt{FunTab} \leftarrow \emptyset$
Ret $1^k$                                 procedure Finalize($a$):
                                          Ret $a$
procedure Fun($x$):
If $\mathtt{FunTab}[x] = \perp$ then
    $\mathtt{FunTab}[x] \leftarrow_{\$} \mathsf{Rng}_k$
Return $\mathtt{FunTab}[x]$
```

Figure 2.3: Security games for pseudorandom function security.

security parameter k. We say the PRF-advantage of a PRF adversary D is

$$\begin{aligned}\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Fun},D}(k) &= \Pr\left[\,\mathrm{REAL}^{D}_{\mathsf{Fun},k} \Rightarrow 1\,\right] \\ &\quad - \Pr\left[\,\mathrm{RAND}^{D}_{\mathsf{Fun},k} \Rightarrow 1\,\right]\ ,\end{aligned}$$

where the security games can be found in Figure 2.3.

Chapter 3

Predictable Randomness

In this chapter we study public-key encryption security when the coins used for encryption may be predictable to an adversary. We first give a high-level overview of the problem before delving into details.

3.1 Overview

As we described in the introduction, many well-known PKE schemes that meet strong notions of security such as IND-CPA completely fail to provide security when the coins used during encryption are predictable. This means that notions of security for PKE that we have long believed to be the "right" ones rely on the implicit assumption that encryptors have access to an unbiased source of random bits. As we described, in practice this assumption often turns out to be unrealistic.

Thus, IND-CPA alone is not sufficient for arguing about PKE security when randomness is predictable, which is unfortunately sometimes the case on real systems.[1] In this chapter we propose a new notion of security for public-key encryption that we call indistinguishability under chosen distribution attack (IND-CDA), and argue that we should aspire to build schemes that are both IND-CPA secure *and* IND-CDA secure. We call such schemes "hedged" against bad randomness.

[1]We note that IND-CCA, IND-CPA's stronger cousin, is also not sufficient for similar reasons. To see this, simply consider an IND-CCA version of the hybrid encryption scheme discussed at length in the introduction.

Building encryption schemes that are hedged against potentially bad assumptions was proposed by Shoup [64]. He was concerned with encryption schemes whose security was explicitly based on hardness assumptions that might be proven false, so he showed how to build an encryption scheme that was proven secure under one assumption (DDH) in the standard model, and was proven secure under a weaker assumption (CDH) in something called the random oracle model. In short, if for some reason DDH turned out to be false[2], all would not necessarily be lost. We note that widely-used hardness assumptions are almost never broken, which is in contrast to RNGs failing relatively frequently.

Other previous work considered something similar to hedging against bad randomness, but in the symmetric encryption setting. Specifically, Rogaway [62] and Rogaway and Shrimpton [63] both reduce the reliance on a random IV in symmetric encryption. The former advocates relying on unique values for each encryption (nonces) instead of uniformly random values. The latter proposes that symmetric encryption schemes should fall back to good PRFs when randomness is bad. In other related work, Kamara and Katz [49] study symmetric encryption security when an adversary can see messages encrypted under adversarially-chosen randomness. However, they still allow perfect randomness for challenge ciphertexts, the idea being that encryptions using bad randomness should not leak information about messages that are encrypted with good randomness. Differing from these works, we focus on the public-key setting, which presents unique challenges since the encryptor does not have access to a random secret symmetric key.

Thus, we need a new notion of security for PKE with bad randomness. Our new notion, IND-CDA, is inspired by the PRIV security notion for deterministic encryption [12], specifically its indistinguishability versions [23, 16]. At a high level, a scheme is IND-CDA secure if encryption hides all information about a message m when encrypted with coins r, as long as the pair (m, r) is sufficiently unpredictable. In other words, even if r is not completely uniform, as long as there is enough unpredictability somewhere in the message and coins, encryption

[2]DDH is now considered a "standard" assumption, so it might seem strange to be concerned about it being false. But, there was a time when it was not so standard and sometimes referred to as the "**D**ay **D**reamers **H**ypothesis".

is secure. In short, schemes that meet IND-CDA borrow some unpredictability from the message to make up for any deficiencies in the unpredictability of the coins.

Why might we expect messages to be unpredictable? Common things one might want to encrypt, such as a long email, are certainly very unpredictable to an adversary. One might also be encrypting random messages such as keys that were generated on a different machine with a good RNG. More philosophically, if messages are not at least somewhat unpredictable, then why encrypt in the first place?

We emphasize that we are considering a scenario where the senders of messages potentially have bad RNGs, but receivers still used good randomness to generate the public and secret keys. We claim this is a reasonable scenario. First, key generation is an infrequent operation, while encryption operations may happen frequently. Second, some programs (e.g., GnuPG [2]) take extra precautions to ensure key generation has access to good randomness. Assuming some amount of good randomness is also necessary, since many past works have shown that much of cryptography is impossible with only a weak random source [52, 34, 24].

To achieve IND-CDA security, we extend the ideas of Boldyreva, Fehr, and O'Neill [23] for achieving PRIV-secure deterministic PKE. We briefly review their ideas. To achieve PRIV security (which turns out to be the same as IND-CDA security for PKE schemes with randomness length 0) we want encryption $\mathcal{E}(pk, m)$ to hide information about m when it has high min-entropy that is independent of pk. This is similar to the setting of strong extractors, where we want $\mathsf{H}(\kappa, m)$ to be (statistically) close to uniform (even given κ) when m has high min-entropy independent of κ. Thus, one idea might be to use a strong extractor for encryption; in other words, define encryption as $\mathcal{E}(pk, m) = \mathsf{H}(pk, m)$. We can construct schemes like this, but unfortunately correct decryption becomes problematic since strong extractors necessarily compress.

To overcome this obstacle, [23] use encryption that is computationally indistinguishable from an extractor. In other words, $\mathcal{E}(pk, m)$ is still injective in m, allowing decryption, but honest public keys are indistinguishable from other keys under which encryption actually does compress and act like a true extractor. This allows one to computationally argue about security while still preserving decryption functionality. BFO call PKE schemes with this property deterministic encryption with hidden universal hash mode. They then show how to build such schemes using universal LTDFs (u-LTDFs) [23, 58].

Given this idea (which is currently the only known way to achieve PRIV-secure PKE in the standard model), we can build *randomized* encryption schemes that are IND-CDA secure. However, IND-CDA is not our only goal; we also need to ensure the resulting schemes are IND-CPA secure. We have two main constructions.

Our first construction composes an IND-CPA secure randomized public-key encryption scheme with a PRIV-secure deterministic PKE scheme. Interestingly, the order of composition turns out to be important. We show that if we first apply a suitable randomized encryption scheme and then apply a PRIV-secure deterministic encryption scheme, then we can achieve both IND-CPA and IND-CDA security.

Our second construction is simpler, yet technically more difficult to prove secure. We show that if we simply pad the message with randomness and then apply a u-LTDF, we can get hedge security. The IND-CDA security follows immediately from the PRIV-security of the u-LTDF. Showing IND-CPA turns out to be more difficult. The difficulty stems from the fact that u-LTDFs are secure when the message and randomness pair (m, r) has high min-entropy *independent of the public key.* However, in the INDCPA security game the adversary is allowed to choose messages that *depend on the public key.* It turns out that we can overcome this issue by carefully setting the parameters and choosing the length of the padded randomness appropriately.

Organization. The rest of the chapter is organized as follows. In the next section we give details of our new security definition IND-CDA and precisely define what it means for a scheme to hedge against bad randomness. In Section 3.3 we describe a variant of the Leftover Hash Lemma (LHL) [48] that we will need for our results. In Section 3.4 we prove two encryption schemes secure in the standard model.

Finally, in Section 3.5 we show how to achieve adaptive security.

3.2 New Security Definition: IND-CDA

Before giving our new definition of PKE security, we give a few brief preliminaries.

Public-Key Encryption. For simplicity, we state our security definitions for PKE schemes $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ with message space $\{0,1\}^{n(k)}$ and coin space $\{0,1\}^{\rho(k)}$, where k is the security parameter.

Sources and Entropy Measures. We generalize the notion of a source to consider a joint distribution on the messages and the randomness with which they will be encrypted. A t-source ($t \geq 1$) with message length $n(\cdot)$ and randomness length $\rho(\cdot)$ is a probabilistic algorithm $\vec{\mathcal{M}}$ that on input 1^k returns a $(t+1)$-tuple $(\mathbf{m}_0, \ldots, \mathbf{m}_{t-1}, \mathbf{r})$ of equal-length vectors, where $\mathbf{m}_0, \ldots, \mathbf{m}_{t-1}$ are over $\{0,1\}^{n(k)}$ and $\mathbf{r}$ is over $\{0,1\}^{\rho(k)}$. We say that $\vec{\mathcal{M}}$ has min-entropy $\mu(\cdot)$ if

$$\Pr\left[\, (\mathbf{m}_b[i], \mathbf{r}[i]) = (m, r) \,\right] \leq 2^{-\mu(k)}$$

for all $k \in \mathbb{N}$, all $b \in \{0, \ldots, t-1\}$, all $i \in \{1, \ldots, |\mathbf{r}|\}$, all $(m, r) \in \{0,1\}^{n(k)} \times \{0,1\}^{\rho(k)}$, and where the probability is over $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r}) \leftarrow_{\$} \vec{\mathcal{M}}(1^k)$. We say it has conditional min-entropy $\mu(\cdot)$ if

$$\Pr\left[\, (\mathbf{m}_b[i], \mathbf{r}[i]) = (m, r) \mid \forall j < i \; (\mathbf{m}_b[j], \mathbf{r}[j]) = (\mathbf{m}'[j], \mathbf{r}'[j]) \,\right] \leq 2^{-\mu(k)}$$

for all $k \in \mathbb{N}$, all $b \in \{0, \ldots, t-1\}$, all i, all (m, r), all vectors $\mathbf{m}', \mathbf{r}'$, and over the coins used by $\vec{\mathcal{M}}$.

A t-source with message length $n(\cdot)$, randomness length $\rho(\cdot)$, and min-entropy $\mu(\cdot)$ is referred to as a (μ, n, ρ)-mr-source when $t = 1$ and $\rho(\cdot) > 0$; a (μ, n)-m-source when $t = 1$ and $\rho(\cdot) = 0$; a (μ, n, ρ)-mmr-source when $t = 2$ and $\rho(\cdot) > 0$; and (μ, n)-mm-source when $t = 2$ and $\rho(\cdot) = 0$. Each "m" indicates the source outputting one message vector and an "r" indicates a randomness vector. When the source has *conditional* min-entropy $\mu(\cdot)$ we write block-source instead of source for each of the above. A $v(\cdot)$-vector source outputs vectors of size $v(k)$ for all k.

```
procedure Initialize:
$pars \leftarrow_{\$} \mathcal{P}(1^k)$
$(pk, sk) \leftarrow_{\$} \mathcal{K}(pars)$
$b \leftarrow_{\$} \{0,1\}$
Ret $pars$

procedure RevealPK():
$\mathsf{pkout} \leftarrow \mathsf{true}$
Ret $pk$

procedure LR($\vec{\mathcal{M}}$):
If $\mathsf{pkout} = \mathsf{true}$ then
    Ret $\perp$
$(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r}) \leftarrow_{\$} \vec{\mathcal{M}}(1^k)$
Ret $\mathcal{E}(pk, \mathbf{m}_b; \mathbf{r})$

procedure Finalize($b'$):
Ret $(b = b')$
```

Figure 3.1: Game $\mathrm{CDA}_{\mathcal{AE},k}$

3.2.1 The Security Definition

We now give our new security definition for public-key encryption, which we call indistinguishability under chosen-distribution attack (IND-CDA). Let $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme. A CDA adversary is one whose **LR** queries are all mmr-sources. Game $\mathrm{CDA}_{\mathcal{AE}}$ of Figure 3.1 provides the adversary with two oracles. The advantage of CDA adversary A is

$$\mathbf{Adv}^{\mathrm{cda}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\, \mathrm{CDA}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true} \,\right] - 1 \, .$$

Discussion. Adversary A can query **LR** with an mmr-source of its choice, an output $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r})$ of which represents choices of message vectors to encrypt and randomness with which to encrypt them. (An alternative formulation might have CDA adversaries query two mr-sources, and distinguish between the encryption of samples taken from one of these. But this would mandate that schemes ensure privacy of messages *and* randomness.) This allows A to dictate a joint distribution on the messages and randomness. In this way it conservatively models even adversarially-subverted random number generators. Multiple **LR** queries are allowed. In the most general case these queries may be adaptive, meaning depend on answers to previous queries.

Given that multiple **LR** queries are allowed, one may ask why an mmr-source needs to produce message and randomness vectors rather than simply a single pair of messages and a single choice of randomness. The reason is that the coordinates in a vector all depend on the same coins underlying an execution of $\vec{\mathcal{M}}$, but the coins underlying the execution of the sources in different queries are independent.

Note that **Initialize** does not return the public key pk to A. A can get it at any time by calling **RevealPK**, but once it does this, **LR** will return $\perp$. The reason is that we inherit from deterministic encryption the unavoidable limitation that encryption cannot hide public-key related information about the plaintexts [12]. (When the randomness has low entropy, the ciphertext itself is such information.)

No encryption scheme is secure when both messages and randomness are predictable. Formally, this means chosen-distribution attacks are trivial when adversaries can query mmr-sources of low min-entropy. Our notions (below) will therefore require security only for sources that have high min-entropy or high conditional min-entropy.

Equality patterns. Suppose A makes a query $\vec{\mathcal{M}}$ which returns $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r}) = ((a, a), (a, a'), (r, r))$ for some $a \neq a'$ and random r. Then it can win trivially because the (two) components of the returned vector $\mathbf{c}$ are equal if $b = 0$ and unequal otherwise. This example points to a fundamental limitation with encryption: equality of plaintext and randomness is leaked by ciphertexts. To have an achievable notion of security, then, we must ensure that CDA adversaries cannot use plaintext-randomness equalities in order to trivially learn the challenge bit b.

We first define an equality pattern, following [12]. For any pair of vectors $(\mathbf{m}, \mathbf{r})$ of length t, the equality pattern is the bit-valued matrix $E^{(\mathbf{m},\mathbf{r})}$ where $E^{(\mathbf{m},\mathbf{r})}_{i,j} = 1$ if $(\mathbf{m}[i], \mathbf{r}[i]) = (\mathbf{m}[j], \mathbf{r}[j])$ and $E^{(\mathbf{m},\mathbf{r})}_{i,j} = 0$ otherwise. This always-symmetric matrix describes the equality relations between all elements of the two vectors. For example, the equality patterns for the pairs of vectors $((a, a), (r, r))$ and $((a, a'), (r, r))$ used in the attack of the last paragraph are

$$E^{(\mathbf{m}_0,\mathbf{r})} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \text{and} \quad E^{(\mathbf{m}_1,\mathbf{r})} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} .$$

In that example, the adversary takes advantage of the fact that $E^{(\mathbf{m}_0,\mathbf{r})} \neq E^{(\mathbf{m}_1,\mathbf{r})}$. We must exclude such "trivial" adversaries by restricting attention to adversaries that only query sources $\vec{\mathcal{M}}$ that do not leak information via equality-patterns. Formally, an mmr-source $\vec{\mathcal{M}}$ has equality-pattern disrespect $\zeta(\cdot)$ if there exists a family of reference equality-patterns $\{E^*_K\}_{k\in\mathbb{N}}$ such that

$$\Pr\left[\, E^{(\mathbf{m}_0,\mathbf{r})} \neq E^*_k \vee E^{(\mathbf{m}_1,\mathbf{r})} \neq E^*_k \;:\; (\mathbf{m}_0,\mathbf{m}_1,\mathbf{r}) \leftarrow_{\$} \vec{\mathcal{M}}(1^k) \,\right] \leq \zeta(k) \qquad (3.1)$$

for all $k \in \mathbb{N}$. Informally an adversary is equality-pattern respecting if it only queries sources $\vec{\mathcal{M}}$ with negligible equality-pattern disrespect. The above definition can be easily generalized to t-sources.

One might think this notion of equality-pattern disrespect is too restrictive. It mandates that each query uses some fixed equality pattern with high probability. Instead, one might suggest a more liberal notion that demands only that nothing about b is directly leaked, e.g. that $E^{(\mathbf{m}_0,\mathbf{r})} = E^{(\mathbf{m}_1,\mathbf{r})}$ holds with high probability for any mmr source queried. However, this security notion is not achievable: an attacker can choose $\vec{\mathcal{M}}$ so that the equality pattern encodes (say) all the bits that are common between the first messages of $\mathbf{m}_0$ and $\mathbf{m}_1$.

Note that, with probability closely related to their conditional min-entropy, block sources already output vectors whose equality pattern is the identity matrix. Thus for block sources we will omit equality-pattern restrictions entirely.

Notions. We can assume (without loss of generality) that a CDA adversary makes a single **RevealPK** query and then no further **LR** queries. We say A is a (μ, n, ρ)-adversary if all of its **LR** queries are (μ, n, ρ)-mmr-sources with negligible equality-pattern disrespect. We say that a PKE scheme $\mathcal{AE}$ with message length $n(\cdot)$ and randomness length $\rho(\cdot)$ is IND-CDA secure for (μ, n, ρ)-mmr-sources if for all PT (μ, n, ρ) adversaries A the function $\mathbf{Adv}^{\mathrm{cda}}_{\mathcal{AE},A}(\cdot)$ is negligible. Scheme $\mathcal{AE}$ is H-IND secure for (μ, n, ρ)-mmr-sources if it is IND-CPA secure and IND-CDA secure for (μ, n, ρ)-mmr-sources. We can extend these notions to mmr-block-sources by restricting to adversaries that query mmr-block-sources.

On adaptivity. We can consider non-adaptive IND-CDA security by restricting attention in the notions above to adversaries that only make a single **LR** query. Why not focus solely on this (simpler) security goal? The standard IND-CPA setting (implicitly) provides security against multiple, adaptive **LR** queries. In that setting a straightforward hybrid argument shows that security against multiple adaptive **LR** queries is implied by security against a single **LR** query [11, 9]. We wish to maintain the same standard of adaptive security in the IND-CDA setting. Unfortunately, in the IND-CDA setting, unlike the IND-CPA setting, adaptive security is not implied by non-adaptive security. In short this is because a CDA adversary necessarily cannot learn the public key before (or while) making **LR** queries. To see the separation, consider a PKE scheme that appends to every ciphertext the public key used. This will not affect the security of the scheme when an adversary can only make a single query. However, an adaptive CDA adversary can query an mmr-source, learn the public key, and craft a second source that uses the public key to help leak the challenge bit.

Given this, our primary goal is the stronger notion of adaptive security. That said, non-adaptive hedge security is also relevant because in practice adaptive adversaries might be rare.

Adaptive PRIV. A special case of our framework occurs when the PKE scheme $\mathcal{AE}$ being considered has randomness length $\rho(k) = 0$ for all k (meaning also that adversaries query mm-sources, instead of mmr-sources). In this case we are considering deterministic encryption, and the IND-CDA definition and notions give a strengthening (by way of adaptivity) of the PRIV security notion from [12, 16, 23]. (For non-adaptive adversaries the definitions are equivalent.) For clarity we will use PRIV to refer to this special case, and let $\mathbf{Adv}^{\mathrm{priv}}_{\mathcal{AE},A}(k) = \mathbf{Adv}^{\mathrm{cda}}_{\mathcal{AE},A}(k)$.

Resource usage. Recall that by our convention, the running time of a CDA adversary is the time for the execution of the adversary with game $\mathrm{CDA}_{\mathcal{AE},k}$. Thus, A being PT implies that the mmr-sources that comprise A's **LR** queries are also PT. This is a distinction from [23] which will be important in our results. Note that in practice we do not expect to see sources that are not PT, so our definition is not restrictive. Non-PT sources were needed in [23] for showing that single-message security implied (non-adaptive) multi-message security for deterministic encryption of block sources.

```
procedure Initialize:
$\pi_{\mathrm{h}} \leftarrow_{\$} \mathsf{P}_{\mathrm{h}}(1^k)$
$\kappa \leftarrow_{\$} \mathsf{K}_{\mathrm{h}}(pars)$
$b \leftarrow_{\$} \{0,1\}$
Ret $\pi_{\mathrm{h}}$

procedure RevealPK:
$\mathsf{pkout} \leftarrow \mathsf{true}$
Ret $\kappa$

procedure RoR($\vec{\mathcal{M}}$):
If $\mathsf{pkout} = \mathsf{true}$ then Ret $\perp$
$\mathbf{m} \leftarrow_{\$} \vec{\mathcal{M}}$
If $b = 1$ then $\mathbf{y} \leftarrow \mathsf{H}(\kappa, \mathbf{m})$
Else $\mathbf{y} \leftarrow_{\$} R(\pi_{\mathrm{h}})$
Ret $\mathbf{y}$

procedure Finalize($b'$):
Ret $(b = b')$
```

Figure 3.2: Game ALH (Adaptive Leftover Hash) associated to a family of hash functions $\mathcal{H}$ and a security parameter k.

3.3 Adaptive Variants of the LHL

Our schemes that hedge against bad randomness will make use of an adaptive version of the Leftover Hash Lemma. Informally, the Leftover Hash Lemma (LHL) [48] states that a collection $\mathcal{H}$ of universal hash functions is a strong extractor. A standard hybrid argument (c.f., [68, Lemma 6]) extends the leftover hash lemma to block sources. That is, for any joint distribution $\boldsymbol{X} = (X_1, \ldots, X_\ell)$ such that each X_i has sufficient min-entropy given $X_1, \ldots, X_{i-1}$, and for a randomly chosen key κ for $\mathcal{H}$, the distribution $(\kappa, \mathsf{H}(\kappa, X_1), \ldots, \mathsf{H}(\kappa, X_\ell))$ is statistically-close to the uniform distribution over the range of the hash function.

We give an adaptive variant of this lemma.[3] Specifically, in game $\mathrm{ALH}_{\mathcal{H}}$ (see Figure 3.2) we consider the following scenario: hash function parameters π_{h} and key κ are chosen at random, and an adversary can adaptively interact with an oracle that on input an m-source $\vec{\mathcal{M}}$ samples $(m_1, \ldots, m_v)$ and, depending on a challenge bit b, outputs either $(\mathsf{H}(\kappa, m_1), \ldots, \mathsf{H}(\kappa, m_v))$, or $(y_1, \ldots, y_v)$ that are each sampled uniformly at random from the range $R(\pi_{\mathrm{h}})$. Then, the adversary receives the key κ, and is no longer allowed to interact with the oracle and it must try to guess the bit b.

More formally, let $\mathcal{H} = (\mathsf{P}_{\mathrm{h}}, \mathsf{K}_{\mathrm{h}}, \mathsf{H})$ be a family of universal hash functions with input length $n(\cdot)$. For every k and all $\pi_{\mathrm{h}} \in [\mathsf{P}_{\mathrm{h}}[1^k)]$ we let $R(\pi_{\mathrm{h}}) = \{\, \mathsf{H}(\kappa, x)$:

[3]We note that hashing of block sources in an adaptive setting was also considered by Lu [51] in the context of the bounded storage model.

$\kappa \in [\mathsf{K}_\mathrm{h}(\pi_\mathrm{h})]$ and $x \in \{0,1\}^n$ }. We associate to $\mathcal{H}$ game $\mathrm{ALH}_\mathcal{H}$ of Figure 3.2. An ALH adversary may make multiple **RoR** queries, each being a vector m-source over $\{0,1\}^{n(k)}$. The setting is adaptive because each query can depend on replies to previous ones. The adversary makes a single **RevealPK** query and after that makes no further **RoR** queries. In Lemma 3.3.1 we bound the advantage of any adversary in this game, formally defined as

$$\mathbf{Adv}^{\mathrm{alh}}_{\mathcal{H},A}(k) = 2 \cdot \Pr\left[\, \mathrm{ALH}^{A}_{\mathcal{H},k} \Rightarrow \mathsf{true} \,\right] - 1 \; .$$

Lemma 3.3.1 Let $\mathcal{H} = (\mathsf{P}_\mathrm{h}, \mathsf{K}_\mathrm{h}, \mathsf{H})$ be a family of universal hash functions with input length $n(\cdot)$ and 2^t-bounded range. We associate to it an ALH adversary making q **RoR** queries, each being an ℓ-vector m-source with conditional min-entropy of at least s. Then for all k

$$\mathbf{Adv}^{\mathrm{alh}}_{\mathcal{H},A}(k) \leq q(k) \cdot \ell(k) \cdot \sqrt{2^{t(k)-s(k)}} \; . \; \square$$

For $q = 1$ and $\ell = 1$, the lemma is a standard variation of the original LHL. The lemma is also well-known for the case $q = 1$ and $\ell > 1$, as this is just a version of the LHL for block sources [68] that can be shown with a hybrid argument. Another hybrid argument extends the lemma to $q > 1$. We briefly describe the argument. Given an adversary A that makes q **RoR** queries, we build an adversary B that makes only one **RoR** query. Adversary B guesses a query $j \in \{1, \ldots, q\}$ and for the first $j-1$ queries from A answers with all uniformly random range points. For the jth query, B forwards A's query to its own **RoR** oracle and returns the result to A. At this point, B learns the hash key through its **RevealPK** oracle, and answers the rest of A's queries using the hash key and applying the hash function to the sampled values. When A asks a **RevealPK** query, B gives it the key it learned earlier. Finally, B guesses the same bit that A guesses. It is easy to see this hybrid argument results in a factor q loss, proving the lemma.

3.4 Hedged PKE Schemes

In this section we present two constructions of hedged public-key encryption schemes and prove their security.

3.4.1 The Schemes

We now present two constructions of hedge-secure PKE schemes. The first scheme composes deterministic and randomized encryption, while the second scheme is built directly from a deterministic encryption scheme.

For the following, let $\mathcal{AE}_{\rm r} = (\mathcal{P}_{\rm r}, \mathcal{K}_{\rm r}, \mathcal{E}_{\rm r}, \mathcal{D}_{\rm r})$ be a (randomized) PKE scheme with message length $n_{\rm r}(\cdot)$ and randomness length $\rho(\cdot)$. Let $\mathcal{AE}_{\rm d} = (\mathcal{P}_{\rm d}, \mathcal{K}_{\rm d}, \mathcal{E}_{\rm d}, \mathcal{D}_{\rm d})$ be a (deterministic) PKE scheme with message length $n_{\rm d}(\cdot)$ and randomness length always 0. Associate to $\mathcal{AE}_c$ for $c \in \{{\rm d}, {\rm r}\}$ the function $\mathsf{maxclen}_c(k)$ mapping any k to the maximum length (over all possible public keys, messages, and if applicable, randomness) of a ciphertext output by $\mathcal{E}_c$.

Randomized-then-deterministic. For our first scheme we compose a randomized encryption scheme with a deterministic one. The order of the composition turns out to be important. We apply randomized encryption first, and then deterministic encryption. Define $\mathsf{RtD}[\mathcal{AE}_{\rm r}, \mathcal{AE}_{\rm d}] = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ with randomness length ρ and message length $n_{\rm r}$ to work as follows. The parameter generation algorithm $\mathcal{P}$ runs $par_{\rm r} \leftarrow\!\!\$\ \mathcal{P}_{\rm r}(1^k)$ and $par_{\rm d} \leftarrow\!\!\$\ \mathcal{P}_{\rm d}(1^k)$ and outputs $pars = (par_{\rm r}, par_{\rm d})$. Key generation $\mathcal{K}$ just runs $(pk_{\rm r}, sk_{\rm r}) \leftarrow\!\!\$\ \mathcal{K}_{\rm r}(par_{\rm r})$ and $(pk_{\rm d}, sk_{\rm d}) \leftarrow\!\!\$\ \mathcal{K}_{\rm d}(par_{\rm d})$ and outputs $pk = (pk_{\rm r}, pk_{\rm d})$ and $sk = (sk_{\rm r}, sk_{\rm d})$. Encryption is defined by

$$\mathcal{E}((pk_{\rm r}, pk_{\rm d}), m\ ;\ r) = \mathcal{E}_{\rm d}(pk_{\rm d}, c \parallel 10^{\ell})\ .$$

where $c = \mathcal{E}_{\rm r}(pk_{\rm r}, m; r)$ and $\ell = n_{\rm d} - |c| - 1$. Here we need that $n_{\rm d}(k) > \mathsf{maxclen}_{\rm r}(k)$ for all k. The decryption algorithm $\mathcal{D}$ works in the natural way. As we will see in subsequent sections, this construction will be secure when the randomized encryption scheme is such that for all k, all $par_{\rm r} \in [\mathcal{P}_{\rm r}(1^k)]$, and all $(pk_{\rm r}, sk_{\rm r}) \in [\mathcal{K}_{\rm r}(par_{\rm r})]$, encryption $\mathcal{E}_{\rm r}(pk_{\rm r}, \cdot)$ is injective in (m, r). Many encryption schemes have this property; El Gamal [38] is one example.

We note that if we instead tried the other order of composition, we would have no guarantee that the scheme is secure. Certainly applying a deterministic scheme first and then a randomized scheme would still result in a secure randomized encryption scheme. However, the resulting scheme would not necessarily be IND-CDA secure. The problem is that the min-entropy could be, say, evenly divided between the message and randomness. In that case, we would end up applying a secure deterministic encryption scheme to a message with some entropy and then a randomized encryption scheme to the deterministic ciphertext using coins that again have some entropy. Contrast this with the other way of composing: in that case the deterministic scheme can rely on *all* of the entropy in the message-coins pair, since it's the outer scheme and the inner randomized scheme is injective in the message and coins.

Pad-then-Deterministic. Our next construction dispenses entirely with the need for a dedicated randomized encryption scheme, instead using simple padding to directly construct a (randomized) encryption scheme from a deterministic one.

Scheme $\mathsf{PtD}[\mathcal{AE}_{\rm d}] = (\mathcal{P}_{\rm d}, \mathcal{K}_{\rm d}, \mathcal{E}, \mathcal{D})$ with randomness length ρ and message length n works as follows. Parameter and key generation are inherited from the underlying (deterministic) encryption scheme. Encryption is defined by

$$\mathcal{E}(pk_{\rm d}, m\,;\, r) = \mathcal{E}_{\rm d}(pk_{\rm d}, r \,\|\, m)$$

where we require that $n_{\rm d}(k) > \rho(k) + n(k)$. Decryption proceeds by applying $\mathcal{D}_{\rm d}$, to retrieve $r \,\|\, m$, and then returning m.

Intuitively, the scheme will get both its IND-CDA security and its IND-CPA security from the security of the deterministic scheme. As we will see in the next section, IND-CPA turns out to be the challenging part to prove.

3.4.2 Security

In this section we prove the IND-CPA and IND-CDA security of the schemes presented above. In the IND-CDA case, the following theorems will show that the adaptive or non-adaptive IND-CDA security depends on the corresponding adaptive or non-adaptive PRIV security of the deterministic PKE scheme. While previous work [23] only achieves non-adaptive PRIV security, we show in the next section how to achieve adaptive PRIV security.

Randomized-then-deterministic. Intuitively, the hedged security of the RtD construction is inherited from the IND-CPA security of the underlying randomized scheme $\mathcal{AE}_{\mathrm{r}}$ and the PRIV security of the underlying deterministic scheme $\mathcal{AE}_{\mathrm{d}}$. As alluded to before, we have one technical requirement on $\mathcal{AE}_{\mathrm{r}}$ for the IND-CDA proof to work. We say $\mathcal{AE}_{\mathrm{r}} = (\mathcal{P}_{\mathrm{r}}, \mathcal{K}_{\mathrm{r}}, \mathcal{E}_{\mathrm{r}}, \mathcal{D}_{\mathrm{r}})$ with message length $n_{\mathrm{r}}(\cdot)$ and randomness length $\rho(\cdot)$ is injective if for all k, all $par_{\mathrm{r}} \in [\mathcal{P}_{\mathrm{r}}(1^k)]$, and all $(pk_{\mathrm{r}}, sk_{\mathrm{r}}) \in [\mathcal{K}_{\mathrm{r}}(par_{\mathrm{r}})]$, the map $\mathcal{E}_{\mathrm{r}}(pk_{\mathrm{r}}, \cdot\,;\,\cdot)$ is injective. We have the following theorem.

Theorem 3.4.1 [RtD **is hedge secure**] Let $\mathcal{AE}_{\mathrm{r}} = (\mathcal{P}_{\mathrm{r}}, \mathcal{K}_{\mathrm{r}}, \mathcal{E}_{\mathrm{r}}, \mathcal{D}_{\mathrm{r}})$ be an injective PKE scheme with message length $n_{\mathrm{r}}(\cdot)$ and randomness length $\rho(\cdot)$. Let $\mathcal{AE}_{\mathrm{d}} = (\mathcal{P}_{\mathrm{d}}, \mathcal{K}_{\mathrm{d}}, \mathcal{E}_{\mathrm{d}}, \mathcal{D}_{\mathrm{d}})$ be a (deterministic) encryption scheme with message length $n_{\mathrm{d}}(\cdot)$ so that $n_{\mathrm{d}}(\cdot) \geq \mathsf{maxclen}_{\mathrm{r}}(\cdot)$. Let $\mathcal{AE} = \mathsf{RtD}[\mathcal{AE}_{\mathrm{r}}, \mathcal{AE}_{\mathrm{d}}] = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be the PKE scheme defined in Section 3.4.1.

- (IND-CPA) Let A be an IND-CPA adversary. Then there exists an IND-CPA adversary B such that for any k

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{AE},A}(k) \leq \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{AE}_{\mathrm{r}},B}(k)$$

  where B runs in time that of A plus the time to run $\mathcal{E}_{\mathrm{d}}$ once.

- (IND-CDA) Let A be a CDA adversary that makes at most q LR queries, each a $v(\cdot)$-vector $(\mu, n_{\mathrm{r}}, \rho)$-$\mathsf{mmr}$-source (resp. block-source). Then there exists a PRIV adversary B such that for any k

$$\mathbf{Adv}^{\text{cda}}_{\mathcal{AE},A}(k) \leq \mathbf{Adv}^{\text{priv}}_{\mathcal{AE}_{\mathrm{d}},B}(k)$$

  where B runs in time that of A plus the time to run $q(k) \cdot v(k)$ executions of $\mathcal{E}_{\mathrm{r}}$ and makes at most q LR queries, each consisting of a $v(\cdot)$-vector (μ, n_{d})-mm-

source (resp. block-source). $\square$

Note that the second part of the theorem states the result for either sources or just block-sources. Before proving in more detail, we give a sketch. The first part of the theorem is immediate from the IND-CPA security of $\mathcal{AE}_{\mathrm{r}}$. For the second part, any mmr-source $\vec{\mathcal{M}}$ queried by A is converted into an mm-source $\vec{\mathcal{M}}'$ to be queried by B. This is done by having $\vec{\mathcal{M}}'$ run $\vec{\mathcal{M}}$ to get $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r})$ and then outputting the pair of vectors $(\mathcal{E}_{\mathrm{r}}(pk, \mathbf{m}_0 \,;\, \mathbf{r}), \mathcal{E}_{\mathrm{r}}(pk, \mathbf{m}_1 \,;\, \mathbf{r}))$ with appropriate padding. (The ciphertexts are the "messages" for $\mathcal{E}_{\mathrm{d}}$.) Because $\mathcal{AE}_{\mathrm{r}}$ is injective, it preserves the min-entropy of the message/coins pair, and thus $\vec{\mathcal{M}}'$ is a source of the appropriate type.

Proof of Thm. 3.4.1: We first show IND-CPA security. Let A be an INDCPA adversary against $\mathcal{AE} = \mathsf{RtD}[\mathcal{AE}_{\mathrm{r}}, \mathcal{AE}_{\mathrm{d}}] = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where $\mathcal{AE}_{\mathrm{r}} = (\mathcal{P}_{\mathrm{r}}, \mathcal{K}_{\mathrm{r}}, \mathcal{E}_{\mathrm{r}}, \mathcal{D}_{\mathrm{r}})$ is a randomized PKE scheme and $\mathcal{AE}_{\mathrm{d}} = (\mathcal{P}_{\mathrm{d}}, \mathcal{K}_{\mathrm{d}}, \mathcal{E}_{\mathrm{d}}, \mathcal{D}_{\mathrm{d}})$ is a deterministic PKE scheme with plaintext length n_{d}. We build an INDCPA adversary B against $\mathcal{AE}_{\mathrm{r}}$ using A; the adversary is shown in Figure 3.3. Adversary B, on input pk_{r}, runs $\mathcal{P}_{\mathrm{d}}$ and $\mathcal{K}_{\mathrm{d}}$ to generate a keypair $(pk_{\mathrm{d}}, sk_{\mathrm{d}})$ for the deterministic PKE scheme. It then runs adversary A with public key $(pk_{\mathrm{r}}, pk_{\mathrm{d}})$. When A queries **LR** with a pair of messages (m_0, m_1), B forwards the query to its own **LR** oracle. When B receives ciphertext c, it returns to adversary A the encryption $\mathcal{E}_{\mathrm{d}}(pk_{\mathrm{d}}, c \,\|\, 10^{\ell})$, where $\ell = n_{\mathrm{d}} - |c| - 1$ is the amount of padding necessary to make c fit in the plaintext space of $\mathcal{AE}_{\mathrm{d}}$. When A outputs a guess bit b', B also outputs this same guess. It is easy to see that the simulation is perfect and the advantages are equal.

We next show IND-CDA security. Let A be a CDA adversary making **LR** queries that are v-vector $(\mu, n_{\mathrm{r}}, \rho)$-sources (resp. block-sources) and attacking $\mathcal{AE}$ constructed as in the theorem statement from $\mathcal{AE}_{\mathrm{r}}$ and $\mathcal{AE}_{\mathrm{d}}$. We will build PRIV adversary B (right side of Figure 3.3) against $\mathcal{AE}_{\mathrm{d}}$ as follows. Adversary B, at the start of the game, runs $\mathcal{P}_{\mathrm{r}}$ and $\mathcal{K}_{\mathrm{r}}$ to generate keys $(pk_{\mathrm{r}}, sk_{\mathrm{r}})$ for the randomized encryption scheme. It then runs adversary A and answers queries as follows. On query **RevealPK**, B queries its own **RevealPK** adversary to learn pk_{d} and then returns $(pk_{\mathrm{r}}, pk_{\mathrm{d}})$ to A. On query $\mathbf{LR}(\vec{\mathcal{M}})$ for mmr-source (resp. block-source) $\vec{\mathcal{M}}$, B constructs mm-source (resp. block-source) $\vec{\mathcal{M}}^*$ that runs $\vec{\mathcal{M}}$ to get vector

```
Adversary $B(1^k, pk_r)$
$pars \leftarrow_\$ \mathcal{P}_d(1^k)$
$(pk_d, sk_d) \leftarrow_\$ \mathcal{K}_d(pars)$
$pk \leftarrow (pk_r, pk_d)$
Run $A(1^k, pk)$.
  On query $\mathbf{LR}(m_0, m_1)$:
  $c \leftarrow \mathbf{LR}_B(m_0, m_1)$
  $\ell \leftarrow n_d - |c| - 1$
  $c' \leftarrow \mathcal{E}_d(pk_d, c \,\|\, 10^\ell)$
  Ret. $c'$
When $A$ halts with output $b'$, halt and output $b'$.
```

```
Adversary $B(1^k)$
$pars \leftarrow_\$ \mathcal{P}_r(1^k)$
$(pk_r, sk_r) \leftarrow_\$ \mathcal{K}_r(pars)$
Run $A(1^k)$.
  On query $\mathbf{RevealPK}()$:
  $pk_d \leftarrow \mathbf{RevealPK}_B()$
  $pk \leftarrow (pk_r, pk_d)$
  Ret. $pk$
  On query $\mathbf{LR}(\vec{\mathcal{M}})$:
  $\mathbf{c} \leftarrow \mathbf{LR}_B(\vec{\mathcal{M}}^*(\vec{\mathcal{M}}))$
  Ret. $\mathbf{c}$
  $\vec{\mathcal{M}}^*(\vec{\mathcal{M}})$:
  $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r}) \leftarrow_\$ \vec{\mathcal{M}}$
  For $i$ in 1 to $|\mathbf{m}_0|$ do:
    $\ell_0 \leftarrow n_d - |\mathcal{E}_r(pk_r, \mathbf{m}_0[i]\,;\,\mathbf{r}[i])| - 1$
    $\mathbf{x}_0[i] \leftarrow \mathcal{E}_r(pk_r, \mathbf{m}_0[i]\,;\,\mathbf{r}[i]) \,\|\, 10^{\ell_0}$
    $\ell_1 \leftarrow n_d - |\mathcal{E}_r(pk_r, \mathbf{m}_1[i]\,;\,\mathbf{r}[i])| - 1$
    $\mathbf{x}_1[i] \leftarrow \mathcal{E}_r(pk_r, \mathbf{m}_1[i]\,;\,\mathbf{r}[i]) \,\|\, 10^{\ell_1}$
  Ret. $(\mathbf{x}_0, \mathbf{x}_1)$
When $A$ halts with output $b'$, halt and output $b'$.
```

Figure 3.3: Adversaries for the proof of Theorem 3.4.1.

$(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r})$ and then outputs the pair

$$(\mathcal{E}_r(pk_r, \mathbf{m}_0\,;\,\mathbf{r}), \mathcal{E}_r(pk_r, \mathbf{m}_1\,;\,\mathbf{r}))\ ,$$

where each component in each vector in the pair is padded out with a single 1 followed by the appropriate number of 0s to make it length n_d. The details of $\vec{\mathcal{M}}^*$ are shown in Figure 3.3. Since $\mathcal{AE}_r$ is injective in the message and coins, it preserves the min-entropy of the message/coins pair, and it follows that $\vec{\mathcal{M}}^*$ is a source (resp. block-source) with the same min-entropy as $\vec{\mathcal{M}}$.

Finally, when A halts with guess bit b', B outputs the same guess. Again, it is easy to see the simulation is perfect. ▌

Pad-then-deterministic. The security of the PtD scheme is more difficult to establish. The IND-CDA security is inherited immediately from the PRIV security of the $\mathcal{AE}_{\mathrm{d}}$ scheme. Here the challenge is, in fact, proving IND-CPA security. For this we will need a stronger assumption on the underlying deterministic encryption scheme — that it is a universal LTDF.

Theorem 3.4.2 [PtD **is hedge secure**] Let $\mathcal{AE}_{\mathrm{d}} = (\mathcal{P}_{\mathrm{d}}, \mathcal{K}_{\mathrm{d}}, \mathcal{E}_{\mathrm{d}}, \mathcal{D}_{\mathrm{d}})$ be a deterministic encryption scheme with message length $n_{\mathrm{d}}(\cdot)$. Let $\mathcal{AE} = \mathsf{PtD}[\mathcal{AE}_{\mathrm{d}}] = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be the PKE scheme defined in Section 3.4.1 with message length $n(\cdot)$ and randomness length $\rho(\cdot)$ such that $n(k) = n_{\mathrm{d}}(k) - \rho(k)$ for all k.

- (IND-CPA) Let $\mathcal{K}_\ell$ be a universal-inducing (n_{d}, ℓ)-lossy key generation algorithm for $\mathcal{AE}_{\mathrm{d}}$. Let A be an INDCPA adversary. Then there exists a KEYIND adversary B such that for all k

  $$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{AE},A}(k) \le 2 \cdot \mathbf{Adv}^{\text{key-ind}}_{\mathcal{AE}_{\mathrm{d}},\mathcal{K}_\ell,B}(k) + \sqrt{2^{3n(k)-\ell(k)+2}}\ .$$

  B runs in time that of A.

- (IND-CDA) Let A be a CDA adversary that makes at most q LR queries, each a $v(\cdot)$-vector (μ, n, ρ)-mmr-source (resp. block-source). Then there exists a PRIV adversary B such that for all k

  $$\mathbf{Adv}^{\text{cda}}_{\mathcal{AE},A}(k) \le \mathbf{Adv}^{\text{priv}}_{\mathcal{AE}_{\mathrm{d}},B}(k)$$

  where B runs in time that of A and makes at most q LR queries, each a $v(\cdot)$-vector (μ, n_{d})-mm-source (resp. block-source). $\square$

The CDA portion of the theorem follows immediately from the PRIV security of the deterministic scheme. Once this is established, one might think that concluding IND-CPA can be based just on PtD being IND-CDA secure, since the padded randomness provides high min-entropy. A proof along these lines might proceed as follows: a CDA adversary C runs an INDCPA adversary A. When A makes an **LR** query (m_0, m_1), C creates a source that chooses coins randomly and prepends them to either m_0 or m_1 depending on the bit. This source will have high min-entropy because of the coins being chosen randomly.

However, this approach does not work! The reason is that an INDCPA adversary expects knowledge of the public-key *before* making any **LR** queries, while a CDA adversary only learns the public-key *after* making its **LR** queries. This issue is discussed in more detail in [16]. Thus, we use a different approach to prove this part of Theorem 3.4.2.

We use universal LTDFs for our deterministic PKE scheme. Then in the proof, once we switch our encryption scheme to the lossy mode, we construct an ALH adversary (recall game ALH in Section 3.3) that makes 2^n queries to its **RoR** oracle. Each of these queries corresponds to one of the 2^n messages in the message space of our encryption scheme. As long as we use enough random padding, the randomness will still give enough min-entropy in each query so that we can apply our adaptive LHL[4] and argue security. We now give details.

Proof of Thm. 3.4.2: We first briefly prove IND-CDA. Let A be a CDA adversary against $\mathcal{AE}$. We can easily construct a PRIV adversary B against $\mathcal{AE}_\mathrm{d}$. B runs A and on **LR** query $\vec{\mathcal{M}}$, a v-vector $(\mu, n_\mathrm{r}, \rho)$-$\mathsf{mmr}$ source (resp. block-source) queries $\vec{\mathcal{M}}'$ that samples from $\vec{\mathcal{M}}$ to get $(\mathbf{m}_0, \mathbf{m}_1, \mathbf{r})$ and outputs $((\mathbf{r} \| \mathbf{m}_0), (\mathbf{r} \| \mathbf{m}_1))$, where $\mathbf{r} \parallel \mathbf{m}_b$ denotes the sequence $((\mathbf{r}[1] \parallel \mathbf{m}_b[1]), \ldots (\mathbf{r}[v] \parallel \mathbf{m}_b[v]))$. This clearly results in a v-vector (μ, n_d) mm-source (resp. block-source), where $n_\mathrm{d} = n_\mathrm{r} + \rho$. The simulation is perfect and security follows.

Next we show IND-CPA. Let A be an INDCPA adversary against PtD. We will go through a series of game transitions to prove the theorem. Game G_0 is simply the INDCPA game, so by definition

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{PtD},A}(k) = 2 \cdot \Pr\left[\, G_0^A \,\right] - 1 \ .$$

[4]Actually, the LHL for block-sources already suffices.

```
Adversary $B(1^k, pk)$
$b \leftarrow_\$ \{0,1\}$
Run $A(1^k, pk)$.
    On query $\mathbf{LR}(m_0, m_1)$:
    $c \leftarrow_\$ \mathcal{E}(pk, m_b)$
    Ret. $c$
When $A$ halts with output $b'$, halt and
output $(b = b')$.
```

```
Adversary $C(1^k, pars)$
$b \leftarrow_\$ \{0,1\}$
For $m$ in $0^n$ to $1^n$:
    $\mathbf{c}[m] \leftarrow \mathbf{RoR}(\vec{\mathcal{M}}_m)$
$pk' \leftarrow_\$ \mathbf{RevealPK}()$
Run $A(1^k, pk')$.
    On query $\mathbf{LR}(m_0, m_1)$:
    $c \leftarrow \mathbf{c}[m_b]$
    Ret. $c$
    $\vec{\mathcal{M}}_m$:
    $r \leftarrow_\$ \{0,1\}^{n_\mathrm{d}-n}$
    Ret. $r \,\|\, m$
When $A$ halts with output $b'$, halt and
output $(b = b')$.
```

Figure 3.4: Adversaries for the proof of Theorem 3.4.2

Game G_1 is identical to G_0 except that **Initialize** uses the lossy key generation algorithm $\mathcal{K}_\ell$. We will define a KEYIND adversary B such that

$$\Pr\left[\, G_0^A \Rightarrow \mathsf{true} \,\right] - \Pr\left[\, G_1^A \Rightarrow \mathsf{true} \,\right] \le \mathbf{Adv}^{\text{key-ind}}_{\mathcal{AE},\mathcal{K}_\ell,B}(k) \ .$$

Adversary B, shown in Figure 3.4, when given a public key pk that is either from $\mathcal{K}_\mathrm{d}$ or $\mathcal{K}_\ell$, simply runs adversary A as in games G_0 and G_1 with pk; if there is a gap between A's success probability in games G_0 and G_1, then B will be able to distinguish whether the key is lossy or not.

Game G_2 is the same as G_1 except that **LR** returns a uniform element from the range of the hash function (instead of returning the encryption of m_b). We claim that there is an unbounded ALH adversary C such that

$$\Pr\left[\, G_1^A \Rightarrow \mathsf{true} \,\right] - \Pr\left[\, G_2^A \Rightarrow \mathsf{true} \,\right] \le \mathbf{Adv}^{\text{alh}}_{\mathcal{H},C}(k) \ ,$$

where $\mathcal{H} = (\mathcal{P}, \mathcal{K}_\ell, \mathcal{E}_\mathrm{d})$ is the universal family of hash functions induced by $\mathcal{AE}$ and $\mathcal{K}_\ell$. The ALH adversary C, shown in Figure 3.4, proceeds as follows. First, C

makes $q = 2^n$ queries to its **RoR** oracle, where $\{0,1\}^n$ is the message space of PtD. The ith **RoR** query is an m-source $\vec{\mathcal{M}}_{m^i}$ of vector length 1 that samples a uniform string of $n_{\mathrm{d}} - n$ bits and concatenates it with m^i, the ith message in the plaintext space $\{0,1\}^n$ according to some known ordering on $\{0,1\}^n$ (i.e., lexicographical order). It is easy to see that due to the padded uniform bits, each m-source has min-entropy of $n_{\mathrm{d}} - n$ bits, even conditioned on all the previous queries. Let the answers C receives to its q **RoR** queries be called $y_1, \ldots, y_q$. Next, C calls oracle **RevealPK** and learns pk'. At this point, C runs adversary A as in games G_1 and G_2, flipping a bit b and giving A the public key pk'. On oracle query $\mathbf{LR}(m_0, m_1)$ from A, C finds the j such that m_b equals m^j, the jth message in the plaintext space according to the known ordering. Adversary C answers the **LR** query with y_j. When A finishes with output b', C outputs 1 if $b = b'$ and 0 otherwise.

Finally, we claim that $\Pr\left[\, G_2^A \Rightarrow \mathsf{true} \,\right] = 1/2$. This is true since the answer to the **LR** query no longer depends on the bit b but is instead simply a uniform range point. Combining the above equations we get

$$\begin{aligned}
\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{PtD},A}(k) &\leq 2 \cdot \mathbf{Adv}^{\text{key-ind}}_{\mathcal{AE}_{\mathrm{d}},\mathcal{K}_\ell,B}(k) + 2 \cdot \mathbf{Adv}^{\text{alh}}_{\mathcal{H},C}(k) \\
&\leq 2 \cdot \mathbf{Adv}^{\text{key-ind}}_{\mathcal{AE}_{\mathrm{d}},\mathcal{K}_\ell,B}(k) + 2 \cdot 2^{n(k)} \cdot \sqrt{2^{n_{\mathrm{d}}(k) - \ell(k) - (n_{\mathrm{d}}(k) - n(k))}} \\
&\leq 2 \cdot \mathbf{Adv}^{\text{key-ind}}_{\mathcal{AE}_{\mathrm{d}},\mathcal{K}_\ell,B}(k) + \sqrt{2^{3n(k) - \ell(k) + 2}}\ ,
\end{aligned}$$

proving the theorem. ▌

It is instructive to see what happens when we instantiate PtD with known constructions for deterministic encryption. Let the deterministic PKE scheme be the DDH-based deterministic scheme from [23] (based on the LTDF in [58]). This deterministic PKE scheme has a corresponding universal-inducing $(n_{\mathrm{d}}, n_{\mathrm{d}} - \log p)$-lossy key generation algorithm $\mathcal{K}_\ell$ for a large prime p. Thus, the induced universal function family $\mathcal{H} = (\mathcal{P}_{\mathrm{d}}, \mathcal{K}_\ell, \mathcal{E}_{\mathrm{d}})$ has range size at most p. If we target non-adaptive security, then applying the bounds from [23, Th. 5.1], we know that for all k

$$\mathbf{Adv}^{\text{priv}}_{\mathcal{AE}_{\mathrm{d}},B}(k) \leq 2 \cdot \mathbf{Adv}^{\text{key-ind}}_{\mathcal{AE}_{\mathrm{d}},\mathcal{K}_\ell,C}(k) + 2 \cdot v(k) \cdot \sqrt{2^{\log p - \mu(k)}}\ .$$

So, if we want to use a 160-bit prime, believe $v(k)$ (the number of ciphertexts seen by an adversary) will be at most 2^{30}, and want the square-root term above to be, conservatively, at most 2^{-80}, then we need approximately 382 bits of min-entropy in our message-randomness pairs. Less min-entropy will simply increase that term, decreasing our confidence in security. Now let us compute how much random padding we need to get a desirable amount of IND-CPA security. Again basing the scheme on a 160-bit prime-order group, if we want to encrypt a 500-bit message and we want 80 bits of security, we would need approximately 1322 bits of random padding.

Finally, we point out that our hedge schemes inherit their IND-CDA security directly from the PRIV security of the deterministic scheme used. This means that if the deterministic scheme is adaptively PRIV secure for sources, then the resulting hedge scheme is adaptively IND-CDA secure for sources. However, we should point out that currently there are no known constructions of such strong deterministic encryption schemes. In particular, the best currently-known schemes are due to [23] and are non-adaptively PRIV secure for block-sources.

Though we do not yet know how to achieve security for sources in the standard model, we show in the next section how to extend the results of [23] to at least achieve adaptive security. Achieving security for sources (as opposed to block-sources) is a major open problem in deterministic PKE.

3.5 Achieving Adaptive PRIV Security

As we explained in Section 3.2, not all PRIV secure deterministic PKE schemes are adaptively secure. One way to see this is to consider a PKE scheme where the ciphertext has the public key appended to it. Then one query to the **LR** oracle will give a CDA adversary the public key. It can then query sources to **LR** that are specifically tailored to leak information based on that public key.

Luckily, we can show that universal LTDFs are adaptively secure. Intuitively, the reason is that ciphertexts returned by the **LR** oracle are computationally uniform, so the adversary does not learn any information that will help it craft bad sources for future **LR** queries. We prove the following theorem using our adaptive LHL from Section 3.3.

Theorem 3.5.1 [u-LTDFs are adaptive PRIV secure for block-sources] Let $\mathcal{AE}_{\mathrm{d}} = (\mathcal{P}_{\mathrm{d}}, \mathcal{K}_{\mathrm{d}}, \mathcal{E}_{\mathrm{d}}, \mathcal{D}_{\mathrm{d}})$ be a deterministic encryption scheme with message length $n_{\mathrm{d}}(\cdot)$. Let $\mathcal{K}_\ell$ be a universal-inducing (n_{d}, ℓ)-lossy key generation algorithm for $\mathcal{AE}_{\mathrm{d}}$. Let A be a PRIV adversary that makes at most q LR queries, each a $v(\cdot)$-vector (μ, n_{d})-mm-block-source. Then there exists a KEYIND adversary B such that for all k

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathcal{AE}_{\mathrm{d}},A}(k) \leq 2 \cdot \mathbf{Adv}^{\mathrm{key\text{-}ind}}_{\mathcal{AE}_{\mathrm{d}},\mathcal{K}_\ell,B}(k) + 2 \cdot q(k) \cdot v(k) \cdot \sqrt{2^{(n_{\mathrm{d}}-\ell)-\mu(k)}}\,.$$

B runs in time that of A.

Proof: Let game G_0 be the PRIV security game. We first move to a game G_1 where a lossy key is used instead of a real key, i.e., $\mathcal{K}_\ell$ is used instead of $\mathcal{K}_{\mathrm{d}}$ to generate the public and secret keys. If A can tell the difference between these games, then there is a KEYIND adversary B with high advantage that simply runs A and uses its challenge public key in the PRIV game. Next, we move to a game G_2 where **LR** always returns uniform points in the appropriate $2^{n_{\mathrm{d}}-\ell}$-bounded range of the universal hash function $\mathcal{H} = (\mathcal{P}_{\mathrm{d}}, \mathcal{K}_\ell, \mathcal{E}_{\mathrm{d}})$ induced by $\mathcal{AE}_{\mathrm{d}}$ and $\mathcal{K}_\ell$. If A can distinguish these games, then we can build an ALH adversary C with high advantage. Adversary C will flip a bit b and then on **LR** query a v-vector mm-block source $\vec{\mathcal{M}}$ from A, it constructs a v-vector m-block source to query to its **RoR** oracle. The m-block source runs $\vec{\mathcal{M}}$ to get $(\mathbf{m}_0, \mathbf{m}_1)$ and simply outputs $\mathbf{m}_b$. Since the hash family $\mathcal{H}$ has $2^{n_{\mathrm{d}}-\ell}$-bounded range, the adaptive LHL (Lemma 3.3.1) applies to give the bound in the theorem statement. Finally, game G_2 gives adversary A no advantage at guessing the bit since random range points that do not depend on the challenge bit are returned. ▌

Is Adaptivity Important? An obvious question is whether or not adaptivity matters. While we are unable to describe practical motivation for adaptive IND-CDA security, we believe it is important to target the strongest achievable notions of security. Further, IND-CDA only providing security for public-key independent message distributions is a significant drawback, so achieving adaptivity at least gives security for ciphertext-dependent message distributions.

3.6 Conclusion and Additional Information

In this chapter we formalized public-key encryption security in a setting where the coins used for encryption are no longer necessarily uniformly chosen. We described a new security notion for PKE, called IND-CDA, and argued that we want our PKE schemes to be both IND-CPA secure and IND-CDA secure. We then presented two schemes and proved they had these properties in the standard model.

Credits. An earlier version [13] of some of the material in this chapter appeared as part of work appearing in the Proceedings of ASIACRYPT 2009, copyright IACR, and co-authored with Mihir Bellare, Zvika Brakerski, Moni Naor, Thomas Ristenpart, Gil Segev, and Hovav Shacham. I was a primary researcher for this work.

Chapter 4

Repeated Randomness

In this chapter we turn our attention to a second type of randomness failure: repeated randomness. Specifically, we will study public-key encryption when the encryption algorithm is given coins that it may have received in the past. Since security models for public-key encyption implicitly assume all parties have access to *fresh* coins for each encryption operation, schemes proven secure under these models might be completely insecure when randomness is repeated.

4.1 Overview

To motivate the study of repeated randomness failures we will focus on virtual machines. In short, a virtual machine (VM) is software that emulates a real machine. A VM consists of a virtual machine monitor (or hypervisor) which emulates multiple virtual computers that can have varying instruction sets and run different operating systems. The VM monitor will then share the physical machine's resources among the virtual machines, translating machine instructions and acting as a simulator for the underlying operating systems.

Servers running on virtual machines are helping fuel the cloud computing boom; popular services like Amazon's Elastic Compute Cloud (EC2) [1] rely heavily on virtualization. In this service, a user buys some compute time and receives access to a virtual machine on one of Amazon's servers. Within that VM, the user can run a fully functional OS and, in particular, run a web server for his or her business. Virtual machines have also become popular on desktop machines. Products like VMWare [5] and VirtualBox [4] allow average users to experiment with virtualization on their home PCs.

Virtual machines can lead to a more efficient use of physical hardware, but also have security benefits. Because VMs provide a type of sandbox, they have been used to test potentially malicious code [30] and isolate web browsers from the rest of the system to mitigate the effects of browser vulnerabilities [31]. VMs have also been used to more easily create large honeypots [59]. Despite this, the focus of this chapter is on how VMs can be *detrimental* to security. The reason we focus on, which also happens to be one of the most useful features of VMs, is their ability to take state snapshots.

State Snapshots. Virtual machines allow a user to take a snapshot of the current system state. This snapshot contains the contents of all the virtual machine's disks and the contents in memory at the time of the snapshot. At a later point in time, the VM can be reverted back to this previous state and restarted. To see why this may be useful, consider the following scenario. Alice, a system administrator, is running an important web server on a virtual machine, and at some point in time there is a crash or some other major problem. Instead of spending time diagnosing the problem and getting the system working again, Alice can instead revert the VM back to a 'good state' for which she has a snapshot. In other words, Alice takes a snapshot of the system when things are running smoothly, and then reverts back to this state whenever things go wrong. In this scenario, the server has effectively traveled back in time; program variables and other state that may have been in memory are now active again. Thus, if an adversary is attacking Alice's server and can make it crash (using, for example, a DoS attack), he essentially has the ability to rewind the server.

Virtual machine state snapshots can also help protect against malware when web browsing. A user who is concerned about their machine being compromised from visiting a malicious website can run a web browser inside of a virtual machine and take a snapshot of the fresh machine state with a browser window open and ready for a URL. Then, if the user visits a malicious site, he can simply erase the current state of the machine and revert back to the fresh, uncompromised state captured in the snapshot to visit another website. Thus, every time the user wants to visit an important or potentially malicious website, he can do so starting from a fresh state. Previously, many of our computing applications such as word processors and video games had "save" features, while now, with virtual machines, our entire machine essentially has a "save" feature.

Though useful, state snapshots on VMs raise some important issues. What happens to our supposedly secure cryptographic tools in a setting with state resets? Are they still secure? Researchers have examined these questions before for zero-knowledge [27, 53, 7] and identification protocols [15], where the motivation was smart cards that cannot keep internal state. However, the growing popularity of virtual machines means we need to ask these questions for a wider range of cryptographic primitives.

To see why reset attacks can have negative effects on cryptographic protocols, consider a common assumption in cryptography: it is possible to continually generate fresh and unbiased random numbers. This is an assumption made in nearly every cryptographic protocol. It is, however, considered reasonable since random number generators (RNGs) are well-studied both in theory and practice (c.f., [45, 33]). In deployed systems, RNGs are often implemented in software and consist of numerous state variables and arrays that are occasionally seeded with entropy and used to generate random numbers. For example, in OpenSSL [3], the software RNG has a 1023-byte array (entropy pool) that is supposed to contain high entropy data from a variety of sources, as well as some variables with counters and other important state. At a high level, when random bytes are requested, data from the entropy pool and information in the state variables is continually mixed together using a cryptographic hash function and the result is the output of the RNG. However, these arrays and variables will be captured by a state snapshot since they reside in memory. If the machine is later reset, the RNG could output a string of "random" bytes that it already outputted sometime in the past before the machine was reset. These un-fresh coins might then be used in a cryptographic operation with potentially disastrous consequences.

Garfinkel and Rosenblum were the first to point out that this threat exists in theory [39]. In the next section we show that damaging attacks exist in practice. Specifically, we show that if a client runs a web browser inside of a virtual machine to, e.g., protect against malware, and in particular resets the virtual machine between browsing sessions, then the browser will send the same secret random keying material to two different websites. So, if from a saved state a user Alice visits a malicious site inside of the virtual machine, then resets the machine back to the saved state and visits her bank, the same secret key material will be sent by the browser to both the malicious site and the bank! An adversary in control of the malicious site can then compromise Alice's banking session.

Due to the danger posed by virtual machines, we propose building cryptographic primitives that are more resilient in the face of repeated randomness. As with the rest of this dissertation, we focus on public-key encryption, and make the following contributions. First, we provide formal security definitions to model public-key encryption security in the face of resetting attacks. Second, we show that existing PKE schemes and their common security notions IND-CPA [41] and IND-CCA [60] are insufficient when such resetting attacks are possible. Third, we show that, perhaps somewhat surprisingly, a small and efficient modification can be made to *any* existing PKE scheme secure under the typical notions (e.g., IND-CCA) in order to ensure security against resetting attacks. Our modification does not rely on random oracles [20], and is very efficient.

Previous Work. Resettability has been considered in cryptography in the setting of zero-knowledge proof systems [27, 53, 7], the related area of identification protocols [15], and multiparty computation [42]. Zero-knowledge proofs allow a prover to prove an assertion to a verifier without revealing any information other than whether or not the assertion is true. Proving the soundness[1] and zero-knowledge properties in a setting where provers and verifiers can rewind each other is a difficult and interesting theoretical question. To see why, consider the notion of resettable-soundness in the standard model, considered by [7]. Nearly all known

[1]Informally, an interactive protocol is sound if it is difficult for a malicious prover to convince the verifier that a false statement is true.

zero-knowledge proofs are designed specifically so that the ability to rewind the verifier allows one to easily convince it of any statement; this is useful for proving the zero-knowledge property. Yet, if we then give the prover that same ability to rewind the verifier, it becomes problematic to prove soundness. This problem has also been studied extensively in other models (c.f., [53]). However, to the best of our knowledge, no one has previously looked at practical and deployed cryptographic primitives like public-key encryption in such a setting.

In the symmetric setting, Rogaway and Shrimpton [63] investigate secure key-wrap and discuss how their techniques can apply to handle IV misuse, where IVs, which should always be fresh, are repeated (possibly because of a faulty implementation). Since IVs are typically counter variables or fresh random numbers, investigating their reuse is similar to investigating the effect of a state reset.

Our work is also loosely related to public-key encryption with randomness re-use [10] and stateful public-key encryption [18]. However, both are concerned with making PKE schemes more efficient by purposely reusing *some, but not all* random coins. Their schemes still require encrypting parties to have access to fresh and unbiased randomness.

Organization. The rest of this chapter is organized as follows. In the next section we describe the details of our attacks on TLS clients running on virtual machines and also discuss the difficulty or patching or even redesigning systems to deal with the attacks. Next, in Section 4.3 we give a new security definition, discuss variants, show how existing PKE schemes do not meet our definition, and finally show how to achieve it.

4.2 VM Reset Vulnerabilities Affecting TLS

To motivate the need for stronger PKE secure against reused randomness, in this section we explore virtual machine (VM) reset vulnerabilities. These arise when applications' security-critical state is captured by a VM snapshot and starting the VM repeatedly from the snapshot leads to security problems. The VM reset vulnerabilities we consider are due to cryptographic randomness being cached by applications and caught in a snapshot. Running multiple times from the snapshot results in cryptographic operations consuming repeated randomness, and in turn, failing to provide security.

4.2.1 TLS Client Vulnerabilities

TLS is used to secure HTTP connections over the Internet. Thus, TLS protects the security of online banking, shopping, and other sensitive traffic. Every popular web browser therefore includes a TLS client, which is used to negotiate a shared secret, called a session key, between it and the remote HTTP server. The most prevalent (for statistics see [67]) mode for establishing a session key is RSA key transport. Here the client chooses a secret value, called the premaster secret (PMS), encrypts it under the server's public RSA key, and then sends the resulting ciphertext to the server. The symmetric session keys used to secure the rest of the session are then derived from the PMS and two other values that are sent in the clear. While this is technically a type of key exchange, it can also be thought of as a type of interactive public-key encryption.

In abstract, a VM reset vulnerability could arise if the PMS, or the randomness used to create it, is generated before a snapshot and consumed upon resumption after the snapshot. This vulnerability would lead to an immediate compromise of sessions if the same PMS is sent to multiple different servers.

Before assessing whether this can occur in practice, we first ask: Why might a user run their browser in a virtual machine? Security experts recommend users do their web browsing within a VM to *increase* security. The idea is that if the browser has a vulnerability and a malicious site exploits it, the damage is contained to the VM. A user can revert to a previous snapshot taken before the browser and VM were compromised to undo the effects of any malware.

We performed experiments on a variety of browsers on both Linux and Windows to determine if there is a real problem. There is. Our results are summarized in Table 4.1. We explain the results in detail below.

Table 4.1: Summary of our TLS client attacks. We performed all of the experiments on both VMWare Server version 1.0.10 and VirtualBox version 3.0.12 and observed the same behavior. Ubuntu refers to version 8.04 (Hardy) Desktop, Windows refers to XP Professional with Service Pack 2.

TLS Client	Guest OS	Same PMS to diff. sites?	Same PMS to same site?	Comments
Firefox 3.5	Windows	Always	Always	Mouse moves $<$ 100 pixels
Chrome 3.0	Windows	Never	Sometimes	-
IE 6.0	Windows	Never	Sometimes	-
Safari 4.0	Windows	Never	Sometimes	-
Firefox 3.0	Ubuntu	Always	Always	Mouse moves $<$ 100 pixels
Chrome 4.0	Ubuntu	Always	Always	Visit one HTTPS site before snapshot

Experimental setup. We used two Apache web servers (call them server1 and server2) running on two separate physical machines. The servers used an instrumented version of OpenSSL that, upon receipt of the client's key exchange message in a TLS session using RSA key transport, would decrypt the premaster secret and write it to a file. Each server was given an RSA certificate signed by our own certificate authority (CA). We ran the various browsers (listed in Table 4.1) within the indicated operating systems as guests inside a VM running in either VMWare 1.0.10 or VirtualBox 3.0.12. The physical host ran Ubuntu 8.04 Desktop. The client browsers, excepting Safari in Windows, were configured to accept our CA. This ensured that, upon visiting one of our servers, a browser in the guest OS would not complain about a certificate signed by an untrusted CA. (For Safari, we ended up just clicking "continue" when presented with a warning about an untrusted certificate.)

Experiments. We start with the following test sequence.

1. Reboot the OS.
2. Load the browser.
3. Take a snapshot of guest in this state.
4. Reset the VM.
5. Navigate browser to server1.
6. Reset the VM.
7. Navigate browser to server2.

For each VM manager OS combination, steps (1-3) were performed once followed by 3 iterations of steps (4-7) for each browser. For Chrome on Linux, we also ran a separate test sequence where step (2) was changed to
(2a) Load the browser, navigate to an HTTPS url, and then navigate to a blank page.

The results were consistent between the two VM managers, meaning the VMM used had no impact on client behavior. For Firefox on Windows or Linux, the same PMS was sent to both servers in all 3 trials. If the user caused 100 mouse events (e.g., moved the mouse 100 pixels) between steps (4) and (5) or (6) and (7) then distinct PMS values were sent to the servers. This is because Firefox folds new entropy into the RNG every 100 mouse events. For Chrome on Linux, when step (2a) was used then the same PMS was sent to both servers in all 3 trials. When step (2) was used, distinct PMS values were sent to the two servers.

On Windows, all browsers except Firefox always sent distinct PMS values to both servers. We note however that on Windows, the same PMS value was sent to the same server in many of the trials. While this does not admit an obvious attack, it violates the TLS specification. For example, on IE 6.0 and VMWare, 2 out of the 3 PMS values sent to server1 were the same and 2 out of the 3 PMS values sent to server2 were the same. We note that all the browser/VMM combinations showed this problem; for Chrome in Windows, it did not even matter whether or not step (2a) or (2) was used.

Attacks on Servers. Virtual machine snapshots can also lead to randomness reuse in server applications, as we show in [61]. However, as those attacks focus on digital signatures and not encryption, they are out of scope for this dissertation.

4.2.2 On Fixing the Vulnerabilities

In the TLS clients we described above, we saw that good randomness was sampled at some point (such as starting the program or launching a child process) and buffered until it was needed at some much later time. This allowed a large window in which snapshots would capture to-be-used randomness. In the browser client vulnerabilities, the randomness was used directly in a cryptographic operation after the snapshot.

In abstract, fixing these vulnerabilities requires ensuring that RNGs get access to sufficient entropy after a snapshot and ensuring that applications take randomness from an RNG at the time of the cryptographic operation. For example, one approach would be to mandate using a guest OS source such as `/dev/random` or `/dev/urandom` to generate randomness right before a cryptographic operation is performed.

Unfortunately, the state of these sources is also reset by snapshots, and so it is unclear whether sufficient entropy is generated between a snapshot resumption and randomness consumption by the cryptographic operation. In general, a better option would likely be linking guest RNG services with hardware-based RNGs or other external sources.

This is a large topic, and we leave finding the best solutions to future work. Instead, we turn our attention to strategies for mitigating the threat of these reset vulnerabilities by building stronger public-key encryption.

4.3 Resettable Public-Key Encryption

As we saw in the previous section, virtual machines and their snapshot feature can cause repeated randomness in cryptographic primitives, leading to damaging attacks. We also discussed how it is not clear how to completely fix the problem by simply patching systems or even designing better systems. Instead of hoping systems researchers are able to fix the problem, or that people will stop using snapshots on virtual machines, we instead propose building stronger cryptographic tools that do not completely fail when randomness is repeated. In this section we propose new, stronger models of security for public-key encryption that take into account the possibility that randomness may be repeated. Many existing schemes are not secure under these stronger models (we give a few notable examples later in this section), so we then show how to make a simple change to any scheme secure under typical security notions (e.g., IND-CPA or IND-CCA) to make it provably resist attacks that arise due to repeated randomness.

Our security notion is similar to IND-CPA and IND-CCA, except that we allow the adversary to continually see encryptions under the same coins, as if the adversary is repeatedly resetting a VM and observing new encryptions. An important aspect of our security definition is that we allow the adversary to see encryptions *under public keys of its choice* and using coins that are not fresh. In particular, the adversary could see a message encrypted under a public key for which it knows the secret key, allowing it to decrypt the ciphertext; because of this, it is important that in the process of decryption not too much information is leaked about the coins used to create the ciphertext, meaning that randomness-recovering encryption cannot meet our security definition. Allowing this power in the definition is important because it models the possibility that a machine sends an encrypted message to some user Bob, is reset by the adversary, and is then forced to encrypt a message to the adversary using the same coins. We want to ensure that even if this happens, the adversary does not learn any information about Bob's message. This is a strong security requirement, but nonetheless, we are able to meet it.

We note that though our security notion provides seemingly the best possible security guarantees for PKE under reset attacks, it may still be insufficient for some applications. This is due to an inherent limitation in a model that allows repeated randomness: if the same message is encrypted twice to the same public key using the same randomness, the resulting ciphertexts will be identical. Thus, plaintext equality may be leaked to an adversary, which could be problematic in some applications. Therefore, we are not proposing resettably secure encryption as a complete solution to virtual machine reset attacks. Instead, we believe that resettably secure encryption should be used in conjunction with systems solutions, some of which are discussed in [61]. In other words, similar to [13], our constructions are a way to hedge against system failures; in our case, if the randomness happens to be repeated, then our schemes do not fail immediately, but instead still provide some meaningful, provable security guarantees.

4.3.1 Security Definition

We now give the details of our new security definition. Let $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme with randomness length ρ. Consider game RA in Figure 4.1. We say an RA-adversary is one who plays game RA and makes exactly one query to its **LR** oracle consisting of two equal-length messages, zero or more queries to

```
procedure Initialize:
    $b \leftarrow_{\$} \{0,1\}$
    $pars \leftarrow_{\$} \mathcal{P}(1^k)$
    $(pk^*, sk^*) \leftarrow_{\$} \mathcal{K}(pars)$
    $r^* \leftarrow_{\$} \{0,1\}^{\rho(k)}$ ; $S \leftarrow \emptyset$
    Ret $pk^*$

procedure Enc($pk, m$):
    $c \leftarrow \mathcal{E}(pk, m; r^*)$
    Return $c$

procedure LR($m_0, m_1$):
    $c \leftarrow \mathcal{E}(pk^*, m_b; r^*)$
    $S \leftarrow S \cup \{c\}$
    Return $c$

procedure Dec($c$):
    If $c \in S$ then return $\perp$
    Else return $\mathcal{D}(sk^*, c)$

procedure Finalize($b'$):
    Ret $(b = b')$
```

Figure 4.1: Game $\mathrm{RA}_{\mathcal{AE},k}$.

the **Enc** oracle, and zero or more queries to the **Dec** oracle. We then say the ra-advantage of an RA-adversary A is

$$\mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\, \mathrm{RA}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true} \,\right] - 1 \ .$$

In game RA, the adversary is given a target public key pk^* and can make queries to three oracles. It can query the **LR** oracle with messages m_0 and m_1. In response, the adversary receives the encryption of m_b under the target public key pk^* using the coins r^* chosen by **Initialize**. The adversary is also given an **Enc** oracle which takes as input a public key pk and message m. The oracle returns the encryption of m under public key pk, again using the coins r^* chosen in **Initialize**. It is important that the adversary can choose the public key pk. In particular, the adversary can query **Enc** with a public key for which it knows the corresponding secret key. Notice the game is similar to INDCCA, but with the addition of the **Enc** oracle so that the adversary can continually see messages encrypted under the same coins used by **LR**. This is how we model resetting attacks. The adversary can also query a **Dec** oracle with a ciphertext (not returned by **LR**) and receive its decryption. Finally, the adversary outputs a guess bit.

Equality Patterns. If there are no restrictions on the **LR** queries that an RA-adversary A can make, then A can trivially win the game. To see this, consider an

RA-adversary that first queries $\mathbf{Enc}(pk^*, m)$ and then queries $\mathbf{LR}(m, m')$, where $m \neq m'$ are equal-length messages. The **Enc** query will give the adversary the encryption of m under coins r^*, and the **LR** query will give the adversary either the encryption of the same message m under the same coins r^*, or it will give the adversary the encryption of m' under coins r^*. Clearly the adversary only needs to compare the two oracle answers and guess 0 if they are the same and guess 1 otherwise.

This attack is an inherent limitation of the resettable PKE setting, since for fixed coins encryption becomes a deterministic function. (It is also therefore similar to limitations in the setting of deterministic PKE [12].) Nevertheless, as we said earlier, we are interested in achieving the best security possible in this situation. Therefore, we consider security against all adversaries that do not trivially win. This informal notion is captured formally by the following definition:

Let A be an RA-adversary making **LR** query (m, m') and q queries (pk_1, m_1) to (pk_q, m_q) to **Enc**. Then we say that A is *equality-pattern respecting* if for all $i \in [q]$, $pk_i = pk^*$ only if $m_i \notin \{m, m'\}$. In other words, an equality-pattern respecting adversary never queries **Enc** on the target public key pk^* and a message that appears in its **LR** query.

We can now define RA security. We say a PKE scheme $\mathcal{AE}$ is IND-R-CPA-secure if for all efficient equality-pattern respecting RA-adversaries A making 0 **Dec** queries the ra-advantage of A with respect to $\mathcal{AE}$ is negligible. Similarly, we say $\mathcal{AE}$ is IND-R-CCA-secure if for all efficient equality-pattern respecting RA-adversaries A the ra-advantage of A with respect to $\mathcal{AE}$ is negligible.

Alternative Definitions. We could instead consider a more complex definition in which there is more than one randomness r^* under which the adversary gets to see encryptions. Additionally, we could also allow the adversary more than one **LR** query. We present this more complex definition later in this section and show that security under it is implied by security under the simpler definition given in this section.

Relation to IND-CPA and IND-CCA. Now that we have formally defined resettable security for public-key encryption, it is useful to compare it to indistinguishability under chosen plaintext and chosen-ciphertext attacks, the typical notions of security for PKE. First, it is easy to see that for XXX as either CPA or CCA, any scheme that is IND-R-XXX is also IND-XXX, since RA is identical to INDCCA except for the additional **Enc** oracle. Thus, any INDXXX-adversary can easily be turned into an RA-adversary making zero **Enc** oracle queries. Second, we prove the following:

Proposition 4.3.1 For XXX $\in \{\mathrm{CPA}, \mathrm{CCA}\}$, if there exists a scheme $\overline{\mathcal{AE}}$ that is IND-XXX secure, then there exists scheme $\mathcal{AE}$ that is IND-XXX secure but is not IND-R-XXX secure. $\square$

Proof: We will prove for XXX=CCA, but the proof easily extends to the CPA setting. Let $\overline{\mathcal{AE}} = (\overline{\mathcal{P}}, \overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be an arbitrary IND-CCA scheme. We construct a new PKE scheme $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ such that $\mathcal{AE}$ is still IND-CCA secure, but $\mathcal{AE}$ is not IND-R-CCA secure. The scheme $\mathcal{AE}$ has encryption algorithm $\mathcal{E}(pk, m; r \,\|\, K \,\|\, K')$ that outputs $c_1 \,\|\, c_2 \,\|\, c_3$, where $c_1 = \overline{\mathcal{E}}(pk, K \,\|\, K'; r)$, $c_2 = K \oplus m$ and $c_3 = \mathsf{MAC}_{K'}(c_2)$. The IND-CCA security of $\mathcal{AE}$ follows from the well-known KEM/DEM composition theorem of [32]. We can construct an RA-adversary A with advantage 1 against $\mathcal{AE}$. Adversary A, upon receiving target public key pk^*, queries the **Enc** oracle with $(pk^*, 0^{n(k)})$ and immediately learns K from the response, since K is xor'd with all 0s. Then, A queries $\mathbf{LR}(m_0, m_1)$ for unique messages m_0 and m_1 (which do not equal the string of all zeroes). A can then use K to decrypt the response and win the game. $\square$

Discussion. There are a few important aspects of our security definition that require more discussion. First, as shown in Proposition 4.3.1, our definition is stronger than previous notions of security. Since we are concerned about random coins being repeated, one might ask why we even need a new definition and do not just use deterministic public-key encryption [12], eliminating the coins altogether. The reason is that we still want our schemes to meet the previous definitions (i.e., IND-CCA) to ensure they have as much security as possible, and it is well-known that no deterministic scheme can ever be IND-CCA (or IND-CPA) secure.

```
procedure Initialize:
$b \leftarrow_\$ \{0,1\}$
$pars \leftarrow_\$ \mathcal{P}(1^k)$
$(pk^*, sk^*) \leftarrow_\$ \mathcal{K}(pars)$
$\texttt{CoinTab} \leftarrow \emptyset$ ; $S \leftarrow \emptyset$
Ret $pk^*$

procedure Enc($pk, j, m$):
If $\texttt{CoinTab}[j] = \perp$ then
    $\texttt{CoinTab}[j] \leftarrow_\$ \{0,1\}^{\rho(k)}$
$r_j \leftarrow \texttt{CoinTab}[j]$
$c \leftarrow \mathcal{E}(pk, m; r_j)$
Return $c$

procedure LR($j, m_0, m_1$):
If $\texttt{CoinTab}[j] = \perp$ then
    $\texttt{CoinTab}[j] \leftarrow_\$ \{0,1\}^{\rho(k)}$
$r_j \leftarrow \texttt{CoinTab}[j]$
$c \leftarrow \mathcal{E}(pk^*, m_b; r_j)$
$S \leftarrow S \cup \{c\}$
Return $c$

procedure Dec($c$):
If $c \in S$ then return $\perp$
Else return $\mathcal{D}(sk^*, c)$

procedure Finalize($b'$):
Ret $(b = b')$
```

Figure 4.2: Game $\mathrm{RA2}_{\mathcal{AE},k}$.

Second, we allow the adversary to give arbitrary public keys to the **Enc** oracle and see the resulting ciphertexts under those keys and the repeated coins. As mentioned in the introduction, this is important to model the situation in which a machine is reset and then an encryption is sent to the adversary; we want to make sure other encryptions using the same coins still maintain their privacy. This aspect of our definition resembles a similar ability allowed in the definition of stateful PKE [18]. This aspect of the definition is also especially important since it closely mirrors the attack we showed on TLS in the previous section.

Third, one might wonder what our equality pattern restriction means in practice. It simply reflects the fact that if a message is encrypted twice using the same public key and the same coins, then the resulting ciphertexts will be the same. An adversary observing the two ciphertexts will know that the underlying plaintexts are the same. This attack is unavoidable in the resettability setting, and whether or not it is a problem will depend on the application.

4.3.2 An Equivalent Security Definition

As we mentioned in Section 4.3.1, we can consider a more complicated security game to capture reset security. Let $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be a PKE scheme. We say the ra2-advantage of an adversary A is

$$\mathbf{Adv}^{\mathrm{ra2}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\, \mathrm{RA2}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true} \,\right] - 1 \; .$$

The security game RA2 can be found in Figure 4.2. In the game, the adversary is given a target public key pk^* and can make queries to three oracles. It can query the **LR** oracle with index j and messages m_0 and m_1. In response, the adversary receives the encryption of m_b under the target public key pk^* using the coins indexed by j. The adversary is also given an **Enc** oracle which takes as input a public key pk, index j, and message m. The oracle returns the encryption of m under public key pk using the coins indexed by j. It is important that the adversary can choose the public key pk. In particular, the adversary can query **Enc** with a public key for which it knows the corresponding secret key. With both the **LR** and **Enc** oracles, an adversary can continually see messages encrypted under the same coins by repeatedly querying the same index. This is how we model resetting attacks. Of course, the adversary can also see messages encrypted under other coins by querying other indices. Finally, the adversary can also query a **Dec** oracle with a ciphertext and receive its decryption.

Equality Patterns. For our alternate definition, we need a much more complicated notion of equality patterns. Let A be any adversary that queries I different indices to its **LR** and **Enc** oracles and makes q_i queries to the **LR** oracle with index i. Let E_i be the set of all messages m such that A makes query $\mathbf{Enc}(pk^*, i, m)$. Let $(m_0^{i,1}, m_1^{i,1})$ to $(m_0^{i,q_i}, m_1^{i,q_i})$ be A's **LR** queries for index $i \in [I]$. Then, if for all $i \in [I]$ and for all $j \neq k \in [q_i]$,

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

and for all $i \in [I]$ and all $j \in [q_i]$

$$m_0^{i,j} \notin E_i \wedge m_1^{i,j} \notin E_i,$$

then we say that A is equality-pattern respecting.

It is easy to see that if we only consider adversaries that query **LR** once and query **LR** and **Enc** on only a single randomness index, then the definition becomes equivalent to the definition in Section 4.3.1. (`CoinTab` has only one entry, which we call r^* in the simpler definition.) To justify our use of the simpler security game, we use a hybrid argument to prove the following lemma:

Lemma 4.3.2 Let $\mathcal{AE}$ be a PKE scheme and A be an equality-pattern respecting adversary querying **LR** and **Enc** on combined at most d different indices and querying **LR** for any given index at most q times. Then there exists an equality-pattern respecting RA-adversary C making one **LR** query such that

$$\mathbf{Adv}^{\mathrm{ra2}}_{\mathcal{AE},A}(k) \leq d \cdot q \cdot \mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},C}(k) \ .$$

$\square$

Proof of Lemma 4.3.2 (Sketch). The proof is similar to the proof in [11] showing PKE IND-XXX security for multiple receivers is implied by one receiver (traditional) IND-XXX security. Let A be any equality pattern respecting adversary querying **LR** and **Enc** on at most d different randomness indices and querying **LR** for any given index at most q times. Assume without loss that A does not repeat any **LR** or **Enc** queries. Clearly the number of total **LR** queries over all indices is at most $d \cdot q$. We will build an adversary B querying **LR** and **Enc** on at most 1 randomness index and querying **LR** only once. This adversary can immediately be translated into an adversary C playing game RA and making one **LR** query since with only one index the games are the same modulo some different syntax. The adversary B runs A and guesses a value $j \in \{1, \ldots, dq\}$. Let $i = \lceil j/q \rceil$ and $\ell = j - q(i-1)$. This means that j corresponds to the ℓth **LR** query made with index i. Once j is guessed, B chooses coins r_t for $t \in [d] \setminus i$ and proceeds to answer

A's **LR** queries as follows. For any **LR** query with index $i' < i$, B encrypts m_0 with coins $r_{i'}$. For any **LR** query with index $i' > i$, B encryptions m_1 with coins $r_{i'}$. Now, if $i' = i$ and this is the wth **LR** query with that index, then B answers with its own **LR** oracle when $w = \ell$, while it uses its own **Enc** oracle with m_0 for $w < \ell$ and its own **Enc** oracle with m_1 for $w > \ell$. It answers **Enc** queries from A with index i using its own **Enc** oracle and all other **Enc** queries with the coins it chose itself. Decryption queries are all forwarded. Finally, B outputs the same bit as A. Standard calculations show a security loss of $d \cdot q$. $\square$

4.3.3 Insecurity of Existing Schemes

Now that we have fully specified our target security notion we can examine whether existing schemes meet it. Unfortunately, many natural schemes do not. We give two examples.

El-Gamal Encryption. El-Gamal encryption, one of the simplest and most well-known encryption schemes, is IND-CPA under the Decisional Diffie-Hellman assumption, yet is not IND-R-CPA secure. To attack the scheme, an adversary, after receiving public key $pk = g^x$, should query **LR** with two distinct group elements h_0 and h_1. The adversary receives back $(C_0, C_1) = (g^r, g^{xr} \cdot h_b)$. Next, the adversary queries **Enc** with pk and $2 \cdot h_0$. If the adversary receives back $(C_0, 2 \cdot C_1)$, it knows the challenge bit is 0; if not, the challenge bit is 1. This is because the second component of the encryption of $2 \cdot h_0$ with randomness r is $g^{rx} \cdot (2 \cdot h_0) = 2 \cdot C_1$.

Hybrid Encryption. Recall that in a hybrid encryption scheme, a PKE scheme is used to encrypt not a message, but a symmetric encryption key. This symmetric key is then used in a symmetric encryption scheme to actually encrypt the message. In other words, a ciphertext is a pair $(\mathcal{E}(pk, K), \mathsf{SE}(K, M))$. In such schemes, the first step is typically to choose a random symmetric key. When randomness is repeated, the same symmetric key will be chosen twice. This was exactly what led to our attack on TLS clients running on virtual machines. More formally, to achieve high advantage in game RA against a hybrid encryption scheme, an adversary would query the **LR** oracle with any two distinct messages. Next, the adversary would generate its own keypair (pk, sk) and request an encryption from the **Enc** oracle with pk. The **Enc** oracle will choose the same symmetric key as the **LR** oracle, and the adversary will be able to learn this key by using its secret key sk. It can then decrypt the **LR** query and win the game.

4.3.4 Achieving IND-R-XXX Security

In this section we show that we can make a simple and efficient modification to any IND-XXX PKE scheme and immediately get an IND-R-XXX secure scheme. Our transformation relies only on the existence of pseudorandom functions. This means that if we take a PKE scheme that is IND-XXX secure in the standard model, our modified scheme will be IND-R-XXX secure in the standard model.

Let $\overline{\mathcal{AE}} = (\overline{\mathcal{P}}, \overline{\mathcal{K}}, \overline{\mathcal{E}}, \overline{\mathcal{D}})$ be a PKE scheme and let $\mathsf{Fun} : \mathsf{Keys}_k \times \mathsf{Dom}_k \to \mathsf{Rng}_k$ be a family of functions with $\mathsf{Keys}_k = \{0,1\}^{\rho(k)}$, $\mathsf{Dom}_k = \{0,1\}^{s(k)}$, and $\mathsf{Rng}_k = \{0,1\}^{\rho(k)}$. The domain size $\{0,1\}^{s(k)}$ should be large enough to encode any public key generated from $\overline{\mathcal{K}}$ and a message in $\{0,1\}^{n(k)}$. We build a PKE scheme $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ from $\overline{\mathcal{AE}}$ and $\mathcal{F}$ as follows. Parameter generation, key generation and decryption are the same as in $\overline{\mathcal{AE}}$, and $\mathcal{E}(pk, m; r)$ computes $\bar{r} \leftarrow \mathsf{Fun}(r, (pk \,\|\, m))$ and returns $\overline{\mathcal{E}}(pk, m; \bar{r})$.

Theorem 4.3.3 If $\overline{\mathcal{AE}}$ is IND-XXX secure and Fun is a secure PRF, then $\mathcal{AE}$ is IND-R-XXX secure. □

Proof: We prove the CCA case but the proof can be easily adapted to the CPA case. Let $\mathcal{AE}$ be constructed from Fun and $\overline{\mathcal{AE}}$ as above. Let A be an efficient equality-pattern respecting RA-adversary attacking $\mathcal{AE}$. We assume that A never makes duplicate queries to the **Enc** oracle; this is without loss because all such queries will return the same response. It is easy to see that this fact combined with the fact that A is equality-pattern respecting means that every query A makes to **Enc** and **LR** results in a unique combination of pk and m; the game will never encrypt the same message twice under the same public key. Now, denote by G_0 the game $\mathrm{RA}_{\mathcal{AE},k}$ defined in Section 4.3.1. Thus by definition,

$$\mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A}(k) = 2 \cdot \Pr\left[\, G_0^A \,\right] - 1 \;.$$

Now consider game G_1. The relevant procedures from games G_0 and G_1 are shown in Figure 4.3. In G_0, oracles **LR** and **Enc** use Fun to derive the randomness used to encrypt since this is what $\mathcal{AE}$ does. However, in G_1, those oracles choose fresh random coins and use those to encrypt the messages. We claim that these games appear close to adversary A by showing there exists an efficient PRF-adversary B such that

$$\Pr\left[\, G_0^A \,\right] - \Pr\left[\, G_1^A \,\right] \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Fun},B}(k) \;.$$

The adversary B, attempting to decide if it is in the real or random world, flips a bit b and chooses a target public key pk^* by running the key generation algorithm. B then runs A just as in G_0 and G_1. On **Enc** and **LR** queries, B uses its **Fun** oracle to derive the randomness for encryption; in the case of the **LR** query, B encrypts the message corresponding to the bit b that it chose. In the CCA case, B answers **Dec** queries simply by using the secret key sk^* (which it knows because it chooses pk^* and sk^*). When A eventually outputs a guess bit b', B outputs 1 if $b = b'$ and 0 otherwise. We can see that when B is in the 'real' world (i.e., its **Fun** queries are answered using Fun), it perfectly simulates G_0 for A, while if B is in the 'random' world (i.e., its **Fun** queries are answered with random range points) then it perfectly simulates G_1 for A. The claim follows.

We then claim that there exists an efficient INDCCA-adversary C such that

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{AE},C}(k) = 2 \cdot \Pr\left[\, G_1^A \,\right] - 1 \;.$$

The adversary C is given a target public key pk^* and access to an **LR** oracle to which it can make a single query. It also has access to a **Dec** oracle. Adversary C runs A as in G_1, answering its oracle queries as follows. On A's single **LR** query, C simply answers with its own **LR** oracle. C answers A's **Dec** queries using its own

```
procedure Enc(pk, m):                      procedure LR(m_0, m_1):   Game G_0
$\bar{r} \leftarrow \mathsf{Fun}(r^*, (pk \| m))$          $\bar{r} \leftarrow \mathsf{Fun}(r^*, (pk^* \| m_b))$
$c \leftarrow \overline{\mathcal{E}}(pk, m; \bar{r})$                $c \leftarrow \overline{\mathcal{E}}(pk^*, m_b; \bar{r})$
Return $c$                                  $S \leftarrow S \cup \{c\}$
                                           Return $c$
```

```
procedure Enc(pk, m):                      procedure LR(m_0, m_1):   Game G_1
$\bar{r} \leftarrow_{\$} \{0,1\}^{\rho(k)}$                   $\bar{r} \leftarrow_{\$} \{0,1\}^{\rho(k)}$
$c \leftarrow \overline{\mathcal{E}}(pk, m; \bar{r})$                $c \leftarrow \overline{\mathcal{E}}(pk^*, m_b; \bar{r})$
Return $c$                                  $S \leftarrow S \cup \{c\}$
                                           Return $c$
```

Figure 4.3: Games for the proof of Theorem 4.3.3. The procedures **Initialize**, **Finalize**, and **Dec** are omitted for brevity.

Dec oracle. On **Enc** queries from A, C encrypts the messages itself using fresh randomness and returns the resulting ciphertexts to A. At the end of execution, C outputs the same bit that A guesses. It is easy to see that C perfectly simulates the G_1 game for A and the claim follows.

Combining the above equations we can see that

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ra}}_{\mathcal{AE},A}(k) &= 2 \cdot \Pr\left[\, G_0^A \,\right] - 1 \\
&\leq 2 \cdot (\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Fun},B}(k) + \Pr\left[\, G_1^A \,\right]) - 1 \\
&\leq 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{Fun},B}(k) + \mathbf{Adv}^{\mathrm{ind\text{-}cca}}_{\overline{\mathcal{AE}},C}(k)\ .
\end{aligned}$$

□

The existence of secure PRFs is implied by the existence of one-way functions (which are necessary for PKE to exist), so we do not need any additional assumptions. In practice, one would want to instantiate the PRF using HMAC [14] or a block-cipher such as AES.

Extensions. A natural question to ask is what happens to security if the same randomness is used across multiple different primitives. For example, what if some randomness r is used for public-key encryption, but then after a virtual machine reset r is used for DSA signing? Formally modeling this situation is an interesting open problem. However, we conjecture that our PRF approach in this section will generalize and provide security in such a setting. Specifically, to protect a primitive P against repeated randomness, one would want to apply the PRF not only to P's inputs, but also to some unique value identifying P, e.g., an algorithm ID. This should guarantee that different primitives use distinct randomness, even after a reset.

4.4 Conclusion and Additional Information

In this chapter we first showed practical attacks against existing, deployed cryptography that arise due to randomness repetition on virtual machines. We then showed how to build more robust cryptography that still provides provable guarantees even when randomness can be repeated. This involved coming up with new, stronger security models, and also showing ways to modify existing schemes to achieve this stronger level of security.

Credits. Section 4.2 is a partial reprint of material [61] that appeared in the Proceedings of the Network and Distributed System Security Symposium, copyright the Internet Society, and co-authored with Thomas Ristenpart. I was a primary researcher for this work. The rest of the sections of this chapter are expanded versions of material [66] that appeared in *Topics in Cryptology – CT-RSA 2010*, copyright Springer. I was a primary researcher for this work and the sole author.

Chapter 5

Revealed Randomness

This chapter focuses on PKE security under various types of selective opening attacks. The strongest type of selective opening attack we consider involves a randomness failure where the coins used for encryption are revealed at some point to an adversary. In fact, as we shall see, the coins being revealed leads to the core technical difficulties.

5.1 Overview

A selective opening attack against a PKE scheme is one in which an adversary sees encryptions of many possibly-related messages and is then able to select some subset of these ciphertexts to "open". What do we mean by open? Think of a ciphertext as a locked box with a message inside. One can then think of opening a ciphertext as convincing someone that a particular message is contained in the box. Some types of opening may be more convincing than others.

The main type of opening we consider in this chapter has the adversary learn the message encrypted as well as the coins used for encryption. Specifically, after seeing encryptions $\mathcal{E}(pk, m_j\ ;\ r_j)$ for $j \in \{1, \ldots, n\}$, and where the messages are potentially related, the adversary selects some subset $I \subseteq \{1, \ldots, n\}$. The ciphertexts corresponding to I are opened by giving the adversary m_i, r_i for $i \in I$. Notice that the adversary, given m_i and r_i (and the known public key pk) can recompute $\mathcal{E}(pk, m_i\ ;\ r_i)$ and verify that the opening is consistent with the ciphertexts it observed. Certainly the opened encryptions can no longer provide any privacy; the main question surrounding selective opening attacks is whether the *unopened* encryptions still provide privacy.

One might think that our existing strong notions of security, such as IND-CPA or IND-CCA, should imply security under the selective opening attack above. While this is plausible, no one knows how to prove it. We point out that if only the message is revealed upon a corruption (not the coins), then it is easy to see that IND-CPA *does* imply selective opening security. We give details later in the chapter. Thus, it appears that a revealed randomness failure causes the main technical hurdles.

Encryption security under selective opening attacks is an important special case of adaptively secure encryption. Adaptively secure encryption, where an adversary is able to corrupt some parties after seeing ciphertexts, has roots in secure multiparty computation [22, 29]. However, past work on the problem has typically focused on receiver corruptions, where the recipient of a ciphertext is corrupted, exposing their secret key (and thus the message) to the adversary. Unfortunately, the known methods for achieving security under receiver corruptions have significant drawbacks, and do not look like typical encryption schemes. Specifically, Canetti, Feige, Goldreich, and Naor [26] show that non-committing encryption solves the problem, but their schemes have keys that are as long as the number of message bits sent. Compare this to typical PKE schemes, where keys are relatively short (say, 1000 bits), but are routinely used to encrypt millions of bits. Unfortunately, Nielsen shows this drawback is necessary [55]. Later work by Canetti, Halevi, and Katz [28] was able to shorten the keys, but at the cost of requiring that secret keys are updated periodically (e.g., every day) with previous keys erased completely.

Alternatively, the selective opening attacks we consider in this chapter only involve corruption of the *sender*. Thus, at most the adversary learns all of the sender's inputs to the encryption algorithm, meaning both the message sent and the coins used. An adversary corrupting senders never learns the secret key. This allows us to get around the negative results of [55].

To see why the sender corruption case is by itself interesting and important, consider the TLS/SSL protocol for HTTPS, the most common way PKE is used in practice. At any one time there are many users buying books on a site like `amazon.com` and using TLS to secure their transactions. Thus, there are many senders – the users – and one receiver – Amazon. Further, most of the senders are probably using web browsers and operating systems with unpatched security vulnerabilities, while Amazon likely goes to great lengths to secure their secret keys. In this scenario it is easy to see that the senders are more easily corrupted by an adversary, motivating our study of selective opening attacks for senders.

Dwork, Naor, Reingold, and Stockmeyer [36] were the first to focus specifically on the case of sender corruptions and recognize the problem's importance and difficulty. Specifically, they explored whether typical PKE schemes (i.e., non-interactive and with short keys) are secure when an adversary can adaptively corrupt half of the ciphertexts and learn the corresponding messages and coins. They observed that encryption usually constitutes a *commitment* to the message. Informally, this means that any given ciphertext can only be opened to one possible message. This seems to follow from the correct decryption property, since if a ciphertext could represent two different messages how would the decryption algorithm know which one it should output?

Dwork et al. then pointed out that this commitment property seems to lead to the core technical difficulties. They showed that the existence of commitment schemes secure under selective opening attack implies the existence of three-round zero-knowledge and magic functions for Fiat-Shamir. Thus, finding such commitment schemes would solve two difficult and long-standing open problems in cryptography. This gave strong evidence that resolving the selective opening problem would be difficult. More recently, Hofheinz gave further evidence of this difficulty [47]. Specifically, he showed that it is impossible to prove in a black-box way that any non-interactive or perfectly-binding commitment scheme meets a simulation-based definition of SOA security.

At this point, let us review the status of the problem. For receiver corruptions, we have numerous encryption schemes that are secure, but all have undesirable properties, like unreasonably long keys, or requiring periodic key updates and erasures. Impossibility results [55] show this is in some sense necessary. For sender corruptions, our main focus in this chapter, encryption schemes with short keys and without erasures typically act as commitments. Unfortunately, both Dwork et al. [36] and Hofheinz [47] give strong evidence that the problem will be difficult to resolve for commitment schemes. It seems there is little hope of finding encryption schemes secure under selective opening attacks.

Luckily, though many PKE schemes with short keys constitute commitments, this is not always the case. Since a ciphertext should decrypt to a unique ciphertext, how can an encryption not constitute a commitment? The answer is that there are many encryption schemes where encryption is committing, *but only under keys generated by the honest key generation algorithm.* For these schemes there will be an alternate key generation algorithm that outputs fake public keys. These fake keys look just like real keys, but encryption under a fake key is no longer committing, and will in fact statistically lose essentially all information about the message being encrypted.

We call schemes with this property lossy encryption schemes due to their similarity to the recently-introduced notion of lossy trapdoor functions [58]. Lossy encryption is a formalization of what Peikert, Vaikuntanathan, and Waters [57] informally call messy encryption while defining a related notion called Dual-mode encryption. It is also very similar to the notion of meaningful/meaningless encryption defined by Kol and Naor [50]. Numerous lossy encryption schemes exist; we show a few notable examples later in the chapter.

Organization. The rest of the chapter is organized as follows. The next section describes notions of PKE security under selective opening attack. Section 5.3 shows IND-CPA is equivalent to selective message opening security and discusses why a revealed randomness failure leads to technical difficulties. Section 5.4 introduces lossy encryption, the main tool for achieving selective randomness opening security. Section 5.5 proves lossy encryption gives selective randomness opening security.

5.2 Encryption Security under Selective Opening Attack

In this section we consider the security of public-key encryption under two types of selective opening attacks. One we call *selective message opening*, and informally means the adversary is able to adaptively corrupt a subset of senders after seeing ciphertexts and learn the *messages* encrypted. The other we call *selective randomness opening*, which informally means the adversary is able to adaptively corrupt a subset of senders after seeing ciphertexts and learn the messages *and the coins used to encrypt them.* For each type of selective opening attack we will present both simulation and indistinguishability-based definitions of security. Foreshadowing, we will show in future sections that IND-CPA implies security under selective message opening, while lossy encryption schemes (defined later in Section 5.4) are secure under selective randomness opening.

5.2.1 Message Sampling and Resampling

Let $n(\cdot)$ be a positive polynomial taking integer values. An n-message sampler $\mathcal{M}$ is a (possibly randomized) algorithm that on input security parameter 1^k and a string $\alpha \in \{0,1\}^*$, outputs a vector $\mathbf{m}$ of $n(k)$ messages. We say an n-message sampler is efficient if it runs in polynomial time in the security parameter.

For some of our definitions, we will consider message samplers that have two modes. An n-message sampler $\mathcal{M}$ is in the "sample" mode if it is given inputs 1^k and bitstring α, while it is in the "resample" mode if it is given the same two inputs as above and additionally a set I, a vector of $|I|$ messages $\mathbf{m}_I$, and a vector $\mathbf{z}$. We say a message sampler is efficiently resamplable if it runs in polynomial time in the security parameter regardless of whether it is in the "sample" or "resample" modes.

To capture how well a message sampler can resample, we consider two games RSMPREAL and RSMPIDEAL, shown in Figure 5.1. Game RSMPREAL has a **Challenge** procedure that resamples using the resample mode of $\mathcal{M}$, while game RSMPIDEAL instead does perfect resampling. We let $\mathcal{A}$ be the set of all (even

```
procedure Initialize:                                   Game RSMPREAL$_{\mathcal{M},k}$
Return $1^k$
procedure Challenge($\alpha, I, \mathbf{m}$):
For $i = 1 \ldots |\mathbf{m}|$ do $\mathbf{z}[i] \leftarrow 1^{|\mathbf{m}[i]|}$
$\mathbf{m}^* \leftarrow_{\$} \mathcal{M}(1^k, \alpha, I, \mathbf{m}[I], \mathbf{z})$
Return $\mathbf{m}^*$
procedure Finalize($b$):
Return $b$
```

```
procedure Initialize:                                   Game RSMPIDEAL$_{\mathcal{M},k}$
$S \leftarrow \emptyset$
Return $1^k$
procedure Challenge($\alpha, I, \mathbf{m}$):
For $i = 1 \ldots |\mathbf{m}|$ do $\mathbf{z}[i] \leftarrow 1^{|\mathbf{m}[i]|}$
For $\rho \in \mathsf{Coins}_{\mathcal{M}}(1^k, \alpha)$ do:
    $\mathbf{m}' \leftarrow \mathcal{M}(1^k, \alpha\,;\,\rho)$
    If $(\mathbf{m}'[I] = \mathbf{m}[I] \wedge \forall i\ |\mathbf{m}[i]| = |\mathbf{z}[i]|$ then $S \leftarrow S \cup \{\rho\}$
$\rho \leftarrow_{\$} S$
$\mathbf{m}^* \leftarrow \mathcal{M}(1^k, \alpha\,;\,\rho)$
Return $\mathbf{m}^*$
procedure Finalize($b$):
Return $b$
```

Figure 5.1: Games for resampling error.

unbounded) adversaries A that play these two games making a single **Challenge** query before outputting a bit. We then define the re-sampling error with respect to $\mathcal{M}$ as

$$\delta_{\mathcal{M}}(k) = \max_{A \in \mathcal{A}} \left\{ \Pr\left[\,\text{RSMPREAL}^{A}_{\mathcal{M},k} \Rightarrow 1\,\right] - \Pr\left[\,\text{RSMPIDEAL}^{A}_{\mathcal{M},k} \Rightarrow 1\,\right] \right\} \ .$$

5.2.2 Simulation-based Security Definitions

We will first consider a simulation-based definition of security for both selective message opening and selective randomness opening. We define security by considering a simulator playing game Id (see Figure 5.2) and an adversary playing

```
procedure Initialize:
    Return $1^k$
procedure Sample($\alpha$):
    $\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
    For $i = 1 \ldots |\mathbf{m}|$ do $\mathbf{z} \leftarrow 1^{|\mathbf{m}[i]|}$
    Return $\mathbf{z}$
procedure Corrupt($I$):
    Return $\mathbf{m}[I]$
procedure Finalize($w$):
    Return $\mathcal{R}(1^k, \mathbf{m}, I, w, \alpha)$
```

Figure 5.2: The identity game $\mathrm{Id}_{\mathcal{M},\mathcal{R},k}$, used to define simulation-based security for both selective message opening and selective randomness opening.

```
procedure Initialize:
    $pars \leftarrow_{\$} \mathcal{P}(1^k)$
    $(pk, sk) \leftarrow_{\$} \mathcal{K}(pars)$
    Return $pars, pk$
procedure Sample($\alpha$):
    $\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
    For $\ell = 1 \ldots |\mathbf{m}|$ do:
        $\mathbf{r}[\ell] \leftarrow_{\$} \mathsf{Coins}_{\mathcal{E}}(pars, pk)$
        $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{m}[\ell] \,;\, \mathbf{r}[\ell])$
    Return $\mathbf{c}$
procedure Corrupt($I$):
    Return $\mathbf{m}[I]$ $\boxed{, \mathbf{r}[I]}$
procedure Finalize($w$):
    Return $\mathcal{R}(1^k, \mathbf{m}, I, w, \alpha)$
```

Figure 5.3: Games SMOSEM (without boxed statements) and SROSEM (with boxed statements) for the simulation-based definitions of selective message and selective randomness opening.

game SMOSEM in the case of selective message opening (Figure 5.3 without boxed code) or game SROSEM for selective randomness opening (Figure 5.3 with boxed code).

Game Id, played with a simulator, is parameterized by a message sampler $\mathcal{M}$, relation $\mathcal{R}$, and security parameter k. It proceeds as follows. The **Initialize** procedure does nothing and returns the security parameter to the simulator. The simulator may then make one query to **Sample** with a string α. The **Sample** procedure runs the message sampler $\mathcal{M}$ on input the security parameter and the string α supplied by the simulator to get a message vector $\mathbf{m}$. The **Sample** procedure then returns the lengths of the sampled messages (but not the actual messages) to the simulator.

At this point, the simulator may corrupt some subset of the messages by making one query to **Corrupt** with a set I of indices to corrupt. The **Corrupt** procedure returns to the simulator the messages $\mathbf{m}[I]$ corresponding to set I.

Finally, the simulator halts with output string w and **Finalize** (and thus the game) outputs the result of applying the relation $\mathcal{R}$ to the security parameter, entire vector of sampled messages $\mathbf{m}$, set of corrupted indices I, output string w, and the string α queried to **Sample**.

Games SROSEM and SMOSEM, played with an adversary, are both parameterized by a PKE scheme $\mathcal{AE}$, message sampler $\mathcal{M}$, relation $\mathcal{R}$, and security parameter k. They give an adversary the same oracles as Id, however each procedure is slightly different. The games proceed as follows. The **Initialize** procedure runs parameter and key generation for $\mathcal{AE}$, returning the results $pars, pk$ to the adversary.

The adversary may then make one query to **Sample** with a string α. Upon such a query, the **Sample** procedure runs the message sampler $\mathcal{M}$ on α to get a vector of messages $\mathbf{m}$. The **Sample** procedure goes on to encrypt each message in the vector with independent and uniform coins and under the public key pk. The resulting ciphertexts are returned to the adversary.

At this point the adversary may make one query to **Corrupt** with a set of indices it wants corrupted. In the game SMOSEM the procedure **Corrupt** returns the corresponding messages $\mathbf{m}[I]$, while in the game SROSEM procedure **Corrupt** returns the messages $\mathbf{m}[I]$ as well as the coins $\mathbf{r}[I]$ used by **Sample**. Lastly, **Finalize** is the same as in Id.

We say that the semsro-advantage of an adversary A against $\mathcal{AE}$, with respect to a message sampler $\mathcal{M}$, a relation $\mathcal{R}$, and a simulator S is

$$\mathbf{Adv}^{\text{sem-sro}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) = \Pr\left[\,\text{SROSEM}^{A}_{\mathcal{AE},\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] - \Pr\left[\,\text{Id}^{S}_{\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] \ .$$

Similarly, the semsmo-advantage of an adversary A against $\mathcal{AE}$, with respect to a

```
procedure Initialize:
$b \leftarrow_{\$} \{0,1\}$
$pars \leftarrow_{\$} \mathcal{P}(1^k)$
$(pk, sk) \leftarrow_{\$} \mathcal{K}(pars)$
Return $pars, pk$

procedure Sample($\alpha$):
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
For $\ell = 1 \ldots |\mathbf{m}|$ do:
    $\mathbf{r}[\ell] \leftarrow_{\$} \mathsf{Coins}_{\mathcal{E}}(pars, pk)$
    $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{m}[\ell] \, ; \, \mathbf{r}[\ell])$
Return $\mathbf{c}$

procedure Corrupt($I$):
Return $\mathbf{m}[I]$ $\boxed{, \mathbf{r}[I]}$

procedure Challenge():
$\mathbf{m}_0 \leftarrow \mathbf{m}$
$\mathbf{m}_1 \leftarrow_{\$} \mathcal{M}(1^k, \alpha, I, \mathbf{m}[I])$
Return $\mathbf{m}_b$

procedure Finalize($b'$):
Return $(b = b')$
```

Figure 5.4: Games SMOIND (without boxed statements) and SROIND (including boxed statements) for the indistinguishability-based definitions of selective opening.

message sampler $\mathcal{M}$, a relation $\mathcal{R}$, and a simulator S is

$$\mathbf{Adv}^{\text{sem-smo}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) = \Pr\left[\, \text{SMOSEM}^{A}_{\mathcal{AE},\mathcal{M},\mathcal{R},k} \Rightarrow 1 \,\right] - \Pr\left[\, \text{Id}^{S}_{\mathcal{M},\mathcal{R},k} \Rightarrow 1 \,\right] \ .$$

Then, we say that a PKE scheme $\mathcal{AE}$ is SEM-SRO-secure (resp. SEM-SMO-secure) if for every positive integer polynomial n, every efficient n-message sampler $\mathcal{M}$, every efficiently computable relation $\mathcal{R}$, and every efficient adversary A, there exists an efficient simulator S such that the semsro-advantage (resp. semsmo-advantage) of A with respect to $\mathcal{M}$, $\mathcal{R}$, and S is negligible in the security parameter. It is easy to see that if a scheme is SEM-SRO secure then it is also SEM-SMO secure.

5.2.3 Indistinguishability-based Security Definitions

We also consider an indistinguishability-based security definition for both selective randomness and selective message opening. To do so we will use games SROIND and SMOIND, both shown in Figure 5.4. Games SROIND and SMOIND, played with an adversary, and parameterized by a PKE scheme $\mathcal{AE}$, resamplable message sampler $\mathcal{M}$, and security parameter k, proceed as follows.

The **Initialize** procedure runs parameter generation and key generation for

$\mathcal{AE}$ as well as flips a bit b. It returns the parameters and public key $\mathit{pars}, \mathit{pk}$ to the adversary.

The adversary may then make one query to **Sample** with a string α. Upon such a query, the **Sample** procedure runs the message sampler $\mathcal{M}$ on α to get a vector of messages $\mathbf{m}$. The **Sample** procedure goes on to encrypt each message in the vector with independent and uniform coins and under the public key pk. The resulting ciphertexts are returned to the adversary.

At this point the adversary may make one query to **Corrupt** with a set of indices it wants corrupted. In the game SMOIND the procedure **Corrupt** returns the corresponding messages $\mathbf{m}[I]$, while in the game SROIND procedure **Corrupt** returns the messages $\mathbf{m}[I]$ as well as the coins $\mathbf{r}[I]$ used by **Sample**.

The adversary may then make one query to **Challenge** with no inputs. Depending on the challenge bit b, **Challenge** either returns the entire actual vector $\mathbf{m}$ used by **Sample**, or a resampled vector, where the resample mode of $\mathcal{M}$ is used for the resampling.

The procedure **Finalize** then takes a guess bit b' from the adversary and outputs true if the guess bit is correct and false otherwise.

We say the indsro-advantage of an adversary A with respect to $\mathcal{AE}$ and a resamplable message sampler $\mathcal{M}$ is

$$\mathbf{Adv}^{\text{ind-sro}}_{A,\mathcal{AE},\mathcal{M}}(k) = 2 \cdot \Pr\left[\, \mathrm{SROIND}^{A}_{\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true} \,\right] - 1 \ .$$

Similarly, we say the indsmo-advantage of an adversary A with respect to $\mathcal{AE}$ and a resamplable message sampler $\mathcal{M}$ is

$$\mathbf{Adv}^{\text{ind-smo}}_{A,\mathcal{AE},\mathcal{M}}(k) = 2 \cdot \Pr\left[\, \mathrm{SMOIND}^{A}_{\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true} \,\right] - 1 \ .$$

Finally, we say that a PKE scheme $\mathcal{AE}$ is IND-SRO (resp. IND-SMO) secure if for every efficient adversary A, for every positive integer polynomial n, and every efficiently resamplable n-message sampler $\mathcal{M}$ with negligible resampling error, the indsro-advantage (resp. indsmo-advantage) of A with respect to $\mathcal{AE}$ and $\mathcal{M}$ is negligible in the security parameter k.

It is easy to see that if a scheme is IND-SRO secure then it is also IND-SMO secure.

5.3 Equivalence of IND-CPA and Selective Message Opening

In this section we show that IND-CPA is equivalent to security under selective message opening. To do so, we first show that IND-CPA implies SEM-SMO security. The proof of this theorem will be useful for understanding our results on SEM-SRO security later. We then show that IND-SMO security implies IND-CPA. The proof ideas used to show the above results can be easily translated to show IND-CPA implies IND-SMO security and SEM-SMO implies IND-CPA, giving us equivalence. Alternatively, we can get equivalence by relying on a result of [19] showing SEM-SRO implies IND-SRO when considering efficiently resamplable message distributions. (This result easily applies to the SMO setting.)

5.3.1 From IND-CPA to SMO

First, we show that any encryption scheme that is IND-CPA secure is also secure under selective message opening. The proof of this result serves as a good warm-up for our main results on selective randomness opening.

Theorem 5.3.1 [IND-CPA implies SEM-SMO] Let k be a security parameter, $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be a public-key encryption scheme, n a positive polynomial taking integer values, $\mathcal{M}$ an efficient n-message sampler, $\mathcal{R}$ an efficiently computable relation, and A be an efficient adversary. Then, there exists an efficient simulator S and an efficient INDCPA adversary C such that

$$\mathbf{Adv}^{\text{sem-smo}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) \leq n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{C,\mathcal{AE}}(k) \; .$$

Adversary C makes one **LR** query. $\square$

```
procedure Sample($\alpha$):                                   $G_1$
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
For $\ell = 1 \ldots |\mathbf{m}|$ do:
    $\mathbf{r}[\ell] \leftarrow_{\$} \mathsf{Coins}_{\mathcal{E}}(pars, pk)$
    $\mathbf{z}[\ell] \leftarrow 0^{|\mathbf{m}[\ell]|}$
    $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{z}[\ell] \; ; \; \mathbf{r}[\ell])$
Return $\mathbf{c}$
```

Figure 5.5: Game for proof of Theorem 5.3.1.

Before proving the theorem we briefly sketch the main idea. We need to design a simulator S that can mimic the output of an adversary A. The obvious thing to do is for S to run A itself and simulate its environment properly. It is unclear how S can simulate the environment *perfectly*, since the adversary expects **Sample** to return encryptions of an entire vector of sampled messages, while S can only learn message *lengths* from querying its own **Sample** oracle. The solution is for S to simply give A encryptions of dummy messages (e.g., all 0s) of the appropriate length. When A makes a **Corrupt** query, S can make the same **Corrupt** query and finally learn the actual messages it needs to open the ciphertexts to. At this point S simply returns the actual messages. The point is that neither A nor any other efficient adversary can tell that it was given encryptions of dummy messages instead of actual messages due to the IND-CPA security of the encryption scheme. We now give the details.

Proof: We will prove the theorem using a sequence of game transitions, starting with a game G_0, which is the same as game SMOSEM. Game G_1 differs from G_0 only in the **Sample** procedure; the modified procedure is shown in Figre 5.5.

First, recall that

$$\mathbf{Adv}^{\text{sem-smo}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) = \Pr\left[\,\text{SMOSEM}^{A}_{\mathcal{AE},\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] - \Pr\left[\,\text{Id}^{S}_{\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] \; . \quad (5.1)$$

We let game G_0 be identical to game $\text{SMOSEM}_{\mathcal{AE},\mathcal{M},\mathcal{R},k}$. Our first game transition to G_1 makes only one change: in **Sample**, instead of encrypting the actual message

```
Adversary $B(pars, pk)$:
Run $A(pars, pk)$.
  On query $\mathbf{Sample}(\alpha)$:
  $\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
  For $\ell = 1 \ldots |\mathbf{m}|$ do:
      $\mathbf{z}[\ell] \leftarrow 0^{|\mathbf{m}[\ell]|}$
      $\mathbf{c}[\ell] \leftarrow \mathbf{LR}_B(\mathbf{m}[\ell], \mathbf{z}[\ell])$
  Return $\mathbf{c}$
  On query $\mathbf{Corrupt}(I)$:
  Return $\mathbf{m}[I]$
When $A$ halts with output $w$,
halt and output $\mathcal{R}(1^k, \mathbf{m}, I, w, \alpha)$
```

```
Simulator $S(1^k)$:
$pars \leftarrow_{\$} \mathcal{P}(1^k)$
$(pk, sk) \leftarrow_{\$} \mathcal{K}(pars)$
Run $A(pars, pk)$.
  On query $\mathbf{Sample}(\alpha)$:
  $\mathbf{z} \leftarrow \mathbf{Sample}_S(i, \alpha)$
  For $\ell = 1 \ldots |\mathbf{z}|$ do:
      $\mathbf{r}[\ell] \leftarrow_{\$} \mathsf{Coins}_{\mathcal{E}}(pars, pk)$
      $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{z}[\ell] \,;\, \mathbf{r}[\ell])$
  Return $\mathbf{c}$
  On query $\mathbf{Corrupt}(I)$:
  $\mathbf{m}_I \leftarrow \mathbf{Corrupt}_S(j)$
  Return $\mathbf{m}_I$
When $A$ halts with output $w$,
halt and output $w$
```

Figure 5.6: Adversary and simulator used in the proof of Theorem 5.3.1

sampled by $\mathcal{M}$, encrypt a dummy message of the appropriate length (we've used all 0s, but any distinguished message in the message space would work just as well). We claim that there is an efficient adversary B such that

$$\Pr\left[\, G_0^A \Rightarrow 1 \,\right] - \Pr\left[\, G_1^A \Rightarrow 1 \,\right] \leq \mathbf{Adv}^{\text{ind-cpa}}_{B,\mathcal{AE}}(k) \;, \tag{5.2}$$

where adversary B, shown in Figure 5.6, makes n queries to **LR**. Applying Proposition 2.4.1, there is an efficient adversary C such that

$$\Pr\left[\, G_0^A \Rightarrow 1 \,\right] - \Pr\left[\, G_1^A \Rightarrow 1 \,\right] \leq n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{C,\mathcal{AE}}(k) \;, \tag{5.3}$$

where C makes one **LR** query.

Finally, we note that running game G_1 with adversary A is the same as running game Id with simulator S, shown in Figure 5.6. Thus,

$$\Pr\left[\, G_1^A \Rightarrow 1 \,\right] = \Pr\left[\, \mathrm{Id}^S_{\mathcal{M},\mathcal{R},k} \Rightarrow 1 \,\right] \;. \tag{5.4}$$

This is because S only needs the message lengths in **Sample** (which it can get from its own **Sample** oracle) and actual sampled messages are not needed until **Corrupt**, where they can be learned by S by querying its own **Corrupt** oracle.

Combining the above equations, we can see that

$$\begin{aligned}
\mathbf{Adv}^{\text{sem-smo}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) &= \Pr\left[\,\text{SMOSEM}^{A}_{\mathcal{AE},\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] - \Pr\left[\,\text{Id}^{S}_{\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] \\
&= \Pr\left[\,G_0^A \Rightarrow 1\,\right] - \Pr\left[\,G_1^A \Rightarrow 1\,\right] \\
&\leq n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{C,\mathcal{AE}}(k) \ ,
\end{aligned}$$

which proves the theorem. ▌

One might wonder why this same strategy does not work for proving selective randomness opening security. The reason is that in the SRO setting the simulator needs to provide messages *and coins* when the adversary makes a **Corrupt** query. Given a message m and coins r, the adversary can recompute encryption (it knows the public key) and in some sense verify the encryption it was given by **Sample** actually is an encryption of m. Thus, the simulator cannot give the adversary the encryption of a dummy message and then later lie to the adversary and claim it was actually an encryption of a different message.

5.3.2 From SMO to IND-CPA

We just showed that IND-CPA implies SEM-SMO. Next, we will show that IND-SMO implies IND-CPA.

Theorem 5.3.2 [IND-SMO implies IND-CPA] Let k be a security parameter, $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be a public-key encryption scheme, and A an efficient INDCPA adversary against $\mathcal{AE}$. Then there exists an efficient adversary B, a positive integer polynomial n and an efficiently resamplable n-message sampler $\mathcal{M}$ such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{A,\mathcal{AE}}(k) \leq 2 \cdot \mathbf{Adv}^{\text{ind-smo}}_{B,\mathcal{AE},\mathcal{M}}(k) \ .$$

□

```
Adversary $B(pars, pk)$:
Run $A(pars, pk)$.
    On query $\mathbf{LR}(m_0, m_1)$:
    $\alpha \leftarrow (m_0, m_1)$
    $\mathbf{c} \leftarrow \mathbf{Sample}_B(\alpha)$
    Return $\mathbf{c}[1]$
When $A$ halts with output $b'$:
$m \leftarrow \mathbf{Challenge}_B()$
If $m = m_{b'}$ return 0, else return 1
```

Figure 5.7: Adversary used in proof of Theorem 5.3.2.

Proof: We first define a 1-message sampler $\mathcal{M}(1^k, \alpha)$ to parse α as a pair of message (m_0, m_1), flip a bit d, and return message vector (m_d). It is easy to see $\mathcal{M}$ runs in polynomial time and is also efficiently resamplable. We define adversary B in Figure 5.7. The adversary B runs A and uses **Sample** to simulate the **LR** oracle for A.

Recall that by definition

$$\mathbf{Adv}^{\text{ind-cpa}}_{A,\mathcal{AE}}(k) = 2 \cdot \Pr\left[\,\text{INDCPA}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true}\,\right] - 1 \,, \tag{5.5}$$

and

$$\mathbf{Adv}^{\text{ind-smo}}_{B,\mathcal{AE},\mathcal{M}}(k) = 2 \cdot \Pr\left[\,\text{SMOIND}^{B}_{\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right] - 1 \,. \tag{5.6}$$

Now, let's look at the probability $\Pr\left[\,\text{SMOIND}^{B}_{\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right]$ in more detail. Let $\text{SMOIND}_{b,\mathcal{AE},\mathcal{M},k}$ be the same game, but parameterized by the challenge bit b. (In other words, the game sets b to that value in **Initialize** instead of choosing it uniformly at random.) Then

$$\Pr\left[\,\text{SMOIND}^{B}_{\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right] = \\ \frac{1}{2} \cdot \Pr\left[\,\text{SMOIND}^{B}_{0,\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right] + \frac{1}{2} \cdot \Pr\left[\,\text{SMOIND}^{B}_{1,\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right] \,.$$

We first claim that

$$\Pr\left[\,\mathrm{SMOIND}^{B}_{0,\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right] = \Pr\left[\,\mathrm{INDCPA}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true}\,\right]\ . \tag{5.7}$$

This is true since if the challenge bit in SMOIND is 0 then the actual encrypted message is returned by **Challenge**. Adversary B will correctly guess 0 only if adversary A guessed the bit of the encrypted message correctly, meaning it won the IND-CPA game.

We next claim

$$\Pr\left[\,\mathrm{SMOIND}^{B}_{1,\mathcal{AE},\mathcal{M},k} \Rightarrow \mathsf{true}\,\right] = \frac{1}{2}\ . \tag{5.8}$$

This is true since if the challenge bit in SMOIND is 1 the resampled message is returned, and this value will just be uniformly chosen between the two options. Yet, this uniform choice will be independent of anything given to A, so A has a 1/2 probability of guessing correctly.

Combining the above equations, we see that

$$\begin{aligned}
&\mathbf{Adv}^{\text{ind-smo}}_{B,\mathcal{AE},\mathcal{M}}(k)\\
&\quad= 2\cdot\left(\frac{1}{2}\cdot\Pr\left[\,\mathrm{SMOIND}^{B}_{0} \Rightarrow \mathsf{true}\,\right] + \frac{1}{2}\cdot\Pr\left[\,\mathrm{SMOIND}^{B}_{1} \Rightarrow \mathsf{true}\,\right]\right) - 1\\
&\quad= \Pr\left[\,\mathrm{INDCPA}^{A}_{\mathcal{AE},k} \Rightarrow \mathsf{true}\,\right] + \frac{1}{2} - 1\\
&\quad= \frac{1}{2}\mathbf{Adv}^{\text{ind-cpa}}_{A,\mathcal{AE}}(k)\ ,
\end{aligned}$$

which proves the theorem. ▮

5.4 Lossy Encryption

Our main results on selective randomness opening rely on what we call a *Lossy Encryption Scheme.* Informally, a public-key encryption scheme is lossy if there is an efficient alternate key generation algorithm (called the "lossy key generation algorithm") that produces "lossy" public keys that are computationally indistinguishable from "real" public keys; however, encryptions under lossy public keys statistically contain little or no information about the encrypted message. Peikert, Vaikuntanathan, and Waters [57] called such lossy keys "messy keys", for *me*ssage lo*ssy*, while defining a related notion called Dual-Mode Encryption. The notion of Lossy Encryption is also similar to Meaningful/Meaningless Encryption [50], formalized by Kol and Naor.

Now we can formally define lossy encryption. We say a public-key encryption scheme $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is *lossy* if there exists an efficient alternate key generation algorithm $\mathcal{K}_\ell$ such that

1. *Indistinguishability of Real and Lossy Keys.* For all efficient adversaries A, the key-ind advantage $\mathbf{Adv}^{\text{key-ind}}_{A,\mathcal{AE},\mathcal{K}_\ell}(k)$ is negligible.

2. *Lossiness of encryption with lossy keys.* For hiding scheme $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}_\ell, \mathcal{E})$ induced by $\mathcal{AE}$ and $\mathcal{K}_\ell$, and for all (even unbounded) adversaries B, the ind-cpa-advantage $\mathbf{Adv}^{\text{ind-cpa}}_{B,\mathsf{Hid}}(k)$ is negligible.

The first property informally says that honestly generated public keys are computationally indistinguishable from lossy public keys, generated by an alternate key generation algorithm. The second property then informally says that encryption under these indistinguishable lossy keys statistically loses essentially all information about the message. This means that, under a lossy public key, a message is no longer uniquely determined by its encryption. Thus, a ciphertext can be opened to more than one message.

We say that an algorithm Open_∞ is an opening algorithm for hiding scheme $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}_\ell, \mathcal{E})$ if for all k, all $pars \in [\mathcal{P}(1^k)]$, all $pk_\ell \in [\mathcal{K}_\ell(pars)]_1$, all m', and all ciphertexts c, the output of $\mathsf{Open}_\infty(pk_\ell, m', c)$ is distributed uniformly in the set $\{r' \mid r' \in \mathsf{Coins}_{\mathcal{E}}(pars, pk_\ell) \wedge \mathcal{E}(pk_\ell, m'; r') = c\}$. This set may be empty, in which case Open_∞ should output $\perp$. Note that an opening algorithm always exists, but may not be polynomial time. From here on out, we let Open_∞ be the unbounded algorithm that finds all r' such that $\mathcal{E}(pk_\ell, m'; r') = c$ and then outputs a uniformly chosen one of the r'.

We also consider efficient opening algorithms that take additional inputs. Specifically, we say Open is an efficient opening algorithm for hiding scheme $\mathsf{Hid} =$

```
procedure Initialize:                    Game OPENREAL_{Hid,Open,k}
Return $1^k$
procedure Challenge($pars, pk, sk, m', m, r, c$):
If ($\mathcal{E}(pars, pk, m\ ;\ r) \neq c$) then return $\perp$
$r^* \leftarrow_{\$} \mathsf{Open}(pars, pk, sk, m', m, r, c)$
Return $r^*$
procedure Finalize($b$):
Return $b$
```

```
procedure Initialize:                    Game OPENIDEAL_{Hid,Open,k}
Return $1^k$
procedure Challenge($pars, pk, sk, m', m, r, c$):
If ($\mathcal{E}(pars, pk, m\ ;\ r) \neq c$) then return $\perp$
$S \leftarrow \emptyset$
For $r' \in \mathsf{Coins}_{\mathcal{E}}(pars, pk)$ do
   If $\mathcal{E}(pars, pk, m'\ ;\ r') = c$ then
      $S \leftarrow S \cup \{r'\}$
$r^* \leftarrow_{\$} S$
Return $r^*$
procedure Finalize($b$):
Return $b$
```

Figure 5.8: Games for Opening Error.

$(\mathcal{P}, \mathcal{K}_\ell, \mathcal{E})$ if for all k, all $pars \in [\mathcal{P}(1^k]$, all $(pk_\ell, sk_\ell) \in [\mathcal{K}_\ell(pars)]$, all messages m and m', all r and all $c = \mathcal{E}(pk_\ell, m\ ;\ r)$, the output of $\mathsf{Open}(pk_\ell, sk_\ell, m', m, r, c)$ is distributed uniformly in the set $\{r' \mid r' \in \mathsf{Coins}_{\mathcal{E}}(pars, pk_\ell) \wedge \mathcal{E}(pk_\ell, m'; r') = c\}$. We formalize this more formally by considering games OPENREAL and OPENIDEAL shown in Figure 5.8. Notice procedure **Challenge** in game OPENIDEAL simply performs the same steps as the canonical unbounded opening algorithm Open_∞. We then say Open is an efficient opening algorithm for hiding scheme Hid if for all even unbounded adversaries A and all k it is the case that

$$\Pr\left[\,\mathrm{OPENREAL}^{A}_{\mathsf{Hid},\mathsf{Open},k} \Rightarrow 1\,\right] = \Pr\left[\,\mathrm{OPENIDEAL}^{A}_{\mathsf{Hid},\mathsf{Open},k} \Rightarrow 1\,\right]\ .$$

```
procedure Initialize:                                   Game DDH_{GpGen,k}
$(\mathbb{G}, g, p) \leftarrow_\$ \mathsf{GpGen}(1^k)$
$r, x \leftarrow_\$ \mathbb{Z}_p$ ; $y \leftarrow_\$ \mathbb{Z}_p \setminus \{x\}$
$h \leftarrow g^r$ ; $g' \leftarrow g^x$ ; $d \leftarrow_\$ \{0,1\}$
If $d = 0$ then $h' \leftarrow g^{rx}$ else $h' \leftarrow g^{ry}$
Return $(p, g, h, g', h')$

procedure Finalize($d'$):
Return $(d = d')$
```

Figure 5.9: Decisional Diffie-Hellman (DDH) security game.

Finally, before looking at specific constructions, we note that it immediately follows that any lossy encryption scheme is IND-CPA secure. We next provide numerous examples of lossy encryption schemes.

5.4.1 Lossy Encryption from DDH

We first describe a lossy public-key encryption scheme based on the DDH assumption. A group generator is an algorithm GpGen that, on input a security parameter 1^k in unary, selects a cyclic group $\mathbb{G}$ of order a prime p and a generator g, and outputs a description of the group $\mathbb{G}$ as well as g and p. Now consider a game DDH (shown in Figure 5.9) played with an adversary. We define the ddh-advantage of an adversary A against a group generator GpGen as

$$\mathbf{Adv}^{\mathrm{ddh}}_{A,\mathsf{GpGen}}(k) = 2 \cdot \Pr\left[\, \mathrm{DDH}^{A}_{\mathsf{GpGen},k} \Rightarrow \mathsf{true} \,\right] - 1 \, .$$

The DDH assumption for a group generator GpGen states that for all efficient adversaries A the ddh-advantage of A against GpGen is a negligible function of the security parameter.

We can now describe our scheme. The scheme is originally from [54], yet some of our notation is taken from [57]. The scheme has structure similar to El Gamal encryption [38].

The scheme $\mathcal{AE}_{\mathrm{ddh1}} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ and the associated lossy key generation algorithm $\mathcal{K}_\ell$ constructed from a group generation algorithm GpGen are as follows. Parameter generation algorithm $\mathcal{P}$, on input 1^k, runs GpGen on input 1^k and outputs the description of the group $\mathbb{G}$, prime p, and generator g. The message space of the scheme is the group, i.e., $\mathsf{MsgSpace}(\mathbb{G}, p, g) = \mathbb{G}$. The rest of the algorithms:

```
Algorithm $\mathcal{K}(pars)$
    $(\mathbb{G}, g, p) \leftarrow pars$
    $x, r \leftarrow_{\$} \mathbb{Z}_p$
    $pk \leftarrow (g, g^r, g^x, g^{rx})$
    $sk \leftarrow x$
    Return $(pk, sk)$

Algorithm $\mathcal{E}(pk, m)$
    $(g, h, g', h') \leftarrow pk$
    $(u, v) \leftarrow_{\$} \mathsf{Rand}(g, h, g', h')$
    Return $(u, v \cdot m)$

Algorithm $\mathcal{D}(sk, c)$
    $(c_0, c_1) \leftarrow c$
    Return $c_1 / c_0^{sk}$
```

```
Algorithm $\mathcal{K}_\ell(pars)$
    $(g, p) \leftarrow pars$
    $r, x \leftarrow_{\$} \mathbb{Z}_p$ ; $y \leftarrow_{\$} \mathbb{Z}_p \setminus \{x\}$
    $pk \leftarrow (g, g^r, g^x, g^{ry})$
    $sk \leftarrow (r, x, y)$
    Return $(pk, sk)$

Subroutine $\mathsf{Rand}(g, h, g', h')$
    $s, t \leftarrow_{\$} \mathbb{Z}_p$
    $u \leftarrow g^s h^t$; $v \leftarrow (g')^s (h')^t$
    Return $(u, v)$
```

To see the correctness property is satisfied, consider a (real) public key $pk = (g, g^r, g^x, g^{rx})$ and corresponding secret key $sk = x$. Then, for any message $m \in \mathbb{G}$

$$\begin{aligned} \mathcal{D}(sk, \mathcal{E}(pk, m)) &= \mathcal{D}(sk, (g^{s+rt}, g^{xs+rxt} \cdot m)) \\ &= (g^{xs+rxt} \cdot m)/(g^{s+rt})^x \\ &= m \end{aligned}$$

Now, we claim that $\mathcal{AE}_{\mathrm{ddh1}}$ is a lossy encryption scheme.

1. *Indistinguishability of Real Keys and Lossy Keys.* This follows from the assumption that DDH is hard for GpGen, since the first output of $\mathcal{K}$ is (g, g^r, g^x, g^{rx}) and the first output of $\mathcal{K}_\ell$ is (g, g^r, g^x, g^{ry}) for $y \neq x$, which exactly matches the case in game DDH.

2. *Lossiness of encryption with lossy keys.* Since a lossy key (g, g^r, g^x, g^y) is such that $rx \neq y$, then $s + rt$ and $sx + yt$ are linearly independent combinations of s and t, so an encryption under a lossy key results in two uniformly random group elements.

We mention that we do not know an efficient opening algorithm for this scheme. However, we will see later in this section that if we instead encrypt bit-by-bit then we can in fact find an efficient opening algorithm.

5.4.2 Lossy Encryption from Lossy TDFs

We now describe an instantiation of lossy (randomized) encryption based on lossy trapdoor functions. The following scheme was given by Peikert and Waters in [58] Let k be a security parameter and let $\mathcal{F} = (\mathsf{P}, \mathsf{K}, \mathsf{F}, \mathsf{F}^{-1})$ define a collection of injective trapdoor functions that are (n, L)-lossy with associated lossy key generation algorithm K_ℓ. Also let $\mathcal{H} = (\mathsf{P}_\mathrm{h}, \mathsf{K}_\mathrm{h}, \mathsf{H})$ be a collection of pair-wise independent hash functions with message length $n(k)$ and output length $\ell(k)$; the message space of the cryptosystem will be $\{0,1\}^\ell$. The parameter ℓ should be such that $\ell \le L - 2\log(1/\delta)$, where δ is a negligible function in the security parameter k. The scheme $\mathcal{AE}_\mathrm{ltdf} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is then defined as follows. The parameter generation algorithm $\mathcal{P}$, on input security parameter 1^k, simply runs $\pi \leftarrow_{\$} \mathsf{P}(1^k)$ and $\pi_\mathrm{h} \leftarrow_{\$} \mathsf{P}_\mathrm{h}(1^k)$ and outputs $pars = (\pi, \pi_\mathrm{h})$. The rest of the algorithms are as follows:

```
Algorithm $\mathcal{K}(pars)$
$(\pi, \pi_\mathrm{h}) \leftarrow pars$
$(\sigma, \tau) \leftarrow_{\$} \mathsf{K}(\pi)$
$\kappa \leftarrow_{\$} \mathsf{K}_\mathrm{h}(\pi_\mathrm{h})$
$pk \leftarrow (\sigma, \kappa); sk \leftarrow (\tau, \kappa)$
Return $(pk, sk)$
```

```
Algorithm $\mathcal{E}(pk, m)$
$(\sigma, \kappa) \leftarrow pk$
$x \leftarrow_{\$} \{0,1\}^n$
$c_1 \leftarrow \mathsf{F}(\sigma, x)$
$c_2 \leftarrow m \oplus \mathsf{H}(\kappa, x)$
Return $(c_1, c_2)$
```

```
Algorithm $\mathcal{D}(sk, c)$
$(\tau, \kappa) \leftarrow sk$
$(c_1, c_2) \leftarrow c$
$x \leftarrow \mathsf{F}^{-1}(\tau, c_1)$
Return $\mathsf{H}(\kappa, x) \oplus c_2$
```

The associated lossy key generation algorithm $\mathcal{K}_\ell$ is simply the same as $\mathcal{K}$, but using K_ℓ instead of K. The correctness of the scheme follows since when $pk = (\sigma, \kappa)$ was generated by $\mathcal{K}$,

$$\begin{aligned} \mathcal{D}(sk, \mathcal{E}(pk, m)) &= \mathsf{H}(\kappa, \mathsf{F}^{-1}(\tau, \mathsf{F}(\sigma, x)) \oplus (m \oplus \mathsf{H}(\kappa, x)) \\ &= \mathsf{H}(\kappa, x) \oplus m \oplus \mathsf{H}(\kappa, x) \\ &= m \end{aligned}$$

We can then show the encryption scheme is lossy:

```
procedure Initialize:                                Game DQR$_{\mathsf{Par},k}$
$(N, p, q) \leftarrow_{\$} \mathsf{Par}(1^k)$
$d \leftarrow_{\$} \{0, 1\}$                                  procedure Finalize($d'$):
$x_0 \leftarrow_{\$} \mathsf{QR}_N$ ; $x_1 \leftarrow_{\$} \mathsf{QNR}_N^{+1}$          Return $(d = d')$
Return $(N, x_d)$
```

Figure 5.10: Quadratic Residuosity Game.

1. *Indistinguishability of real keys and lossy keys.* We need to show that any efficient adversary A, the key-ind advantage $\mathbf{Adv}^{\text{key-ind}}_{A,\mathcal{AE}_{\text{ltdf}},\mathcal{K}_\ell}(k)$ is negligible. This follows since $\mathcal{K}$ uses K while $\mathcal{K}_\ell$ uses K_ℓ, and since $\mathcal{F}$ is lossy, we know that for all efficient adversaries B, the key-ind advantage $\mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{F},\mathsf{K}_\ell}(k)$ is negligible.

2. *Lossiness of encryption with lossy keys.* This was shown by Peikert and Waters in [58]. Reviewing their argument, it is true because the average conditional min-entropy $\tilde{\mathrm{H}}_\infty(x|(c_1, pk_\ell))$ of the random variable x, given $\mathsf{F}(\sigma_\ell, x)$ and $pk_\ell = (\sigma_\ell, \kappa)$ is at least L, and since $\ell \leq L - 2\log(1/\delta)$ for negligible δ, it follows that $\mathsf{H}(\kappa, x)$ will be statistically close to uniform and $m_b \oplus \mathsf{H}(\kappa, x)$ will also be statistically close to uniform for either bit b. Thus, even unbounded adversaries have negligible advantage in the INDCPA game.

5.4.3 The GM Probabilistic Encryption Scheme is Lossy with Efficient Opening

The Goldwasser-Micali probabilistic encryption scheme [41] is one example of a lossy encryption scheme with efficient opening. We briefly recall the GM scheme. Let Par be an algorithm that on input 1^k efficiently chooses two large distinct random primes p and q congruent to 3 (mod 4) and outputs them along with their product N (a Blum integer). Let $\mathcal{J}_p(x)$ denote the Jacobi symbol of x modulo p. We denote by QR_N the group of quadratic residues modulo N and we denote by QNR_N^{+1} the group of quadratic non-residues x such that $\mathcal{J}_N(x) = +1$. Recall that the security of the GM scheme is based on the Quadratic Residuosity

Assumption, which states that it is difficult to distinguish a random element of QR_N from a random element of QNR_N^{+1}. This is captured by game DQR, shown in Figure 5.10 and a dqr-advantage of an adversary A with respect to Par

$$\mathbf{Adv}^{\mathrm{dqr}}_{A,\mathsf{Par}}(k) = 2 \cdot \Pr\left[\, \mathrm{DQR}^{A}_{\mathsf{Par},k} \Rightarrow \mathsf{true} \,\right] - 1 \,.$$

The quadratic residuosity assumption for Par is that for all efficient adversaries, the dqr-advantage of A with respect to Par is negligible in k.

The scheme $\mathcal{AE}_{\mathrm{GM}} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is then defined as follows. The parameter generation algorithm $\mathcal{P}$, on input 1^k, simply outputs 1^k. The other algorithms:

```
Algorithm $\mathcal{K}(1^k)$
$(N,p,q) \leftarrow_{\$} \mathsf{Par}(1^k)$
$x \leftarrow_{\$} \mathsf{QNR}_N^{+1}$
$pk \leftarrow (N,x)$
$sk \leftarrow (p,q)$
Return $(pk, sk)$
```

```
Algorithm $\mathcal{E}(pk, m)$
$(N,x) \leftarrow pk$
For $i = 1$ to $|m|$
    $r_i \leftarrow_{\$} \mathbb{Z}_N^*$
    $\mathbf{c}[i] \leftarrow r_i^2 \cdot x^{m_i} \bmod N$
Return $\mathbf{c}$
```

```
Algorithm $\mathcal{D}(sk, \mathbf{c})$
$(p,q) \leftarrow sk$
For $i = 1$ to $|\mathbf{c}|$
    If $\mathcal{J}_p(\mathbf{c}[i]) = \mathcal{J}_q(\mathbf{c}[i]) = +1$
        $m_i \leftarrow 0$
    Else $m_i \leftarrow 1$
Return $m$
```

The associated lossy key generation algorithm $\mathcal{K}_\ell$ is the same as $\mathcal{K}$ except that x is chosen at random from QR_N instead of QNR_N^{+1}; the lossy secret key is still the factorization of N.

The correctness of the above scheme was shown in [41], while the indistinguishability of real keys from lossy keys follows directly from the quadratic residuosity assumption. It is also clear that encryptions under lossy keys are (perfectly) indstinguishable since lossy ciphertexts are just sequences of random quadratic residues.

Now, we claim that $\mathcal{AE}_{\mathrm{GM}}$ is also efficiently openable. To see this consider the (efficient) algorithm Open that takes as input lossy secret key $sk = (p,q)$, lossy public key $pk = (N,x)$, plaintext m', ciphertext $\mathbf{c}$, and m and $\mathbf{r}$ such that $\mathcal{E}(pk, m; \mathbf{r}) = \mathbf{c}$. Say m' has length n bits. For each $i \in [n]$, Open uses p and q to efficiently compute the four square roots of $\mathbf{c}[i]/x^{m'_i}$ and lets $\mathbf{r}'[i]$ be a randomly chosen one of the four. The output of Open is the sequence $\mathbf{r}'$. Notice this is exactly what the inefficient Open_∞ would do (find the four square roots and output a random one of the four); the efficiency is gained by knowing the factorization of N.

5.4.4 A Scheme with Efficient Opening from DDH

We point out that if we modify the scheme in Section 5.4.1 to only encrypt one bit at a time, we can get an encryption scheme $\mathcal{AE}_{\mathrm{ddh2}}$ that is lossy with efficient opening. More formally, modify the encryption algorithm to be as follows:

```
Algorithm $\mathcal{E}(pk, m)$
$(g, h, g', h') \leftarrow pk$
For $i = 1$ to $|m|$
    $(u, v) \leftarrow_{\$} \mathsf{Rand}(g, h, g', h')$
    $(\mathbf{c}_1[i], \mathbf{c}_2[i]) \leftarrow (u, v \cdot g^{m_i})$
Return $(\mathbf{c}_1, \mathbf{c}_2)$
```

In the above, $|m|$ is the bit length of m and m_i is the ith bit. The decryption algorithm is modified in the obvious way, while parameter generation, key generation, Rand, and the corresponding lossy key generation algorithms are unmodified from Section 5.4.1.

We will describe the efficient opening algorithm for a 1-bit message; the algorithm can be repeated on each individual component of the ciphertext in the multi bit case. The efficient opening algorithm, on input

$$\begin{aligned}
pk_\ell &= (g, g^r, g^x, g^{ry}) \\
sk_\ell &= (r, x, y) \\
(c_1, c_2) &= \left(\, g^s \cdot (g^r)^t \;,\; (g^x)^s \cdot (g^{ry})^t \cdot g^m \,\right) \\
m &\in \{0, 1\} \\
s, t &\in \mathbb{Z}_p \\
m' &\in \{0, 1\} \;,
\end{aligned}$$

needs to find $s', t' \in \mathbb{Z}_p$ such that encrypting m' with s' and t' as randomness will result in ciphertext (c_1, c_2). To do so, the opening algorithm solves the equations

$$\begin{aligned} s + rt &= s' + rt' \\ xs + ryt + m &= xs' + ryt' + m' \ , \end{aligned}$$

which is possible in our scheme since $x \neq y$. Note that there is only one such pair (s', t'), so the output of the efficient opener will be the same as the output would have been from the inefficient opener Open_∞.

5.5 Lossy Encryption implies Selective Randomness Opening Security

We now state our main results for encryption security under selective opening attacks: any lossy public-key encryption scheme is IND-SRO-secure, and any lossy public-key encryption scheme with efficient opening is SEM-SRO-secure. Specifically, we prove the following two theorems.

Theorem 5.5.1 [Lossy Encryption implies IND-SRO security] Let k be a security parameter, $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be any lossy public-key encryption scheme with associated lossy key generation algorithm $\mathcal{K}_\ell$; let $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}_\ell, \mathcal{E})$ be the resulting hiding scheme. Let n be a positive integer polynomial and $\mathcal{M}$ be an efficiently resample n-message sampler with resampling error $\delta_{\mathcal{M}}$ and let A be an efficient adversary. Then, there exists an efficient KEYIND-adversary B, and an unbounded INDCPA-adversary D such that

$$\mathbf{Adv}^{\text{ind-sro}}_{A,\mathcal{AE},\mathcal{M}}(k) \leq 2 \cdot \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + 2 \cdot n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) + 2 \cdot \delta_{\mathcal{M}}(k) \ .$$

Adversary D makes 1 **LR** query. $\square$

Theorem 5.5.2 [Lossy Encryption with efficient opening implies SEM-SRO] Let k be a security parameter, $\mathcal{AE} = (\mathcal{P}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ be a lossy public-key encryption scheme with associated lossy key generation algorithm $\mathcal{K}_\ell$ and efficient opening algorithm Open; let $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}_\ell, \mathcal{E})$ be the resulting hiding scheme. Let n be a positive integer polynomial and $\mathcal{M}$ be an efficient n-message sampler, $\mathcal{R}$ be an efficiently computable relation, and A be an efficient adversary. Then, there exists an efficient simulator S, an efficient KEYIND-adversary B, and an unbounded INDCPA-adversary D such that

$$\mathbf{Adv}^{\text{sem-sro}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) \le \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) \ .$$

Adversary D makes 1 **LR** query. $\square$

Before proving the theorems we discuss how their proofs are similar to our proof that IND-CPA implies security under selective message opening. Recall in that proof the simulator could give the adversary an encryption of all 0s but then later claim that it was an encryption of some other message. In other words, the simulator would give the adversary $c = \mathcal{E}(pk, 00\ldots0\,;\, r)$ but then later claim the encrypted message was m. We explained that in the case of selective randomness opening, this strategy would not work since the simulator also had to provide coins r', and the adversary could then compute whether $c = \mathcal{E}(pk, m; r')$. Unfortunately, this is actually impossible with most standard encryption schemes. The reason is that for schemes with perfect completeness (decryption always outputs the correct encrypted message), the ciphertext space can be partitioned depending on the message. Thus, it is impossible for a simulator to find r' such that $\mathcal{E}(pk, m\,;\, r') = \mathcal{E}(pk, 00\ldots0\,;\, r)$ when $m \neq 00\ldots0$. Informally, this means that encryption is a commitment to the message.

It appears we are stuck. However, note that the above property only has to hold for public keys pk that come from the key generation algorithm $\mathcal{K}$; we will call such keys "real" public keys. The main observation that leads to our results in this section is that there may be an alternate key generation algorithm $\mathcal{K}_\ell$ that outputs public keys pk_ℓ that "look like" real public keys. However, these lossy keys have the property that using them in the encryption algorithm no longer results in a commitment to the message. Schemes that have this property are the lossy

```
procedure Initialize:                                   $G_1, G_2, G_3, G_4$
$pars \leftarrow_{\$} \mathcal{P}(1^k)$ ; $(pk, sk) \leftarrow_{\$} \mathcal{K}_\ell(pars)$
Return $pk$
```

```
procedure Corrupt($I$):                                 $G_2, G_3, G_4$
For $i \in I$ do:
    $\mathbf{r}'[i] \leftarrow_{\$} \mathsf{Open}_\infty(pars, pk, \mathbf{m}[i], \mathbf{c}[i])$
Return $\mathbf{m}[I], \mathbf{r}'[I]$
```

```
procedure Sample($\alpha$):                             $G_3, G_4$
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
For $\ell = 1 \ldots |\mathbf{m}|$ do:
    $\mathbf{r}[\ell] \leftarrow_{\$} \mathsf{Coins}_{\mathcal{E}}(pars, pk)$ ; $\mathbf{z}[\ell] \leftarrow 0^{|\mathbf{m}[\ell]|}$
    $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{z}[\ell] ; \mathbf{r}[\ell])$
Return $\mathbf{c}$
```

```
procedure Challenge():                                  $G_4$
$\mathbf{m}_0 \leftarrow \mathbf{m}$
$\mathbf{m}_1 \leftarrow_{\$} \mathcal{M}_\infty(1^k, \alpha, I, \mathbf{m}[I], \mathbf{z})$
Return $\mathbf{m}_b$
```

Figure 5.11: Games for the proof of Theorem 5.5.1.

encryption schemes we defined in the last section.

Thus, with lossy encryption, our simulator can give the adversary a lossy public key, the adversary cannot tell the difference, and now the simulator can find coins r' to open encryptions of all 0s to any message of its choosing. One caveat is that finding these coins r' may or may not be efficient. For schemes where this is efficient, we can prove SEM-SRO security, while for schemes where it is not known to be efficient we can still prove IND-SRO security. Now that we have given the high-level ideas behind our results, we give detailed proofs.

Proof of Theorem 5.5.1: We will prove the theorem using a sequence of game transitions, starting with a game G_0, which is the same as game $\text{SROIND}_{\mathcal{AE},\mathcal{M},k}$. All of the game transitions can be found in Figure 5.11.

```
Adversary $C(pars, pk)$:
$b \leftarrow_{\$} \{0,1\}$
Run $A(pars, pk)$.
  On query $\mathbf{Sample}(\alpha)$:
  $\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
  For $\ell = 1 \ldots |\mathbf{m}|$ do:
    $\mathbf{z}[\ell] \leftarrow 0^{|\mathbf{m}[\ell]|}$
    $\mathbf{c}[\ell] \leftarrow \mathbf{LR}_C(\mathbf{m}[\ell], \mathbf{z}[\ell])$
  Return $\mathbf{c}$
  On query $\mathbf{Corrupt}(I)$:
  For $i \in I$ do:
    $\mathbf{r}'[i] \leftarrow_{\$} \mathsf{Open}_{\infty}(pars, pk, \mathbf{m}[i], \mathbf{c}[i])$
  Return $\mathbf{m}[I], \mathbf{r}'[I]$
  On query $\mathbf{Challenge}()$:
  $\mathbf{m}_0 \leftarrow \mathbf{m}$
  $\mathbf{m}_1 \leftarrow_{\$} \mathcal{M}(1^k, \alpha, I, \mathbf{m}[I], \mathbf{z})$
  Return $\mathbf{m}_b$
When $A$ halts with output $b'$,
halt and output 1 if $(b = b')$ and 0 other-
wise.
```

Figure 5.12: Adversary used in the proof of Theorem 5.5.1.

Thus, by definition

$$\mathbf{Adv}^{\text{ind-sro}}_{A,\mathcal{AE},\mathcal{M}}(k) = 2 \cdot \Pr\left[\, G_0^A \Rightarrow \mathsf{true} \,\right] - 1 \; . \tag{5.9}$$

We will bound this probability by gradually changing game G_0 until we end up at a game in which A has no advantage. The first game transition replaces the public key returned by **Initialize** with a lossy public key, resulting in game G_1.

We claim there exists an efficient KEYIND adversary B such that

$$\Pr[\, G_0 \Rightarrow \mathsf{true} \,] - \Pr[\, G_1 \Rightarrow \mathsf{true} \,] \leq \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) \; . \tag{5.10}$$

Adversary B, given a challenge public-key pk that is either real or lossy, runs adversary A as in game G_0 with input the challenge key. Notice that since adversary B needs to be efficient, we need the resampling algorithm to be efficient; this is the only place in our proof where we need this requirement.

In our next game, G_2, we modify the **Corrupt** procedure so that instead of opening to the actual coins used to encrypt in **Sample**, it uses the unbounded Open_∞ algorithm to find just-as-likely randomness and returns that to the adversary. From the adversary's point of view, this randomness is equally-likely, so

$$\Pr[\, G_1 \Rightarrow \mathsf{true} \,] = \Pr[\, G_2 \Rightarrow \mathsf{true} \,] \ . \tag{5.11}$$

Next, for game G_3 we modify the **Sample** procedure so that after sampling a message vector $\mathbf{m}$ it only uses the lengths of the messages and encrypts dummy messages of the appropriate lengths. We emphasize that in the **Corrupt** procedure we still open to the actual messages that were sampled.

We can bound the gap between games G_2 and G_3 using the ind-cpa advantage of an unbounded adversary against the hiding scheme $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}_\ell, \mathcal{E})$. To see this, first consider adversary C, playing game INDCPA, and shown in Figure 5.12. If the challenge bit is 0, C perfectly simulates game G_2 for A, while if the challenge bit is 1 C perfectly simulates game G_3. Thus, it follows that

$$\Pr[\, G_2 \Rightarrow \mathsf{true} \,] - \Pr[\, G_3 \Rightarrow \mathsf{true} \,] \leq \mathbf{Adv}^{\text{ind-cpa}}_{C,\mathsf{Hid}}(k) \ , \tag{5.12}$$

where adversary C makes n queries to **LR**. Applying Proposition 2.4.1, there is an unbounded adversary D such that

$$\Pr\left[\, G_2^A \Rightarrow \mathsf{true} \,\right] - \Pr\left[\, G_3^A \Rightarrow \mathsf{true} \,\right] \leq n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) \ , \tag{5.13}$$

where D makes only one **LR** query in game INDCPA.

Next, we change game G_3 so that instead of using the resample mode of the message sampler $\mathcal{M}$ it instead uses the (unbounded) perfect resampling procedure

$\mathcal{M}_\infty$. We claim that

$$\Pr\left[\, G_3^A \Rightarrow \mathsf{true} \,\right] - \Pr\left[\, G_4^A \Rightarrow \mathsf{true} \,\right] \le \delta_{\mathcal{M}}(k)\ . \tag{5.14}$$

Finally, we claim that

$$\Pr\left[\, G_4^A \Rightarrow \mathsf{true} \,\right] = \frac{1}{2}\ . \tag{5.15}$$

This is true since A is given no information about $\mathbf{m}_0$ other than the lengths of the messages and the messages at indices I (since dummy messages of the appropriate lengths are encrypted). Thus, the resampled message $\mathbf{m}_1$, which has the same lengths and same messages at I, is by definition exactly equally as likely as $\mathbf{m}_0$. Thus, the adversary has no advantage at guessing the bit.

Combining the above equations, we can see that

$$\begin{aligned}
&\frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-sro}}_{A,\mathcal{AE},\mathcal{M}}(k) + \frac{1}{2} \\
&\quad = \Pr\left[\, G_0^A \Rightarrow \mathsf{true} \,\right] \\
&\quad \le \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + \Pr\left[\, G_1^A \Rightarrow \mathsf{true} \,\right] \\
&\quad \le \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + \Pr\left[\, G_2^A \Rightarrow \mathsf{true} \,\right] \\
&\quad \le \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) + \Pr\left[\, G_3^A \Rightarrow \mathsf{true} \,\right] \\
&\quad \le \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) + \delta_{\mathcal{M}}(k) + \Pr\left[\, G_4^A \Rightarrow \mathsf{true} \,\right] \\
&\quad \le \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) + n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) + \delta_{\mathcal{M}}(k) + \frac{1}{2}
\end{aligned}$$

which proves the theorem.

▮

Proof of Theorem 5.5.2: We will prove the theorem using a sequence of game transitions, starting with a game G_0, which is the same as game SROSEM. All of the game transitions are shown in Figure 5.13.

```
procedure Initialize:                                   $G_1, G_2, G_3, G_4$
$pars \leftarrow_{\$} \mathcal{P}(1^k)$ ; $(pk, sk) \leftarrow_{\$} \mathcal{K}_\ell(pars)$
Return $pk$
```

```
procedure Corrupt($I$):                                 $G_2, G_3$
For $i \in I$ do:
    $\mathbf{r}'[i] \leftarrow_{\$} \mathsf{Open}_\infty(pars, pk, \mathbf{m}[i], \mathbf{c}[i])$
Return $\mathbf{m}[I], \mathbf{r}'[I]$
```

```
procedure Sample($\alpha$):                             $G_3, G_4$
$\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
For $\ell = 1 \ldots |\mathbf{m}|$ do:
    $\mathbf{r}[\ell] \leftarrow_{\$} \mathsf{Coins}_{\mathcal{E}}(pars, pk)$ ; $\mathbf{z}[\ell] \leftarrow 0^{|\mathbf{m}[\ell]|}$
    $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{z}[\ell] ; \mathbf{r}[\ell])$
Return $\mathbf{c}$
```

```
procedure Corrupt($I$):                                 $G_4$
For $i \in I$ do:
    $\mathbf{r}'[i] \leftarrow_{\$} \mathsf{Open}(pars, pk, sk, \mathbf{m}[i], \mathbf{z}[i], \mathbf{r}[i], \mathbf{c}[i])$
Return $\mathbf{m}[I], \mathbf{r}'[I]$
```

Figure 5.13: Games for proof of Theorem 5.5.2.

First, recall that

$$\mathbf{Adv}^{\text{sem-sro}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) = \Pr\left[\,\text{SROSEM}^{A}_{\mathcal{AE},\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] - \Pr\left[\,\text{Id}^{S}_{\mathcal{M},\mathcal{R},k} \Rightarrow 1\,\right] . \quad (5.16)$$

We let game G_0 be identical to game $\text{SROSEM}_{\mathcal{AE},\mathcal{M},\mathcal{R},k}$. Game G_1 differs from G_0 only in that **Initialize** uses the lossy key generation algorithm $\mathcal{K}_\ell$ instead of $\mathcal{K}$. It is easy to see that there exists an efficient KEYIND adversary B such that

$$\Pr[\,G_0 \Rightarrow 1\,] - \Pr[\,G_1 \Rightarrow 1\,] \leq \mathbf{Adv}^{\text{key-ind}}_{B,\mathcal{AE},\mathcal{K}_\ell}(k) . \quad (5.17)$$

Adversary B, given a challenge public-key pk^* that is either real or lossy, runs adversary A as in game G_0, but with the challenge key pk^* as input.

```
Adversary $C(pars, pk)$:
Run $A(pars, pk)$.
  On query $\mathbf{Sample}(\alpha)$:
  $\mathbf{m} \leftarrow_{\$} \mathcal{M}(1^k, \alpha)$
  For $\ell = 1 \ldots |\mathbf{m}|$ do:
    $\mathbf{z}[\ell] \leftarrow 0^{|\mathbf{m}[\ell]|}$
    $\mathbf{c}[\ell] \leftarrow \mathbf{LR}_C(\mathbf{m}[\ell], \mathbf{z}[\ell])$
  Return $\mathbf{c}$
  On query $\mathbf{Corrupt}(I)$:
  For $i \in I$ do:
    $\mathbf{r}'[i] \leftarrow_{\$} \mathsf{Open}_{\infty}(pars, pk, \mathbf{m}[i], \mathbf{c}[i])$
  Return $\mathbf{m}[I], \mathbf{r}'[I]$
When $A$ halts with output $w$,
halt and output $\mathcal{R}(1^k, \mathbf{m}, I, w, \alpha)$
```

Figure 5.14: Adversary used in the proof of Theorem 5.5.2.

Next, we change G_1 into G_2 by changing the **Corrupt** procedure so that on a query I, instead of returning the actual coins $\mathbf{r}[I]$ used in **Sample**, it finds equally-likely coins $\mathbf{r}'[I]$ that still open the ciphertexts to the actual messages. It does this by running the (possibly unbounded) opening algorithm Open_{∞}. Since these coins are just as likely given the view of the adversary, it follows that the games are identical from the adversary's point of view and

$$\Pr[\, G_1 \Rightarrow 1 \,] = \Pr[\, G_2 \Rightarrow 1 \,] \ . \tag{5.18}$$

The next game, G_3, modifies the **Sample** procedure to encrypt dummy messages of the appropriate length instead of the messages sampled by $\mathcal{M}$. However, importantly, **Corrupt** still opens the ciphertexts to the actual messages sampled by $\mathcal{M}$ (by again using Open_{∞}). We can bound the gap between games G_2 and G_3 using the ind-cpa advantage of an unbounded adversary against the hiding scheme $\mathsf{Hid} = (\mathcal{P}, \mathcal{K}_{\ell}, \mathcal{E})$. To see this, first consider adversary C, playing game INDCPA, and shown in Figure 5.15. If the challenge bit is 0, C perfectly simulates game G_2 for A, while if the challenge bit is 1 C perfectly simulates game G_3. Thus, it

```
Simulator $S(1^k)$:
$pars \leftarrow_\$ \mathcal{P}(1^k)$
$(pk, sk) \leftarrow_\$ \mathcal{K}_\ell(pars)$
Run $A(pars, pk)$.

 On query $\mathbf{Sample}(\alpha)$:
 $\mathbf{z} \leftarrow \mathbf{Sample}_S(\alpha)$
 For $\ell = 1 \ldots |\mathbf{z}|$ do:
    $\mathbf{r}[\ell] \leftarrow_\$ \mathsf{Coins}_{\mathcal{E}}(pars, pk)$
    $\mathbf{c}[\ell] \leftarrow \mathcal{E}(pars, pk, \mathbf{z}[\ell]\ ;\ \mathbf{r}[\ell])$
 Return $\mathbf{c}$

 On query $\mathbf{Corrupt}(I)$:
 $\mathbf{m}[I] \leftarrow \mathbf{Corrupt}_S(I)$
 For $i \in I$ do:
    $\mathbf{r}'[i] \leftarrow_\$ \mathsf{Open}(pars, pk, sk, \mathbf{m}[i], \mathbf{z}[i], \mathbf{r}[i], \mathbf{c}[i])$
 Return $\mathbf{m}[I], \mathbf{r}'[I]$

When $A$ halts with output $w$,
halt and output $w$
```

Figure 5.15: Simulator used in the proof of Theorem 5.5.2.

follows that

$$\Pr[\, G_2 \Rightarrow 1 \,] - \Pr[\, G_3 \Rightarrow 1 \,] \leq \mathbf{Adv}^{\text{ind-cpa}}_{C,\mathsf{Hid}}(k)\ , \tag{5.19}$$

where adversary C makes n queries to **LR**. Applying Proposition 2.4.1, there is an unbounded adversary D such that

$$\Pr\left[\, G_2^A \Rightarrow 1 \,\right] - \Pr\left[\, G_3^A \Rightarrow 1 \,\right] \leq n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k)\ , \tag{5.20}$$

where D makes only one **LR** query in game IND.

The final game transition is to game G_4, where the **Corrupt** procedure (shown in Figure 5.13) uses the efficient opening algorithm Open instead of the potentially unbounded opening algorithm. Algorithm Open takes a few extra inputs, which the game chooses. Since algorithms Open_∞ and Open have the same output distribution, the view of an adversary playing the game is identical and we see that

$$\Pr\left[\, G_3^A \Rightarrow 1 \,\right] = \Pr\left[\, G_4^A \Rightarrow 1 \,\right] \; . \tag{5.21}$$

Finally, we note that running game G_4 with adversary A is the same as running game Id with simulator S, shown in Figure 5.15, meaning that

$$\Pr\left[\, G_4^A \Rightarrow 1 \,\right] = \Pr\left[\, \mathrm{Id}_{\mathcal{M},\mathcal{R},k}^S \Rightarrow 1 \,\right] \; . \tag{5.22}$$

This is because S only needs the message length in **Sample** (which it can get from its own **Sample** oracle) and actual sampled messages are not needed until **Corrupt**, where they can be learned by S by querying its own **Corrupt** oracle. Also in **Corrupt**, S can run the efficient opening algorithm Open since S chooses all of the inputs to Open.

Combining the above equations, we can see that

$$\begin{aligned}
&\mathbf{Adv}^{\text{sem-sro}}_{A,S,\mathcal{AE},\mathcal{M},\mathcal{R}}(k) \\
&\quad = \Pr\left[\, \mathrm{SROSEM}^A_{\mathcal{AE},\mathcal{M},\mathcal{R},k} \Rightarrow 1 \,\right] - \Pr\left[\, \mathrm{Id}^S_{\mathcal{M},\mathcal{R},k} \Rightarrow 1 \,\right] \\
&\quad = \Pr\left[\, G_0^A \Rightarrow 1 \,\right] - \Pr\left[\, G_4^A \Rightarrow 1 \,\right] \\
&\quad = (\Pr\left[\, G_0^A \Rightarrow 1 \,\right] - \Pr\left[\, G_1^A \Rightarrow 1 \,\right]) + (\Pr\left[\, G_1^A \Rightarrow 1 \,\right] - \Pr\left[\, G_2^A \Rightarrow 1 \,\right]) + \\
&\qquad (\Pr\left[\, G_2^A \Rightarrow 1 \,\right] - \Pr\left[\, G_3^A \Rightarrow 1 \,\right]) + (\Pr\left[\, G_3^A \Rightarrow 1 \,\right] - \Pr\left[\, G_4^A \Rightarrow 1 \,\right]) \\
&\quad \leq \mathbf{Adv}^{\text{key-ind}}_{A,\mathcal{AE},\mathcal{K}_\ell}(k) + 0 + n(k) \cdot \mathbf{Adv}^{\text{ind-cpa}}_{D,\mathsf{Hid}}(k) + 0
\end{aligned}$$

which proves the theorem. ▌

5.6 Conclusion and Additional Information

In this chapter we investigated PKE security under selective opening attacks. We saw that revealed randomness failures lead to the main technical difficulties in this setting. Our main technical contribution was showing that lossy encryption schemes can be proven secure against selective opening attacks.

Credits. An earlier version [17] of the material in this chapter appeared as part of work appearing in the Proceedings of EUROCRYPT 2009, copyright IACR, and co-authored with Mihir Bellare and Dennis Hofheinz. I was a primary researcher for this work.

Bibliography

[1] Amazon elastic computing cloud. http://aws.amazon.com/ec2/.

[2] GNU Privacy Guard. http://www.gnupg.org/.

[3] Openssl. http://www.openssl.org/.

[4] VirtualBox. http://www.virtualbox.org.

[5] VMWare. http://www.vmware.com.

[6] Paolo Abeni, Luciano Bello, and Maximiliano Bertacchini. Exploiting DSA-1571: How to break PFS in SSL with EDH, July 2008. http://www.lucianobello.com.ar/exploiting_DSA-1571/index.html.

[7] Boaz Barak, Oded Goldreich, Shafi Goldwasser, and Yehuda Lindell. Resettably-sound zero-knowledge and its applications. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science – FOCS 2001*, pages 116–125. IEEE, 2001.

[8] Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In *Proceedings of the 12th ACM Conference on Computer and Communications Security – CCS 2005*, pages 203–212. ACM, 2005.

[9] Olivier Baudron, David Pointcheval, and Jacques Stern. Extended notions of security for multicast public key cryptosystems. In *Proceedings of the 27th Colloquium on Automata, Languages and Programming – ICALP 2000*, volume 1853 of *LNCS*, pages 499–511. Springer, 2000.

[10] Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. Multi-recipient encryption schemes: Efficient constructions and their security. *IEEE Transactions on Information Theory*, 53(11):3927–3943, 2007.

[11] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology – EUROCRYPT 2000*, number 1807 in LNCS, pages 259–274. Springer, 2000.

[12] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology – CRYPTO 2007*, number 4622 in LNCS, pages 535–552. Springer, 2007.

[13] 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. In *Advances in Cryptology – ASIACRYPT 2009*, number 5912 in LNCS, pages 232–249. Springer, 2009.

[14] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO 1996*, number 1109 in LNCS, pages 1–15. Springer, 1996.

[15] Mihir Bellare, Marc Fischlin, Shafi Goldwasser, and Silvio Micali. Identification protocols secure against reset attacks. In *Advances in Cryptology – EUROCRYPT 2001*, number 2045 in LNCS, pages 495–511. Springer, 2001.

[16] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *LNCS*, pages 360–378. Springer, 2008.

[17] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In *Advances in Cryptology – EUROCRYPT 2009*, number 5479 in LNCS, pages 1–35. Springer, 2009.

[18] Mihir Bellare, Tadayoshi Kohno, and Victor Shoup. Stateful public-key cryptosystems: How to encrypt with one 160-bit exponentiation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security – CCS 2006*, pages 380–389. ACM, 2006.

[19] Mihir Bellare, Saurabh Panjwani, and Scott Yilek. Relations among notions of security under selective decryption/decommitment attack. Unpublished Manuscript, 2008.

[20] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of 1st ACM Conference on Computer and Communications Security – CCS 1993*, pages 62–73. ACM, 1993.

[21] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. In *Advances in Cryptology – EUROCRYPT 2006*, number 4004 in LNCS, pages 409–426. Springer, 2006.

[22] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th ACM Symposium on Theory of Computing – STOC 1988*, pages 1–10. ACM, 1988.

[23] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *Advances in Cryptology – CRYPTO 2008*, number 5157 in LNCS, pages 335–359. Springer, 2008.

[24] Carl Bosley and Yevgeniy Dodis. Does privacy require true randomness? In *Proceedings of the 4th Theory of Cryptography Conference – TCC 2007*, number 4392 in LNCS, pages 1–20. Springer, 2007.

[25] Daniele R.L. Brown. A weak randomizer attack on RSA-OAEP with e=3. IACR ePrint Archive, 2005.

[26] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing – STOC 1995*, pages 639–648. ACM Press, 1996.

[27] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing – STOC 2000*, pages 235–244. ACM, 2000.

[28] Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-secure, non-interactive public-key encryption. In *Proceedings of the Second Theory of Cryptography Conference – TCC 2005*, number 3378 in LNCS, pages 150–168. Springer, 2005.

[29] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proceedings of the 20th ACM Symposium on Theory of Computing – STOC 1988*, pages 11–19. ACM, 1988.

[30] Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the 2001 Workshop on Hot Topics in Operating Systems*, pages 133–138, 2001.

[31] Richard S. Cox, Steven D. Gribble, Henry M. Levy, and Jacob Gorm Hansen. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 350–364. IEEE, 2006.

[32] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[33] Anand Desai, Alejandro Hevia, and Yiqun Lisa Yin. A practice-oriented treatment of pseudorandom number generators. In *Advances in Cryptology – EUROCRYPT 2002*, number 2332 in LNCS, pages 368–383. Springer, 2002.

[34] Yevgeniy Dodis, Shien Jin Ong, Manoj Prabhakaran, and Amit Sahai. On the (im)possibility of cryptography with imperfect randomness. In *Proceedings of the 45th Symposium on Foundations of Computer Science – FOCS 2004*, pages 196–205. IEEE, 2004.

[35] Leo Dorrendorf, Zvi Gutterman, and Benny Pinkas. Cryptanalysis of the windows random number generator. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security – CCS 2007*, pages 476–485. ACM, 2007.

[36] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry Stockmeyer. Magic functions. *Journal of the ACM*, 50(6):852–921, 2003.

[37] Serge Fehr, Dennis Hofheinz, Eike Kiltz, and Hoeteck Wee. Encryption schemes secure against chosen-ciphertext selective opening attacks. In *Advances in Cryptology – EUROCRYPT 2010*, number 6110 in LNCS. Springer, 2010.

[38] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – CRYPTO 1984*, number 196 in LNCS, pages 10–18. Springer, 1985.

[39] Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proceedings of the 10th Workshop on Hot Topics in Operating Systems – HotOS-X*, May 2005.

[40] Ian Goldberg and David Wagner. Randomness in the Netscape browser. Dr. Dobb's Journal, January 1996.

[41] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[42] Vipul Goyal and Amit Sahai. Resettably secure computation. In *Advances in Cryptology – EUROCRYPT 2009*, number 5479 in LNCS, pages 54–71. Springer, 2009.

[43] Zvi Gutterman and Dahlia Malkhi. Hold your sessions: An attack on Java session-id generation. In *Topics in Cryptology – CT-RSA 2005*, number 3376 in LNCS, pages 44–57. Springer, 2005.

[44] Zvi Gutterman, Benny Pinkas, and Tzachy Reinman. Analysis of the linux random number generator. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 371–385. IEEE, 2006.

[45] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[46] Brett Hemenway, Benoit Libert, Rafail Ostrovsky, and Damien Vergnaud. Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. IACR ePrint Archive Report 2009/088.

[47] Dennis Hofheinz. Possibility and impossibility results for selective decommitments. IACR ePrint Archive, April 2008.

[48] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing – STOC 1989*, pages 12–24. ACM, 1989.

[49] Seny Kamara and Jonathan Katz. How to encrypt with a malicious random number generator. In *Fast Software Encryption – FSE 2008*, number 5086 in LNCS, pages 303–315. Springer, 2008.

[50] Gillat Kol and Moni Naor. Cryptography and game theory: Designing protocols for exchanging information. In *Proceedings of the 5th Theory of Cryptography Conference – TCC 2008*, number 4948 in LNCS, pages 320–339. Springer, 2008.

[51] Chi-Jen Lu. Encryption against storage-bounded adversaries from on-line strong extractors. *J. Cryptology*, 17(1):27–42, 2004.

[52] James L. McInnes and Benny Pinkas. On the impossibility of private key cryptography with weakly random keys. In *Advances in Cryptology – CRYPTO 1990*, number 537 in LNCS, pages 421–435. Springer, 1991.

[53] Silvio Micali and Leonid Reyzin. Soundness in the public-key model. In *Advances in Cryptology – CRYPTO 2001*, number 2139 in LNCS, pages 542–565. Springer, 2001.

[54] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms – SODA 2001*, pages 448–457. ACM/SIAM, 2001.

[55] Jesper B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology – CRYPTO 2002*, number 2442 in LNCS, pages 111–126. Springer, 2002.

[56] Khaled Ouafi and Serge Vaudenay. Smashing SQUASH-0. In *Advances in Cryptology – EUROCRYPT 2009*, number 5489 in LNCS, pages 300–312. Springer, 2009.

[57] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology – CRYPTO 2008*, number 5157 in LNCS, pages 554–571. Springer, 2008.

[58] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing – STOC 2008*, pages 187–196. ACM Press, 2008.

[59] Niels Provos. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, 2004.

[60] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology – CRYPTO 1991*, number 576 in LNCS, pages 433–444. Springer, 1992.

[61] Thomas Ristenpart and Scott Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Proceedings of the 17th Annual Network and Distributed System Security Symposium – NDSS 2010*. Internet Society, 2010.

[62] Phillip Rogaway. Nonce-based symmetric encryption. In *Fast Software Encryption – FSE 2004*, number 3017 in LNCS, pages 348–359. Springer, 2004.

[63] Phillip Rogaway and Thomas Shrimpton. Deterministic authenticated-encryption: A provable-security treatment of the key-wrap problem. In *Advances in Cryptology – EUROCRYPT 2006*, number 4004 in LNCS, pages 373–390. Springer, 2006.

[64] Victor Shoup. Using hash functions as a hedge against chosen ciphertext attack. In *Advances in Cryptology – EUROCRYPT 2000*, number 1807 in LNCS, pages 275–288. Springer, 2000.

[65] Robert Woolley, Mark Murray, Maxim Dounin, and Ruslan Ermilov. arc4random predictable sequence vulnerability. `http://security.freebsd.org/advisories/FreeBSD-SA-08:11.arc4random.asc`, 2008.

[66] Scott Yilek. Resettable public-key encryption: How to encrypt on a virtual machine. In *Topics in Cryptology – CT-RSA 2010*, number 5985 in LNCS, pages 41–56. Springer, 2010.

[67] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL

vulnerability. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement – IMC 2009*, pages 15–27. ACM, 2009.

[68] David Zuckerman. Simulating BPP using a general weak random source. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science – FOCS 1991*, pages 79–89. IEEE, 1991.