Computing Securely with Untrusted Resources

by

Seny Kamara

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

April, 2008

Abstract

When designing and analyzing cryptosystems, it is usually assumed that the computational devices used by the honest parties have access to resources that are outside of the malicious parties' control. In such a model, it is known, under standard cryptographic assumptions, that essentially any operation can be performed securely as long as a majority of the parties are honest.

In many practical settings, however, the assumption that computational resources can be protected from an adversary does not hold. This dissertation explores various security problems in settings where honest parties wish to make use of computational resources that are under adversarial control. We focus on resources that are fundamental to cryptography, such as randomness and storage.

We first consider the problem of encrypting with a malicious random number generator. We introduce the notions of security against chosen-randomness attacks (CRA) and security against chosen-ciphertext and randomness attacks (CCRA), which formally capture the security of private-key encryption when used with sources of randomness that are under adversarial control. We study the relationships be-

tween these notions and the traditional notions of security for encryption. We also show how to design efficient schemes that are CRA-secure, and how to transform any CPA-secure scheme into a CRA-secure one, and any CRA-secure scheme into a CCRA-secure one.

We then turn to the task of authenticating data stored in unreliable memory. We propose a general framework for designing efficient "proofs of data possession", which are proof systems that enable one to convince a verifier that it stores a particular piece of data. We give a compiler that transforms any sigma-protocol (i.e., a three-round public-coin zero-knowledge proof of knowledge) into a proof of data possession.

Finally, we consider the problem of storing private data in untrusted memory. We show how to design private-key encryption schemes that allow one to search over encrypted content. Our constructions are optimal in terms of search time. We also introduce searchable encryption in the multi-user setting, where search privileges can be delegated to a set of authorized users.

Thesis Readers:

Fabian Monrose
Associate Professor & Advisor
Department of Computer Science
The Johns Hopkins University

Giuseppe Ateniese
Associate Professor
Department of Computer Science
The Johns Hopkins University

Juan Garay
Member of Technical Staff
Bell Labs - Alcatel-Lucent

Jonathan Katz
Associate Professor
Department of Computer Science
Univeristy of Maryland

Acknowledgements

I would like to thank my advisor Fabian Monrose for his guidance, advice, and generous support. I first met Fabian during a summer internship at Bell Labs, and since then he has taken on numerous roles, inclduing supervisor, mentor, advisor, collaborator and, most importantly, friend. Fabian has taught me how to do research, how to communicate my ideas effectively, and how to be critical of my own work. He has always given me the freedom to pursue my own ideas, all the while making sure I stayed productive. As anyone at Hopkins can attest to, his support and dedication to students are an inspiration.

Over the past three years, I have learned a great deal about cryptography from Jonathan Katz. Jonathan patiently listened to all my ideas, read all my emails and corrected all my mistakes. Though I don't always keep up, I have benefited tremendously by working with Jonathan and observing him.

I am also grateful to Juan Garay and Rafail Ostrovsky. I always had a great time during our collaborations and invariably learned something new and interesting from our interactions. I am particularly thankful to Rafi for inviting me to spend a

semester at the UCLA Institute for Pure and Applied Mathematics. This was a great program where I learned a tremendous amount and met great people.

The Hopkins Information Security Institute was a wonderful place to work, and I am indebted to Gerry Masson, Avi Rubin and Fabian for offering me the possibility to stay at ISI a bit longer, allowing me to focus on my family during difficult times. Avi's continuous optimism about my research and career was always helpful when I was less confident about what lied ahead. I am also very grateful to Susan Hohenberger from whom I've learned a considerable amount in a short period of time. Her belief in me, professional guidance and overall enthusiasm was invaluable.

Over the years, I have benefited greatly from discussions with various people. My conversations with Lucas Ballard often lead to new and intriguing questions. Breno de Medeiros has consistently and patiently answered all my questions about mathematics and cryptography, and Susanne Wetzel has always went out of her way to support me in my growth as a researcher.

I have also learned a great deal from my other collaborators, including Giuseppe Ateniese, Ryan Caudy, Reza Curtmola, Darren Davis, Kevin Fu, Yoshi Kohno, and Mike Reiter. I am especially thankful to Mike Reiter and the members of the Cylab for hosting me during a summer at Carnegie Mellon.

My time at Hopkins would not have been the same without the students in the SPAR lab and its many visitors: Lucas Ballard, Steve Bono, Scott Coull, Reza Curtmola, Darren Davis, Anna Lisa Ferrara, Kevin Fu, Sujata Garera, Yong Ho

Hwang, Paolo Gasti, Ryan Gardner, Matt Green, Yoshi Kohno, Josh Mason, Breno de Medeiros, Moheeb Abu Rajab, Sam Small, Chris Soghoian, Aniello del Sorbo, Sophie Qiu, Charles Wright, and Jay Zarfoss.

I decided to pursue a Ph.D. while I was an undergraduate at Purdue University after Gene Spafford and Steve Hare gave me my first opportunity to do research. I am grateful to them, as well as to Wojciech Szpankowski, Sonia Fahmy, Pascal Meunier and the Center for Education and Research in Information Assurance and Security for the opportunity.

Throughout the years, I was generously supported by a Bell Labs Graduate Research Fellowship, a Phillips and Camille Bradford fellowship, the UCLA Institute for Pure and Applied Mathematics, and the Johns Hopkins Information Security Institute.

Graduate school is not the easiest of times, but my wife Mariam's unending support and belief in me always made it easier for me to work through difficult times. I cannot imagine how I could have accomplished this, or much else, without her. Though I did not realize it at the time, moving to Indiana was the best decision I ever made. Thanks also to Sani, who consistenly reminds us not to take ourselves too seriously. Finally, I would like to thank my mother for the sacrifices she made, the unconditional support she gave, and the example she set. She showed me the world and inspired me to pursue my dreams. Though my own research will never inspire as many people as hers, everything I have accomplished is a direct result of her teaching.

This thesis is dedicated to my mother, Sylviane Diouf.

Contents

List of Figures

Chapter 1

Introduction

The advent of information technology has profoundly transformed our society. Almost everything, from our financial transactions to our elections are conducted electronically, and today most of our communications are carried out over the Internet. While the benefits of electronic communication, such as decreased costs and increased speed and reliability, are clear, it has also given rise to new challenges. So far, one of the most difficult of these challenges seems to be that of striking a balance between the utility and the security of information systems.

Indeed, while — prior to the advent of information technology — society had put legal and physical mechanisms in place to achieve such a balance, our transition to an electronic society has required us to develop new tools for this purpose. From a technical perspective, one of the most important developments to come out of this motive is perhaps the "modernization" of cryptography. In popular culture, cryptography

is understood to be "encryption". In other words, the study of techniques used to make communication unintelligible so as to hide its meaning from anyone other than its intended recipient. And indeed, this correctly characterizes most of the work done in cryptography until the late 20^{th} century. In the late 70's and early 80's, however, cryptography underwent a revolution of sorts, both in terms of its scope and its approach. Today it is concerned not only with protecting communications, but, among other things, with authenticating it, with identifying users, and with securing distributed computations, such as electronic auctions and elections. And while prior to the 50's the methodology used to design cryptographic primitives was mostly ad-hoc and rested on experience and intuition, cryptography now makes use of general and principled design paradigms that are supported by a well-established mathematical foundation.

Traditionally, the settings considered in cryptography include a set of honest parties that wish to perform some operation and a set of malicious parties that wish to either learn information related to the operation (e.g., its inputs or outputs) or influence it in some way (e.g., by changing its output). In particular, it is usually assumed that the honest parties have access to certain resources, such as randomness or storage, which are outside of the malicious parties' control. In such an adversarial model it is known that, under standard cryptographic assumptions, essentially any operation can be performed securely as long as the majority of participants are honest [65, 38, 37].

In many practical settings, however, this assumption does not hold. In fact, more often than not honest parties have access to resources which they only partially trust. This can occur, for example, when the honest parties wish to make use of resources that are controlled by an untrusted party; or because they are unwittingly using resources that have been compromised. In this thesis we explore various security problems in settings such as the one described above, where honest parties wish to make use of resources which are under adversarial control. We focus on randomness and storage, and we consider the problems of encrypting data with a malicious random number generator; of authenticating data stored on an unreliable storage server; and of storing private data on an untrusted storage server.

1.1 Encrypting with a malicious random number generator

Though today the scope of modern cryptography is wide and includes a multitude of applications beyond that of secure communication, the most widely used cryptographic primitive is still encryption. An encryption scheme is an algorithm that takes a message, also referred to as a plaintext, and uses a key to transform it into a ciphertext. In a *private-key* encryption scheme, this is done in such a way that the plaintext can be recovered from the ciphertext if and only if the key used during encryption is known. Traditionally, private-key encryption has been used to secure communication

between two parties that share a key. If a sender, Alice, wishes to send a private message to a recipient, Bob, then Alice uses their shared-key to encrypt her message before sending it. Since Bob knows the key that Alice used to encrypt the message, he can decrypt the ciphertext and recover the plaintext.

We begin by exploring how private-key encryption can be made secure in settings where encryption is carried out using a potentially untrusted pseudo-random generators (PRG). Today, it is well known how to construct "secure" [1] private-key encryption schemes under the assumption that the sender has access to a source of randomness that has a high amount of entropy. In fact, most private-key encryption scheme assume such a source is available during the key generation step and for each call to the encryption algorithm. In practice, however, such an assumption does not always hold. Consider, for example, the case where encryption is carried out on a smartcard. In this setting an adversary with physical access to the card might be able to tamper with its source of randomness and force the encryption scheme to be used with "weak" randomness, i.e., randomness that is partly predictable. Another example is when encryption is performed on a multi-user system, such as a Unix server, where the entropy pool is generated from user actions, such as user typing patterns, mouse movements or packet inter-arrival times. In systems such as these, where an adversary can affect the entropy pool directly, it is not always clear whether the system has "enough" entropy or not. Finally, the system could simply be using

[1] In chapter 2 we make precise what we mean by a secure encryption scheme.

a defective PRG that outputs biased bits.

Contributions. In Chapter 3, we show how to design private-key encryption schemes that guarantee the security of data, even when they are used with "weak" randomness. More precisely, our encryption schemes guarantee that if a message is encrypted using a truly random key and a truly random source of bits, the message will be protected even against an adversary that will control the source of randomness in the future, or one that controlled it in the past. So while we assume that a high entropy source is available during the key generation and encryption steps [2], we are still able to guarantee security even if the source of randomness is completely under the adversary's control at any other point in time. We begin by proposing two formal definitions that capture security against the kind of attacks previously described: security against *chosen-randomness attacks* (CRA) and the stronger security against *chosen-ciphertext and randomness attacks* (CCRA). We then show how to construct efficient private-key encryption schemes that are provably CRA-secure, and propose two general and efficient transformations that turn any CPA-secure encryption scheme into a CRA-secure scheme, and any CRA-secure scheme into a CCRA-secure scheme.

[2] We show in Chapter 3 that these assumptions are necessary.

1.2 Authenticating data stored on an unreliable server

Advances in networking technology and the rapid accumulation of information have fueled a trend toward outsourcing data management to external service providers. By outsourcing, organizations and individuals can concentrate on their core tasks rather than incurring the substantial hardware, software and personnel costs involved in maintaining data "in house".

Storage outsourcing prompts a number of interesting challenges. In the context of large outsourced archival storage, it is important to consider the issues surrounding potential misbehaving service providers. The main problem is in verifying that the server continually and faithfully stores the entire and authentic content entrusted to it by the client. The server is untrusted in terms of both security and reliability: it might maliciously or accidentally erase the data or place it onto slow or off-line storage media. This could occur for numerous reasons, including to save storage or to comply with external pressures (e.g., government censure). Also, the server could accidentally erase some data and choose not to notify the owner. Exacerbating the problem (and precluding naïve approaches) are factors such as: limited real-time bandwidth between owners and servers, as well as the owner's limited computing and storage resources. We note that traditional approaches to verify integrity, such as message authentication codes or digital signatures, cannot be applied since the client

presumably does not store the data

To address this, the notion of a *proof of data possession* (PDP) system was recently put forth [4]. A PDP system is a proof system based on public-key techniques that enables the server to efficiently prove to the client — or anyone in possession of the client's public-key — that it possesses the client's data. PDP systems are similar to proofs of knowledge (POK) [41, 31], which are proof systems that enable a prover to convince a verifier that it knows a secret in "zero-knowledge", i.e., without leaking any partial information about the secret to the verifier. The main differences between POKs and PDP systems are that (1) PDP systems do not require the interaction to be zero-knowledge; and (2) since the intended use of PDPs is for proving possession of very large amounts of data, the communication complexity must be less than the size of the data.

Contributions. In Chapter 4, we begin by establishing an explicit connection between POKs and PDP systems. Specifically, we show how to compile 3-round POKs for **NP** languages into PDP systems. We note that our transformation is not a straightforward application of a POK where the data is committed to and the server proves knowledge. For most known constructions (e.g., [58, 43]), the previous approach would have communication and storage complexity that is linear in the size of the data. On the contrary, we make use of the protocols in a non-standard way, and show that if it possesses certain homomorphic properties, then the resulting PDP system can achieve communication and storage complexity at the client that is

independent of the size of the data.

1.3 Storing private data on an untrusted server

Recently, we have seen private-key encryption being increasingly used to secure stored data. This is mainly due to the fact that, as corporate storage needs are increasing, many companies find it easier and more cost effective to outsource the storage and management of their data to a third party. Another reason, however, is the passing of legislation such as the Gramm-Leach-Bliley Act [2] and the Health Insurance Portability and Accountability Act (HIPAA) [1], which mandate that financial and medical institutions protect customer information by implementing proper safeguards.

While it is clear that encrypting this sensitive data with a private-key encryption scheme before storage will guarantee its security, such a simple approach can be very inefficient. Indeed, consider the case of a hospital which stores all its patient records encrypted using a private-key encryption scheme. Note that in such a setting, even the existence of a record for a given patient should be kept private (i.e., the untrusted server should not even learn that a given person is a patient at the hospital). In this case, in order to recover a single record from the server a staff member will have to download the entire encrypted data collection, decrypt it, and search for the appropriate record. Clearly, this solution is inefficient both in terms of communication complexity and in terms of the computation performed by the server and the client.

Contributions. In Chapter 5, we show how to design private-key encryption schemes that preserve the security of data while allowing its owner to selectively retrieve certain segments of it. We refer to such primitives as *searchable symmetric encryption* (SSE) schemes, and begin our treatment of SSE by considering the problem of formally defining security for such schemes. While there has been some previous work that proposes formal security notions for SSE, we point out that all previous attempts have limitations. To address this, we propose two new notions of security: *security against non-adaptive adversaries*, and the stronger *security against adaptive adversaries*. In addition to our new definitions, we also propose two provably secure non-interactive constructions which achieve optimal (asymptotic) efficiency in terms of communication complexity and computation at both the server and the client.

Chapter 2

Notation and Preliminaries

Notation. The set of all binary strings of length n is denoted as $\{0,1\}^n$, and the set of all finite binary strings as $\{0,1\}^*$. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly at random from a set X. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. We write either $\langle a, b \rangle$ or $a||b$ to refer to the concatenation of strings a and b. The set of all functions from $\{0,1\}^n$ to $\{0,1\}^m$ is denoted as $\mathsf{Func}[n,m]$, and the set of all permutations from $\{0,1\}^n$ to $\{0,1\}^n$ as $\mathsf{Perm}[n,n]$. The set of all primes of length k is denoted as $\mathsf{primes}(k)$. Given a vector $\mathbf{v}$ in some arbitrary vector space we denote its i^{th} component as $\mathbf{v}[i]$.

Throughout this thesis $k \in \mathbb{N}$ will refer to the security parameter. A function $\nu : \mathbb{N} \to \mathbb{N}$ is *negligible* in k if for every polynomial $p(\cdot)$ and sufficiently large k, $\nu(k) < 1/p(k)$. Let $\mathsf{poly}(k)$ and $\mathsf{negl}(k)$ denote unspecified polynomial and negligible

functions in k. We write $f(k) = \mathsf{poly}(k)$ to mean that there exists a polynomial $p(\cdot)$ such that for all sufficiently large k, $f(k) \leq p(k)$; and $f(k) = \mathsf{negl}(k)$ to mean that there exists a negligible function $\nu(\cdot)$ such that for all sufficiently large k, $f(k) \leq \nu(k)$. A function $f : \mathbb{N} \to \mathbb{N}$ is *noticeable* in k if $f(k) \geq 1/\mathsf{poly}(k)$. We say that a function $f : \{0,1\}^* \to \{0,1\}^*$ is *efficiently computable* if there exists a polynomial-time algorithm that computes f.

Probabilistic algorithms. A probabilistic algorithm is a Turing Machine with an input tape, a work tape and a random tape. If $\mathcal{A}$ is a probabilistic algorithm, then $\mathcal{A}(x)$ denotes an execution of $\mathcal{A}$ on input x with uniformly chosen random coins, and $\mathcal{A}(x; r)$ denotes an execution of $\mathcal{A}$ on input x with random tape r. Given a probabilistic algorithm $\mathcal{A}$, we will consider oracle Turing machines that are given access to an oracle $\mathcal{A}(\cdot;\cdot)$ that on input $\langle x, r\rangle$ outputs $\mathcal{A}(x; r)$. Note that this is different from the usual case where the oracle Turing machine is given access to an oracle $\mathcal{A}(\cdot)$ that on input x returns $\mathcal{A}(x; r)$ for uniformly chosen random coins r.

A *probabilistic polynomial-time* (PPT) algorithm $\mathcal{A}$ is a Turing machine for which there exists a polynomial $p(\cdot)$ such that for all $x \in \{0,1\}^*$ and all random tapes $r \in \{0,1\}^*$, the running time of $\mathcal{A}(x; r)$ is at most $p(|x|)$.

Experiments. Our security definitions will be formalized using various probabilistic experiments. So if $\mathcal{A}_1$ through $\mathcal{A}_n$ are probabilistic algorithms and π is a predicate,

then

$$\Pr\left[\,\pi(x_1,\ldots,x_n) : x_1 \leftarrow \mathcal{A}_1;\ldots;x_n \leftarrow \mathcal{A}_n\,\right]$$

is the probability that $\pi(x_1,\ldots,x_n) = 1$ after executing $\mathcal{A}_1$ through $\mathcal{A}_n$. Alternatively, we may write this as

$$\Pr\left[\,\mathsf{Exp}(k) = 1\,\right],$$

where $\mathsf{Exp}(k)$ is defined as the probabilistic experiment that consists of executing $\mathcal{A}_1$ through $\mathcal{A}_n$ and outputting $\pi(x_1,\ldots,x_n)$.

Computational Indistinguishability. A *probability ensemble* is a collection of distributions $X = \{X_k\}_{k\in\mathbb{N}}$ indexed by a parameter k. Two probability ensembles $X = \{X_k\}_{k\in\mathbb{N}}$ and $Y = \{Y_k\}_{k\in\mathbb{N}}$ are *computationally indistinguishable* if for all PPT $\mathcal{A}$,

$$\left|\Pr\left[\,\mathcal{A}(x) = 1 : x \leftarrow X_k\,\right] - \Pr\left[\,\mathcal{A}(y) = 1 : y \leftarrow Y_k\,\right]\right| \leq \mathsf{negl}(k)$$

where the probabilities are over the choices of x and y and the coins of $\mathcal{A}$.

Relations. A binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is polynomially-bounded if for all $(x,w) \in R$, $|w| = \mathsf{poly}(|x|)$. Let $W_R(x) = \{w : (x,w) \in R\}$ be the witness set of x. We refer to the language induced by a relation R, as $L_R = \{x : |W_R(x)| > 0\}$. If $L_R \in \mathbf{NP}$, then we say that R is an **NP**-relation. We say that a relation R is *hard on average* if there exists a PPT algorithm Gen_R that takes a security parameter k as input and outputs a pair $(x,w) \in R$ such that $|x| = k$, and if for all PPT adversaries

$\mathcal{A}$,

$$\Pr\left[\,(x,w') \in R : (x,w) \leftarrow \mathsf{Gen}(1^k); w' \leftarrow \mathcal{A}(x)\,\right] \leq \mathsf{negl}(k).$$

2.1 Basic Cryptographic Primitives

Private-key encryption. Private-key encryption allows two parties that share a common and secret key to communicate privately. Intuitively, such a scheme should guarantee that, given a ciphertext, no adversary is able to learn any information about the message. Before formalizing this security intuition, however, we begin by defining private-key encryption schemes.

Definition 2.1.1 (Private-key encryption)**.** *A private-key encryption scheme* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *consists of three polynomial-time algorithms such that,*

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$*: takes as input a security parameter* 1^k *and outputs a key* K*.*

$c \leftarrow \mathsf{Enc}(\mathrm{K}, m)$*: is a probabilistic algorithm that takes as input the key* K *and a message* m *from some associated message space, and outputs a ciphertext* c*. We sometimes write this as* $c \leftarrow \mathsf{Enc}_{\mathrm{K}}(m)$*.*

$m \leftarrow \mathsf{Dec}(\mathrm{K}, c)$*: takes as input the key* K *and a ciphertext* c*, and outputs either a message* m *in the message space or a special failure symbol* $\perp$*. We sometimes write this as* $m \leftarrow \mathsf{Dec}_{\mathrm{K}}(c)$*. We assume without loss of generality that* Dec *is deterministic.*

For ease of exposition we will make certain simplifying assumptions about private-key encryption schemes, but we note that all the results in this thesis hold for any scheme. In most private-key encryption schemes, $\mathsf{Gen}(1^k)$ simply outputs a random key of length k. When this is the case, we will write $\Sigma = (\mathsf{Enc}, \mathsf{Dec})$. Also, when considering schemes that work over fixed-length messages, we will always assume that the message space is the set of strings of length k and that Enc uses k random coins. Finally, we will always assume that a private-key encryption scheme is perfectly correct in the sense that for all $k \in \mathbb{N}$, all K output by $\mathsf{Gen}(1^k)$, and all messages m in the message space, $\mathsf{Dec}_{\mathrm{K}}(\mathsf{Enc}_{\mathrm{K}}(m)) = m$.

Today, there are two commonly accepted notions of security for encryption: CPA-security and the stronger CCA-security. Roughly speaking, CPA-security guarantees that, given a ciphertext generated using an unknown key K, a computationally-bounded adversary cannot recover any partial information about the underlying plaintext, even if it is given access to an encryption oracle that returns encryptions (using the same key K) of any message m provided by the adversary. This "encryption oracle" is meant, in part, to model potential real-world actions of an adversary that might influence the honest sender to encrypt certain messages that are, either partially or entirely, under the adversary's control. CPA-security was first introduced for public-key encryption by Goldwasser and Micali in [40]. In that work the previous intuition was formally defined in two different ways. The first, which we recall in Definition 2.1.2, is "game-based" and is formulated in terms an experiment between an

adversary and a (honest) challenger. The second definition is "simulation-based" and is formulated in terms of two experiments: one in which the adversary interacts with the scheme under consideration and one where it interacts with an ideal version of the scheme. CPA-security for private-key encryption was later considered by Bellare, Desai, Jokipii and Rogaway [8].

Definition 2.1.2 (CPA-security for private-key encryption [40, 8])**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme,* $k \in \mathbb{N}$ *be its security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{CPA}_{\Sigma,\mathcal{A}}(k)$ *as the following probabilistic experiment.*

$$
\begin{aligned}
&\mathbf{CPA}_{\Sigma,\mathcal{A}}(k)\\
&\quad \mathrm{K} \leftarrow \mathsf{Gen}(1^k)\\
&\quad (m_0, m_1, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_{\mathrm{K}}(\cdot)}(1^k)\\
&\quad b \xleftarrow{\$} \{0,1\}\\
&\quad c \leftarrow \mathsf{Enc}_{\mathrm{K}}(m_b)\\
&\quad b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_{\mathrm{K}}(\cdot)}(\mathrm{ST}, c)\\
&\quad \textit{if } b' = b, \textit{ output } 1\\
&\quad \textit{otherwise output } 0
\end{aligned}
$$

with the restriction that the messages m_0 *and* m_1 *be such that* $|m_0| = |m_1|$*. We say that* Σ *is secure against chosen-plaintext attacks if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr\left[\,\mathbf{CPA}_{\Sigma,\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k)$$

where the probability is taken over the coins of Gen*,* Enc *and* $\mathcal{A}$*.*

While CPA-security is appropriate for many uses of encryption, it was soon realized that some applications required schemes with stronger security properties. This

motivated researchers to consider new attacks such as non-adaptive chosen-ciphertext attacks [52], where the adversary has access to a decryption oracle before the challenge message is encrypted; and adaptive chosen-ciphertext attacks [56] (CCA), where it has access to the decryption oracle before and after the target message is encrypted.

Definition 2.1.3 (CCA-security for private-key encryption)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{CCA}_{\Sigma,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{CCA}_{\mathcal{A},\Sigma}(k)$
$\quad \mathrm{K} \leftarrow \mathsf{Gen}(1^k)$
$\quad (m_0, m_1, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_\mathrm{K}(\cdot),\mathsf{Dec}_\mathrm{K}(\cdot)}(1^k)$
$\quad b \xleftarrow{\$} \{0,1\}$
$\quad c \leftarrow \mathsf{Enc}_\mathrm{K}(m_b)$
$\quad b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_\mathrm{K}(\cdot),\mathsf{Dec}_\mathrm{K}(\cdot)}(\mathrm{ST}, c)$
$\quad$ *if* $b' = b$, *output* 1
$\quad$ *otherwise output* 0

with the restriction that the messages m_0 *and* m_1 *be such that* $|m_0| = |m_1|$*, and that* $\mathcal{A}_2$ *never query its decryption oracle on its challenge ciphertext* c*. We say that* Σ *is secure against chosen-ciphertext attacks if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr\left[\,\mathbf{CCA}_{\mathcal{A},\Sigma}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen*,* Enc *and* $\mathcal{A}$*.*

Public-key encryption. Public-key encryption allows two parties who do not share a secret key to communicate privately. To use a public-key encryption scheme

a receiver begins by generating a public and private key pair. It then publishes the public key while keeping the private key secret. The sender uses the receiver's public key to encrypt messages, while the receiver uses the private key to decrypt.

Definition 2.1.4 (Public-key encryption)**.** *A public-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *consists of three polynomial-time algorithms such that,*

$(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k)$*: takes as input a security parameter* 1^k *and outputs a secret key* SK *and a public key* PK*.*

$c \leftarrow \mathsf{Enc}(\mathrm{PK}, m)$*: is a probabilistic algorithm that takes as input the public key* PK *and a message* m *from some associated message space, and outputs a ciphertext* c*. We sometimes write this as* $c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m)$*.*

$m \leftarrow \mathsf{Dec}(\mathrm{SK}, c)$*: takes as input the secret key* SK *and a ciphertext* c*, and outputs either a message* m *in the message space or a special failure symbol* $\perp$*. We sometimes write this as* $m \leftarrow \mathsf{Dec}_{\mathrm{SK}}(c)$*. We assume without loss of generality that* Dec *is deterministic.*

Similarly to the case of private-key encryption, we will always assume that public-key encryption schemes are perfectly correct in the sense that for all $k \in \mathbb{N}$, all $(\mathrm{PK}, \mathrm{SK})$ output by $\mathsf{Gen}(1^k)$, and all messages m in the message space, $\mathsf{Dec}_{\mathrm{SK}}(\mathsf{Enc}_{\mathrm{PK}}(m)) = m$.

We now define the analogues of CPA and CCA-security for public-key encryption.

Definition 2.1.5 (CPA-security for public-key encryption [40])**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$

be a public-key encryption scheme, $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{CPA}_{\Pi,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{CPA}_{\Pi,\mathcal{A}}(k)$
$(\mathrm{SK}, \mathrm{PK}) \leftarrow \mathsf{Gen}(1^k)$
$(m_0, m_1, \mathrm{ST}) \leftarrow \mathcal{A}_1(\mathrm{PK})$
$b \xleftarrow{\$} \{0,1\}$
$c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m_b)$
$b' \leftarrow \mathcal{A}_2(\mathrm{ST}, c)$
if $b' = b$, *output* 1
otherwise output 0

with the restriction that the messages m_0 *and* m_1 *be such that* $|m_0| = |m_1|$. *We say that* Π *is secure against chosen-plaintext attacks if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\left[\,\mathbf{CPA}_{\Pi,\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen, Enc *and* $\mathcal{A}$.

Definition 2.1.6 (CCA-security for public-key encryption [56])**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{CCA}_{\Pi,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{CCA}_{\Pi,\mathcal{A}}(k)$
$(\mathrm{SK}, \mathrm{PK}) \leftarrow \mathsf{Gen}(1^k)$
$(m_0, m_1, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Dec}_{\mathrm{SK}}(\cdot)}(\mathrm{PK})$
$b \xleftarrow{\$} \{0,1\}$
$c \leftarrow \mathsf{Enc}_{\mathrm{PK}}(m_b)$
$b' \leftarrow \mathcal{A}_2^{\mathsf{Dec}_{\mathrm{SK}}(\cdot)}(\mathrm{ST}, c)$
if $b' = b$, *output* 1
otherwise output 0

with the restriction that the messages m_0 and m_1 be such that $|m_0| = |m_1|$, and that $\mathcal{A}_2$ not query its decryption oracle on its challenge ciphertext c. We say that Π is secure against chosen-ciphertext attacks if for all PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\Pr\left[\,\mathbf{CCA}_{\Pi,\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen, Enc *and $\mathcal{A}$.*

Pseudo-random functions. A pseudo-random function (PRF) is an efficiently computable function family that is computationally indistinguishable from a random function. More intuitively, a PRF can be seen as a function that when evaluated on an input for the first time, returns a value that appears random (to a computationally-bounded adversary); but when evaluated on an input for a second time, returns the same value that was output the first time. PRFs were introduced by Goldreich, Goldwasser and Micali [36] and are known to exist under the assumption that one-way functions exist [44].

Definition 2.1.7 (Pseudo-random functions and permutations)**.** *Let F be an efficiently-computable keyed function, where for a fixed key K of length k we have $F_{\mathrm{K}} : \{0,1\}^{\ell_{in}(k)} \to \{0,1\}^{\ell_{out}(k)}$ with ℓ_{in}, ℓ_{out} polynomial in k. We say that F is a pseudo-random function if for all* PPT *adversaries $\mathcal{A}$,*

$$\left|\Pr\left[\,\mathcal{A}^{F_{\mathrm{K}}(\cdot)}(1^k) = 1 : \mathrm{K} \xleftarrow{\$} \{0,1\}^k\,\right] - \Pr\left[\,\mathcal{A}^{f(\cdot)}(1^k) = 1 : f \xleftarrow{\$} \mathsf{Func}[\ell_{in}(k), \ell_{out}(k)]\,\right]\right|$$

is negligible in k, and where the probabilities are over the coins of $\mathcal{A}$ and the choices

of K *and* f*. If* F *is a PRF and for each choice of* K*,* F_{K} *is an efficiently-invertible permutation, then we call* F *a pseudo-random permutation (PRP).*

Message authentication codes. A message authentication code (MAC) allows two parties that share a secret key to authenticate their communication. To authenticate a message, the sender uses the MAC keyed with the secret key to generate a tag that is associated and sent with the message. The receiver then uses the secret key to run a verification algorithm on the tag and message pair. Intuitively, a secure MAC guarantees that no adversary (that does not know the secret key) can generate message and tag pairs that verify.

Definition 2.1.8 (Message authentication code)**.** *A message authentication code* $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Ver})$ *consists of three polynomial-time algorithms such that,*

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$*: takes as input a security parameter* 1^k *and outputs a key* K*.*

$t \leftarrow \mathsf{Mac}(\mathrm{K}, m)$*: takes as input a key* K *and a message* $m \in \{0,1\}^*$ *and outputs a tag* t*. We sometimes write this as* $t \leftarrow \mathsf{Mac}_{\mathrm{K}}(m)$*. We assume, without loss of generality, that* Mac *is deterministic*

$b \leftarrow \mathsf{Ver}(\mathrm{K}, m, t)$*: is a deterministic algorithm that takes as input a key* K*, a message* $m \in \{0,1\}^*$ *, and a tag* t*; and outputs a bit* b *where a '1' indicates acceptance and a '0' indicates rejection. We sometimes write this as* $b \leftarrow \mathsf{Ver}_{\mathrm{K}}(m, t)$*.*

Notice that we define MACs to work for arbitrary-length messages. This is not

essential, but simplifies the presentation. Similarly to the case of private-key encryption, the key generation algorithm of most MACs simply output a random key of length k. When this is the case, we write $\mathsf{MAC} = (\mathsf{Mac}, \mathsf{Ver})$. We assume perfect correctness in the sense that for all $k \in \mathbb{N}$, all K output by $\mathsf{Gen}(1^k)$, and all $m \in \{0,1\}^*$, it holds that $\mathsf{Ver}_{\mathrm{K}}(m, \mathsf{Mac}_{\mathrm{K}}(m)) = 1$.

The standard notion of security for MACs, namely existential unforgeability under an adaptive chosen message attack, was adapted to the private-key setting by Bellare, Kilian and Rogaway [10] from Goldwasser, Micali and Rivest's security definition for digital signatures [41].

Definition 2.1.9 (Existential unforgeability under chosen-message attack [10])**.** *A message authentication code* $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Ver})$ *is existentially unforgeable under an adaptive chosen-message attack if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\ \mathsf{Ver}_{\mathrm{K}}(m,t) = 1 : \mathrm{K} \leftarrow \mathsf{Gen}(1^k); (m,t) \leftarrow \mathcal{A}^{\mathsf{Mac}_{\mathrm{K}}(\cdot),\mathsf{Ver}_{\mathrm{K}}(\cdot)}(1^k)\ \right] \leq \mathsf{negl}(k),$$

with the restriction that $\mathcal{A}$ *cannot query its* Mac *oracle on* m*, and where the probability is taken over the coins of* Gen *and* $\mathcal{A}$*.*

We say that a message authentication code has *unique tags* if for all $k \in \mathbb{N}$, for all K output by $\mathsf{Gen}(1^k)$, and all $m \in \{0,1\}^*$, there is a unique t such that $\mathsf{Ver}_{\mathrm{K}}(m,t) = 1$.

Chapter 3

Encrypting with a Malicious Random Number Generator

3.1 Introduction

Today, security against chosen-plaintext attacks (CPA-security) [40, 8, 47] is considered a minimal notion of security that any private-key encryption scheme deployed in practice should satisfy. It is not hard to see from Definitions 2.1.2 and 2.1.5 that any scheme secure with respect to chosen-plaintext attacks must be probabilistic. Indeed, if the encryption scheme were deterministic, then the adversary could succeed in its experiment by querying its encryption oracle on m_0 and checking whether the result is equal to its challenge.

Furthermore, it is by now well-understood how to construct CPA-secure schemes

under the assumption that the sender is able to generate a fresh set of uniformly random coins each time a message is encrypted. In practice, such coins might be generated by using a combination of randomness extractors and pseudo-random number generators (PRGs) to distill pseudo-random coins from a high-entropy source available to the sender.

The above, however, neglects the possibility that the random coins used to encrypt may sometimes be "less than perfect". For example, the sender may be using a faulty PRG that produces biased or partially predictable outputs. Or, the random source used to seed the PRG may have less entropy than originally thought. More malicious scenarios include the possibilities that an adversary may have tampered with the PRG used by the sender, or may be able to effect some control over the random source used to seed the PRG. In the most extreme case, the adversary may have physical access to the device performing the encryption (as might be the case if, e.g., encryption is carried out on a lightweight device captured by the adversary), and may then have complete control over the "random coins" that will actually be used to encrypt. We refer to such attacks as *chosen-randomness attacks*.

In this chapter, we introduce new definitions of security that offer protection against the attacks just described. Our definitions assume the worst possible case: that the randomness used by the encryption oracle is under the complete control of the adversary. In fact, the only random coins that are not under the adversary's control (other than those used to generate the key) are those that are used to encrypt

the challenge ciphertext. We note, however, that *some* assumption regarding these coins is necessary in our setting. Otherwise, if the adversary has complete control over *all* coins, then the scheme degenerates to a deterministic one that cannot be secure. In addition, our assumption that the coins used to generate the challenge ciphertext are truly random is made for simplicity, and can be relaxed by using randomness extractors and assuming only access to a high-entropy source during encryption. Our definition, then, can be viewed as offering *semantic security* for any messages that are encrypted using "good" random coins, even if the adversary is able to cause the sender to use "poor" random coins when encrypting other messages.

3.1.1 Previous Work

The most relevant prior work to ours is perhaps Rogaway's notion of nonce-based private-key encryption [57], which treats the encryption algorithm as a deterministic function of the message and a user-provided nonce. With respect to this viewpoint, it is the responsibility of the user – not the encryption algorithm – to ensure, e.g., that nonces are chosen at random. In this context, Rogaway defines a notion of security that, roughly speaking, guarantees semantic security as long as nonces never repeat. While this definition is somewhat similar to our own, we show in Section 3.2.1 that the notion considered by Rogaway is incomparable to the notion of CRA-security considered here; i.e., there are schemes satisfying his definition and not ours, and vice versa. We remark further that the motivations for our work and Rogaway's work are

very different: as argued by Rogaway [57], nonce-based security is best understood as a usability requirement, whereas we are interested in examining a stronger attack model.

Adversarial manipulation of a PRG was mentioned as motivation for our work. While there has been prior work developing forward and backward-secure PRGs [12, 6, 34], simply composing such generators with a standard CPA or CCA-secure encryption scheme does *not* defend against the attacks considered here. The reason is that these prior works consider only adversaries that *learn* the internal state of the PRG, whereas our notions consider stronger adversaries that may *control* the state of the PRG. One can therefore view our notion of CRA-security as achieving a strong variant of backward and forward-security with respect to the underlying source of randomness. In other words, our definitions guarantee that a plaintext encrypted using high-quality randomness is protected even against adversaries that can control the source after the present plaintext is encrypted (i.e., strong forward-security), or that have controlled it in the past (i.e., strong backward-security).

Work of McInnes and Pinkas [50] and Dodis *et al.* [27, 21] has also considered the security of encryption when truly random coins are not available to the sender. Although these works are superficially related to our own, the problems being considered — as well as the motivation — are very different. The work of [50, 27, 21] is unwilling to assume *any* truly random coins, even during generation of the secret key, and is interested in exploring what can be achieved in such an extreme setting.

For this reason, they are primarily concerned with information-theoretic security (although later work [26, 21] treats computational security) and do not consider security against chosen-plaintext attacks at all. In this work, in contrast, we are willing to assume that truly random coins *exist* (e.g., during key generation and, at least once, when encrypting), but are concerned that the adversary may otherwise be able to tamper with the honest user's ability to generate true random coins. We are then interested in the question of whether the analogue of CPA-security is achievable.

3.1.2 Summary of Contributions

We formally define security against chosen-randomness attacks (CRA-security), both with and without the additional presence of a decryption oracle. We refer to the latter as security against chosen-ciphertext and randomness attacks (CCRA-security). We then show two secure constructions that can be based on any block cipher. The first is a relatively simple fixed-length construction, while the second is a scheme that can encrypt arbitrary-length messages. We also show a generic transformation converting any CPA-secure scheme into a CRA-secure scheme. Finally, we propose a simple way to extend any CRA-secure scheme so as to also achieve security against chosen-ciphertext attacks.

The contents of this Chapter appear in [46].

3.2 Defining Security Against Chosen-Randomness Attacks

We now present our definitions of CRA and CCRA-security. Intuitively, CRA-security guarantees that, given a ciphertext, no polynomially-bounded adversary can recover any partial information about the plaintext, even if it has access to an encryption oracle and complete control over its source of randomness.

Notice that in both experiments the adversary is able to set the encryption oracle's random tape, but that the randomness used to generate the challenge ciphertext on the other hand is outside of the adversary's control.

Definition 3.2.1 (CRA-security)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{CRA}_{\Sigma,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{CRA}_{\Sigma,\mathcal{A}}(k)$
$\quad \mathrm{K} \leftarrow \mathsf{Gen}(1^k)$
$\quad (m_0, m_1, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot)}(1^k)$
$\quad b \xleftarrow{\$} \{0,1\}$
$\quad r \xleftarrow{\$} \{0,1\}^k$
$\quad c \leftarrow \mathsf{Enc}_\mathrm{K}(m_b; r)$
$\quad b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot)}(\mathrm{ST}, c)$
$\quad$ *if* $b' = b$, *output* 1
$\quad$ *otherwise output* 0

with the restriction that the messages m_0 *and* m_1 *be such that* $|m_0| = |m_1|$*. We say that* Σ *is secure against chosen-randomness attacks if for all* PPT *adversaries*

$\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\left[\,\mathbf{CRA}_{\Sigma,\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen, Enc *and* $\mathcal{A}$.

Notice that since $\mathcal{A}$ can choose r uniformly at random, CRA-security is at least as strong as CPA-security.

The stronger notion of CCRA-security guarantees that, given a ciphertext, no polynomially-bounded adversary can recover any partial information about the plaintext, even if it has access to both an encryption and a decryption oracle and complete control over the encryption oracle's source of randomness.

Definition 3.2.2 (CCRA-Security)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{CCRA}_{\Sigma,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{CCRA}_{\Sigma,\mathcal{A}}(k)$
$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$
$(m_0, m_1, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot),\mathsf{Dec}_\mathrm{K}(\cdot)}(1^k)$
$b \xleftarrow{\$} \{0,1\}$
$r \xleftarrow{\$} \{0,1\}^k$
$c \leftarrow \mathsf{Enc}_\mathrm{K}(m_b; r)$
$b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot),\mathsf{Dec}_\mathrm{K}(\cdot)}(\mathrm{ST}, c)$
if $b' = b$, *output* 1
otherwise output 0

with the restriction that the messages m_0 *and* m_1 *be such that* $|m_0| = |m_1|$, *and that* $\mathcal{A}_2$ *not query its decryption oracle on its challenge ciphertext* c. *We say that* Σ *is*

secure against chosen-ciphertext and randomness attacks if for all PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\left[\,\mathbf{CCRA}_{\Sigma,\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen, Enc *and* $\mathcal{A}$.

Notice that since $\mathcal{A}$ can ignore its Dec oracle, CCRA-security is at least as strong as CRA-security. Also, since $\mathcal{A}$ can choose r uniformly at random, CCRA-security is at least as strong as CCA-security.

3.2.1 Comparison to Traditional Definitions

In this Section we compare our new definitions to previous security notions for private-key encryption. We first show that CRA and CCRA-security are *strictly* stronger than the traditional notions of CPA and CCA-security. We then show that they are incomparable to the related notions of nonce-based CPA and CCA-security from [57].

Theorem 3.2.3. *CRA-security is strictly stronger than CPA-security.*

Proof. Since we already noted that CRA-security implies CPA-security, we only show that the converse is not true. Let F be a PRF, and consider the standard CPA-secure private-key encryption scheme with encryption algorithm $\mathsf{Enc}_{\mathrm{K}}(m; r) = \langle c_1, c_2 \rangle = \langle r, F_{\mathrm{K}}(r) \oplus m \rangle$; and decryption algorithm $\mathsf{Dec}_{\mathrm{K}}(c) = F_{\mathrm{K}}(r) \oplus c_2$. We claim that this scheme is not CRA-secure. To see why, note that an adversary that is given a

challenge ciphertext $c^* = \langle c_1^*, c_2^* \rangle = \langle r, F_{\mathrm{K}}(r) \oplus m_b \rangle$ can query its oracle on $\langle 0^k, r \rangle$, receiving $\langle r, F_{\mathrm{K}}(r) \rangle$, and compute $m_b \leftarrow F_{\mathrm{K}}(r) \oplus c_2^*$.

■

Theorem 3.2.4. *CCRA-security is strictly stronger than CCA-security.*

Proof. Since it is easy to see that CCRA-security implies CCA-security, we only show that the converse is not true. Let $\mathsf{MAC} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Ver})$ be a secure message authentication code with unique tags, and consider the standard CCA-secure scheme with key generation algorithm $\mathsf{Gen}(1^k)$ that outputs $\mathrm{K} = \langle \mathrm{K}_1, \mathrm{K}_2 \rangle$ such that $\mathrm{K}_1, \mathrm{K}_2 \xleftarrow{\$} \{0,1\}^k$; encryption algorithm $\mathsf{Enc}_{\mathrm{K}}(m; r) = \langle c_1, c_2, c_3 \rangle = \langle r, F_{\mathrm{K}_1}(r) \oplus m, \mathsf{Mac}_{\mathrm{K}_2}(c_1 \| c_2) \rangle$; and decryption algorithm $\mathsf{Dec}_{\mathrm{K}}(c)$ which outputs $F_{\mathrm{K}_1}(c_1) \oplus c_2$ if $\mathsf{Ver}_{\mathrm{K}_2}(c_1 \| c_2, c_3) = 1$ and $\perp$ otherwise.

To see why this scheme is not CCRA-secure, note that an adversary that is given a challenge ciphertext $c^* = \langle c_1^*, c_2^*, c_3^* \rangle = \langle r, F_{\mathrm{K}_1}(r) \oplus m_b, \mathsf{Mac}_{\mathrm{K}_2}(c_1 \| c_2) \rangle$ can query its encryption oracle with $\langle 0^k, r \rangle$ in order to receive $\langle r, F_{\mathrm{K}_1}(r), t \rangle$. It can then compute $m_b \leftarrow F_{\mathrm{K}_1}(r) \oplus c_2^*$.

■

3.2.2 Comparison to Nonce-Based Security

Nonce-based encryption [57] is a formalization of private-key encryption where the encryption algorithm is a deterministic function of a message and a nonce, and the

user is responsible for providing the nonce. In the case of CBC-mode encryption, for example, the IV would be an additional input provided to the encryption algorithm as opposed to being generated "internally". This formulation gives more flexibility with respect to how the nonce is chosen: by assuming the nonce is chosen uniformly each time the encryption algorithm is called, the standard notion of probabilistic encryption is recovered, but another option is to assume only that nonces never repeat (but are not necessarily random).

Rogaway [57] considers definitions of security for nonce-based schemes in which the adversary is given some control over the nonce that is used to encrypt at all times, i.e., both when interacting with an encryption oracle as well as when the challenge ciphertext is computed. We offer a definition in the spirit of nonce-based security [57], but we do not require that ciphertexts be indistinguishable from random strings. We note, however, that this extra requirement is irrelevant as far as the results of this thesis.

Definition 3.2.5 (NCPA-security [57])**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{NCPA}_{\Sigma,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{NCPA}_{\Sigma,\mathcal{A}}(k)$

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$

$(m_0, m_1, r, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot)}(1^k)$

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

$c \leftarrow \mathsf{Enc}(m_b; r)$

$b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot)}(\mathrm{ST}, c)$

if $b' = b$, *output* 1

otherwise output 0

with the restriction that m_0 *and* m_1 *be such that* $|m_0| = |m_1|$, *and that* $\mathcal{A}$ *never reuse the same randomness for its oracle queries and its choice of* r. *We say that* Σ *is secure against nonce-based chosen-plaintext attacks if for all* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\left[\,\mathbf{NCPA}_{\Sigma,\mathcal{A}}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen, Enc *and* $\mathcal{A}$.

Definition 3.2.6 (NCCA-security [57])**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{NCCA}_{\Sigma,\mathcal{A}}(k)$ *as the following probabilistic experiment:*

$\mathbf{NCCA}_{\Sigma,\mathcal{A}}(k)$

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$

$(m_0, m_1, r, \mathrm{ST}) \leftarrow \mathcal{A}_1^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot),\mathsf{Dec}_\mathrm{K}(\cdot)}(1^k)$

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

$c \leftarrow \mathsf{Enc}(m_b; r)$

$b' \leftarrow \mathcal{A}_2^{\mathsf{Enc}_\mathrm{K}(\cdot,\cdot),\mathsf{Dec}_\mathrm{K}(\cdot)}(\mathrm{ST}, c)$

if $b' = b$, *output* 1

otherwise output 0

with the restriction that the messages m_0 *and* m_1 *must be such that* $|m_0| = |m_1|$, *that* $\mathcal{A}$ *never reuse the same randomness for its oracle queries and its choice of* r, *and*

that $\mathcal{A}_2$ not query its decryption oracle on its challenge ciphertext c. We say that Σ is secure against nonce-based chosen-ciphertext attacks if for all PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\Pr\left[\,\mathbf{NCCA}_{\mathcal{A},\Sigma}(k) = 1\,\right] \leq \frac{1}{2} + \mathsf{negl}(k),$$

where the probability is over the coins of Gen, Enc *and $\mathcal{A}$.*

Intuitively, these definitions are incomparable to our own because:

- on one hand, we assume the adversary has *no control* over the randomness used to encrypt the challenge ciphertext, whereas Rogaway allows the adversary to have some control over the randomness even in this case.
- on the other hand, we give the adversary *full control* over the randomness used by the encryption oracle, whereas Rogaway restricts the adversary to never using the same nonce twice.

We now formally prove that the notions are incomparable.

Theorem 3.2.7. *Nonce-based CPA-security and CRA-security are incomparable.*

The theorem is a consequence of the following two lemmas.

Lemma 3.2.8. *Assuming the existence of one-way functions, there exists a private-key encryption scheme that is nonce-based CPA-secure but not CRA-secure.*

Proof. We take the standard encryption scheme used in the proof of Theorem 3.2.3. Recall that F is a pseudo-random function, which may be constructed from any one-

way function. Encryption is given by $\mathsf{Enc}_K(m;r) = \langle c_1, c_2 \rangle = \langle r, F_K(r) \oplus m \rangle$, where we treat r as a nonce, and decryption is given by $\mathsf{Dec}_K(c) = F_K(r) \oplus c_2$.

We have already shown in the proof of Theorem 3.2.3 that this scheme is not CRA-secure. On the other hand, it is not hard to see that it is nonce-based CPA-secure: since the adversary is not allowed to use the same nonce twice, it holds that the nonce r used when encrypting the challenge ciphertext is distinct from any nonce used in answering any queries to the encryption oracle. It then follows from the pseudo-randomness of F that the scheme is nonce-based CPA-secure.

■

Lemma 3.2.9. *Assuming the existence of one-way functions, there is a CRA-secure scheme that is not nonce-based CPA-secure.*

The proof of the lemma rests on the assumption that CRA-secure encryption scheme exist, but, as we show in Section 3.3, such schemes can be built out of any one-way function.

Proof. Let $\Sigma = (\mathsf{Enc}, \mathsf{Dec})$ be a CRA-secure private-key encryption scheme. We again treat the random coins used by Enc as a nonce. Define a modified encryption scheme $\Sigma' = (\mathsf{Enc}', \mathsf{Dec})$ (decryption remains unchanged) as follows:

$$\mathsf{Enc}'_K(m; r\|b) = \mathsf{Enc}_K(m; r),$$

where b is a bit and $r \in \{0,1\}^k$. It is easy to see that Σ' is not nonce-based CPA secure: an adversary can simply request to have the challenge ciphertext encrypted

using the nonce $r\|0$ and then query its encryption oracle using the (distinct) nonce $r\|1$. It is similarly easy to see that Σ' remains CRA-secure: oracle queries with respect to the modified scheme Σ' are no more powerful than oracle queries with respect to the original scheme Σ; when the challenge ciphertext is encrypted, it will be encrypted using algorithm Enc with uniform random coins.

■

We note that, similarly to the case of CRA-security and nonce-based CPA-security, there is a separation between CCRA-security and nonce-based CCA-security. The proofs are similar to those of Lemmas 3.2.8 and 3.2.9, except that Σ is replaced with a nonce-based CCA-secure scheme and a CCRA-secure scheme, respectively.

3.3 Achieving Security Against Chosen-Randomness Attacks

In this Section we propose two CRA-secure private-key encryption schemes based on PRPs. Our first construction handles only fixed-length messages, while our second construction handles messages of variable length. We then describe general a transformation from any CPA-secure scheme to a CRA-secure one.

$\mathsf{Gen}(1^k)$: sample $\mathrm{K}_1, \mathrm{K}_2 \xleftarrow{\$} \{0,1\}^k$, and output $\mathrm{K} = \langle \mathrm{K}_1, \mathrm{K}_2 \rangle$.

$\mathsf{Enc}_{\mathrm{K}}(m; r)$: output $c = \langle c_1, c_2 \rangle = \langle P_{\mathrm{K}_1}(r), F_{\mathrm{K}_2}(r) \oplus m \rangle$.

$\mathsf{Dec}_{\mathrm{K}}(c)$: compute $r \leftarrow P^{-1}_{\mathrm{K}_1}(c_1)$ and recover $m \leftarrow c_2 \oplus F_{\mathrm{K}_2}(r)$.

Figure 3.1: A fixed-length CRA-secure private-key encryptions scheme.

3.3.1 A Fixed-Length CRA-Secure Construction

Our first construction is a modification of the standard CPA-secure encryption scheme based on pseudo-random functions which we described in the proof of Theorem 3.2.3. Let P be a PRP on k-bit strings and F be a PRF mapping k-bit inputs to k-bit outputs. Our scheme is described in Figure 3.1.

Theorem 3.3.1. *If P is a pseudo-random permutation and F is a pseudo-random function, then the scheme described above is CRA-secure.*

Proof. Consider the encryption scheme $\widetilde{\Sigma} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ in which $\widetilde{\mathsf{Gen}}$ samples $p \xleftarrow{\$} \mathsf{Perm}[k,k]$ and $f \xleftarrow{\$} \mathsf{Func}[k,k]$, and $\widetilde{\mathsf{Enc}}(m; r)$ outputs the ciphertext $\langle p(r), f(r) \oplus m \rangle$. We analyze the security of $\widetilde{\Sigma}$ in an information theoretic sense and note that the CRA-security of our construction against polynomially-bounded adversaries can be derived by a standard argument.

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary making at most $q(k)$ queries to its oracle in experiment $\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k)$. Also, let r be the randomness used to generate its challenge ciphertext during this experiment, and let query be the event that at least one of $\mathcal{A}$'s

oracle queries uses randomness r. Clearly,

$$\begin{aligned}
\Pr\left[\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k)=1\right] &= \Pr\left[b'=b\right] \\
&= \Pr\left[b'=b \wedge \mathsf{query}\right] + \Pr\left[b'=b \wedge \overline{\mathsf{query}}\right] \\
&\leq \Pr\left[\mathsf{query}\right] + \Pr\left[b'=b \mid \overline{\mathsf{query}}\right].
\end{aligned}$$

The following two claims complete the proof of the theorem.

Claim. $\Pr\left[\mathsf{query}\right] \leq q(k)/2^k$.

Let query_1 and query_2 be the events that $\mathcal{A}_1$ and $\mathcal{A}_2$ submit at least one query that includes r, respectively. Then,

$$\begin{aligned}
\Pr\left[\mathsf{query}\right] &= \Pr\left[\mathsf{query}_1\right] + \Pr\left[\mathsf{query}_2 \wedge \overline{\mathsf{query}_1}\right] \\
&\leq \Pr\left[\mathsf{query}_1\right] + \Pr\left[\mathsf{query}_2 \mid \overline{\mathsf{query}_1}\right].
\end{aligned}$$

Let $q(k) = q_1(k)+q_2(k)$, where $q_1(k)$ and $q_2(k)$ are the number of queries submitted by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Since r has not been chosen when $\mathcal{A}_1$ submits its queries, the best it can do is guess, which means that $\Pr\left[\mathsf{query}_1\right] \leq q_1(k)/2^k$. Since p and f are random functions, it follows that if $\mathcal{A}_1$ did not submit any queries with r, then from $\mathcal{A}_2$'s point of view $p(r)$ and $f(r)$ are uniformly distributed. This implies that it cannot learn any information about r from its challenge ciphertext $c = \langle p(r), f(r) \oplus m_b \rangle$. Therefore, the best $\mathcal{A}_2$ can do (with respect to querying its oracle with r) is to guess. From this we have that $\Pr\left[\mathsf{query}_2 \mid \overline{\mathsf{query}_1}\right] \leq q_2(k)/2^k$, from which the claim follows.

□

$\mathsf{Gen}(1^k)$: sample $\mathrm{K}_1, \mathrm{K}_2 \xleftarrow{\$} \{0,1\}^k$ and output $\mathrm{K} = \langle \mathrm{K}_1, \mathrm{K}_2 \rangle$.

$\mathsf{Enc}_{\mathrm{K}}(m; r)$: parse m into n blocks $m = \langle m_1, \ldots, m_n \rangle$ each of length k. For $1 \leq i \leq n$ compute $c_i \leftarrow F_{\mathrm{K}_1}(r+i) \oplus m_i$, where we view r as a k-bit integer and addition is modulo 2^k. Output $c = \langle P_{\mathrm{K}_2}(r), c_1, \ldots, c_n \rangle$.

$\mathsf{Dec}_{\mathrm{K}}(c)$: compute $r \leftarrow P^{-1}_{\mathrm{K}_2}(c_0)$, and for $1 \leq i \leq n$ compute $m_i \leftarrow F_{\mathrm{K}_1}(r+i) \oplus c_i$. Output $m = \langle m_1, \ldots, m_n \rangle$.

Figure 3.2: A variable-length CRA-secure private-key encryption scheme.

Claim. $\Pr[\, b' = b \mid \overline{\mathsf{query}} \,] \leq 1/2$.

Notice that if query does not occur then $p(r)$ and $f(r)$ are both uniformly distributed so $\mathcal{A}$ cannot learn any information about b from its challenge ciphertext $c = \langle p(r), f(r) \oplus m_b \rangle$. Therefore, the best $\mathcal{A}$ can do (with respect to outputting b) is to output a bit at random.

□

■

3.3.2 A Variable-Length CRA-secure Construction

Our second construction applies a similar modification as in the previous section to the counter-mode (CTR) mode of encryption [8]. Let P be a PRP and F be a PRF as in the previous section. Our construction is described in Figure 3.2.

Theorem 3.3.2. *If P is a pseudo-random permutation and F is a pseudo-random*

function, then the scheme described above is CRA-secure.

Proof. Consider the encryption scheme $\widetilde{\Sigma} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \widetilde{\mathsf{Dec}})$ in which $\widetilde{\mathsf{Gen}}$ samples $p \xleftarrow{\$} \mathsf{Perm}[k,k]$ and $f \xleftarrow{\$} \mathsf{Func}[k,k]$, and $\widetilde{\mathsf{Enc}}(m;r)$ is defined in the natural way according to the scheme described above. We analyze the security of $\widetilde{\Sigma}$ in an information theoretic sense and note that the CRA-security of our construction against polynomially-bounded adversaries can be derived by a standard argument.

Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an adversary making at most $q(k)$ queries to its oracle in experiment $\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k)$, where the messages in these queries are at most $n = q(k)$ blocks long. Also, let $q(k)$ be an upper bound on the block-length of the messages m_0 and m_1 output by $\mathcal{A}_1$. Let r be the randomness used to generate the challenge ciphertext during this experiment, and let query be the event that at least one of $\mathcal{A}$'s oracle queries uses randomness $r' \in \Gamma \stackrel{\text{def}}{=} \{r-q+1, \ldots, r+q-1\}$. Clearly,

$$\Pr\left[\,\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k) = 1\,\right] \leq \Pr\left[\,\mathsf{query}\,\right] + \Pr\left[\,b' = b \mid \overline{\mathsf{query}}\,\right].$$

The following two claims complete the proof of the theorem.

Claim. $\Pr\left[\,\mathsf{query}\,\right] \leq (2q(k)^2 - q(k))/2^k$

Let query_1 and query_2 be the events that $\mathcal{A}_1$ and $\mathcal{A}_2$ submit at least one query with randomness in Γ, respectively. Then,

$$\begin{aligned}\Pr\left[\,\mathsf{query}\,\right] &= \Pr\left[\,\mathsf{query}_1\,\right] + \Pr\left[\,\mathsf{query}_2 \wedge \overline{\mathsf{query}_1}\,\right] \\ &\leq \Pr\left[\,\mathsf{query}_1\,\right] + \Pr\left[\,\mathsf{query}_2 \mid \overline{\mathsf{query}_1}\,\right].\end{aligned}$$

Let $q(k) = q_1(k) + q_2(k)$, where $q_1(k)$ and $q_2(k)$ are the number of queries submitted

by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Since r has not been chosen when $\mathcal{A}_1$ submits its queries, the best it can do (with respect to querying its oracle with a value in Γ) is guess. Applying a union bound to the $q_1(k)$ queries made by $\mathcal{A}_1$, we have

$$\Pr[\,\mathsf{query}_1\,] \leq \frac{2 \cdot q(k) \cdot q_1(k) - q_1(k)}{2^k}.$$

Now recall that the challenge ciphertext given to $\mathcal{A}_2$ during the experiment is of the form $\langle p(r), c_1^*, \ldots, c_n^* \rangle = \langle p(r), f(r+1) \oplus m_{b,1}, \ldots, f(r+n) \oplus m_{b,n} \rangle$ where $\langle m_{b,1}, \ldots, m_{b,n} \rangle = m_b$. Since f is a random function it follows that if $\mathcal{A}_1$ did not submit any queries with $r' \in \Gamma$, then from $\mathcal{A}_2$'s point of view $\langle c_1^*, \ldots, c_n^* \rangle$ is uniformly distributed. This means that $\mathcal{A}_2$ cannot learn any information about any value in Γ from $\langle c_1^*, \ldots, c_n^* \rangle$. Similarly, $\mathcal{A}_2$ will not learn any information about r from $p(r)$. It follows then that the best $\mathcal{A}_2$ can do (with respect to querying its oracle on a value in Γ) is to guess. Applying a union bound to the $q_2(k)$ queries made by $\mathcal{A}_2$, we have

$$\Pr[\,\mathsf{query}_2 \mid \overline{\mathsf{query}_1}\,] \leq \frac{2 \cdot q(k) \cdot q_2(k) - q_2(k)}{2^k},$$

from which the claim follows.

□

Claim. $\Pr[\,b' = b \mid \overline{\mathsf{query}}\,] \leq 1/2$.

Notice that if query does not occur then $p(r)$ and $f(r+1)$ through $f(r+n)$ are uniformly distributed. This means that $\mathcal{A}$ cannot learn any information about m_b from its challenge ciphertext $\langle p(r), f(r+1) \oplus m_{b,1}, \ldots, f(r+n) \oplus m_{b,n} \rangle$. Therefore, the best $\mathcal{A}$ can do (with respect to outputting b) is to guess.

$\mathsf{Gen}(1^k)$: compute $\mathrm{K}_1 \leftarrow \mathsf{Gen}'(1^k)$ and sample $\mathrm{K}_2 \xleftarrow{\$} \{0,1\}^k$. Output $\mathrm{K} = \langle \mathrm{K}_1, \mathrm{K}_2 \rangle$.

$\mathsf{Enc}_{\mathrm{K}}(m; r)$: compute $r' \leftarrow F_{\mathrm{K}_2}(m \| r)$, and output $c \leftarrow \mathsf{Enc}'_{\mathrm{K}_1}(m; r')$.

$\mathsf{Dec}_{\mathrm{K}}(c)$: output $m \leftarrow \mathsf{Dec}'_{\mathrm{K}_1}(c)$

Figure 3.3: A CPA-to-CRA transformation.

□

■

3.3.3 A CPA-to-CRA Transformation

Finally, we present a transformation that turns any CPA-secure private-key encryption scheme into a CRA-secure scheme. Our transformation assumes the existence of PRFs for arbitrary-length inputs. Note that these may be constructed from any one-way function, whose existence is implied by the existence of a CPA-secure encryption scheme.

Let F be a PRF mapping $2k$-bit inputs to k-bit outputs, and let $\Sigma' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a CPA-secure private-key encryption scheme. Our transformation is defined in Figure 3.3.

Theorem 3.3.3. *If F is a pseudo-random function and Σ' is CPA-secure then the scheme described above is CRA-secure.*

Proof. Consider the private-key encryption scheme $\widetilde{\Sigma} = (\widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}}, \mathsf{Dec})$ in which $\widetilde{\mathsf{Gen}}$

samples $f \xleftarrow{\$} \mathsf{Func}[2k, k]$, and $\widetilde{\mathsf{Enc}}_{\mathrm{K}}(m; r)$ is defined in the natural way according to Σ. Similarly to the previous proofs, we analyze the security of $\widetilde{\Sigma}$ and note that the CRA-security of our construction against polynomially-bounded adversaries can be derived by a standard argument.

Let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be the encryption scheme described above, and let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary that makes at most $q(k)$ queries to its oracle during a $\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k)$ experiment. Also, let r be the randomness used to generate the challenge ciphertext during the experiment, and let query be the event that at least one of $\mathcal{A}$'s oracle queries uses randomness r. Clearly,

$$\Pr\left[\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k) = 1\right] \leq \Pr[\mathsf{query}] + \Pr[b' = b \wedge \overline{\mathsf{query}}]$$

The following two claims complete the proof of the theorem.

Claim. $\Pr[\mathsf{query}] \leq q(k)/2^k$

Let query_1 and query_2 be the events that $\mathcal{A}_1$ and $\mathcal{A}_2$ submit at least one query that includes r, respectively. It follows that

$$\begin{aligned} \Pr[\mathsf{query}] &= \Pr[\mathsf{query}_1] + \Pr[\mathsf{query}_2 \wedge \overline{\mathsf{query}_1}] \\ &\leq \Pr[\mathsf{query}_1] + \Pr[\mathsf{query}_2 \mid \overline{\mathsf{query}_1}] \end{aligned}$$

Let $q(k) = q_1(k) + q_2(k)$, where $q_1(k)$ and $q_2(k)$ are the number of queries submitted by $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Since r has not been chosen when $\mathcal{A}_1$ submits its queries, the best it can do is guess, from which we have $\Pr[\mathsf{query}_1] \leq q_1(k)/2^k$.

If query does not occur, then because f is a random function it follows that

$r' = f(m||r)$ is uniformly distributed from $\mathcal{A}_2$'s point of view. This means that it cannot learn any information about r from its challenge ciphertext $c^* = \mathsf{Enc}'_{\mathrm{K}_1}(m_b; r')$, and, therefore, the best it can do (in terms of querying its oracle with r) is guess. From this we have $\Pr[\,\mathsf{query}_2 \mid \overline{\mathsf{query}_1}\,] \leq q_2(k)/2^k$.

□

Claim. $\Pr[\,b' = b \wedge \overline{\mathsf{query}}\,] \leq 1/2 + \mathsf{negl}(k)$

Let repeat be the event that at least two of $\mathcal{A}$'s queries during the experiment result in the encryption of two different plaintexts under the same randomness r'. In other words, repeat is the event that $\mathcal{A}$ receives ciphertexts of the form $c_1 = \mathsf{Enc}'_{\mathrm{K}_1}(m_1; r')$ and $c_2 = \mathsf{Enc}'_{\mathrm{K}_1}(m_2, r')$ where $m_1 \neq m_2$. Clearly,

$$\begin{aligned}\Pr[\,b' = b \wedge \overline{\mathsf{query}}\,] &= \Pr[\,b' = b \wedge \overline{\mathsf{query}} \wedge \mathsf{repeat}\,] + \Pr[\,b' = b \wedge \overline{\mathsf{query}} \wedge \overline{\mathsf{repeat}}\,] \\ &\leq \Pr[\,\mathsf{repeat}\,] + \Pr[\,b' = b \mid \overline{\mathsf{query}} \wedge \overline{\mathsf{repeat}}\,].\end{aligned}$$

We now bound each term of the previous inequality. First notice that

$$\begin{aligned}\Pr[\,\mathsf{repeat}\,] &= \Pr\left[\, f(m_1\|r') = f(m_2\|r') : f \xleftarrow{\$} \mathsf{Func}[2k, k]\,\right] \\ &\leq \frac{1}{2^k},\end{aligned}$$

where the inequality follows from the fact that $m_1 \neq m_2$.

To bound the second term, we argue that if neither query nor repeat occur, then $\mathcal{A}$'s view during the $\mathbf{CRA}_{\widetilde{\Sigma},\mathcal{A}}(k)$ experiment is identical to the view it would have

during a $\mathbf{CPA}_{\widetilde{\Sigma},\mathcal{A}}(k)$ experiment. Indeed, if query does not occur then from $\mathcal{A}$'s point of view r, and therefore $r' = f(m\|r)$, is uniformly distributed. In other words, the randomness used to generate the challenge ciphertext is uniformly distributed. In addition, if repeat does not occur then it follows that for every (distinct) query that $\mathcal{A}$ makes, the ciphertext it receives will be generated using a new uniformly distributed string. It follows then that

$$\begin{aligned} \Pr\left[\, b' = b \mid \overline{\mathsf{query}} \wedge \overline{\mathsf{repeat}} \,\right] &= \Pr\left[\mathbf{CPA}_{\widetilde{\Sigma},\mathcal{A}}(k) = 1\right] \\ &\leq \frac{1}{2} + \mathsf{negl}(k) \end{aligned}$$

where the inequality follows from the fact that $\widetilde{\Sigma}$ is CPA-secure.

□

■

3.4 Achieving Security against Chosen-Ciphertext and Randomness Attacks

We now show that the standard "encrypt-then-MAC" transformation [11] from CPA-secure schemes to CCA-secure ones also works in our setting. Let $\mathsf{MAC} = (\mathsf{Mac}, \mathsf{Ver})$ be a secure unique MAC, and let $\Sigma' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a CRA-secure encryption scheme. Our transformation is described in Figure 3.4.

$\mathsf{Gen}(1^k)$: compute $\mathrm{K}_1 \leftarrow \mathsf{Gen}'(1^k)$ and sample $\mathrm{K}_2 \leftarrow \{0,1\}^k$. Output $\mathrm{K} = \langle \mathrm{K}_1, \mathrm{K}_2 \rangle$.

$\mathsf{Enc}_{\mathrm{K}}(m; r)$: compute $c_1 \leftarrow \mathsf{Enc}'_{\mathrm{K}_1}(m; r)$ and $c_2 \leftarrow \mathsf{Mac}_{\mathrm{K}_2}(c_1)$. Output $c = \langle c_1, c_2 \rangle$.

$\mathsf{Dec}_{\mathrm{K}}(c)$: if $\mathsf{Ver}_{\mathrm{K}_2}(c_1, c_2) = 1$ then output $m \leftarrow \mathsf{Dec}'_{\mathrm{K}_1}(c_1)$. Otherwise output $\perp$.

Figure 3.4: A CRA-to-CCRA transformation.

Theorem 3.4.1. *If* MAC *is a secure message authentication code with unique tags and if* Σ' *is CRA-secure, then the scheme described above is CCRA-secure.*

Proof. Let Σ be the scheme described above, and let $\mathcal{A}$ be a PPT adversary attacking Σ in a $\mathbf{CCRA}_{\Sigma,\mathcal{A}}$ experiment. Let query be the event that $\mathcal{A}$ submits a decryption query $\langle c, t \rangle$ to its decryption oracle such that $\mathsf{Dec}_{\mathrm{K}}(c, t) \neq \perp$ and such that $\langle c, t \rangle$ was not the result of a previous encryption query. Clearly,

$$\Pr[\,\mathbf{CCRA}_{\Sigma,\mathcal{A}}(k) = 1\,] \;\leq\; \Pr[\,\mathsf{query}\,] + \Pr[\,b' = b \mid \overline{\mathsf{query}}\,].$$

The following two claims complete the proof.

Claim. $\Pr[\,\mathsf{query}\,] \leq \mathsf{negl}(k)$.

We show that if there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\Pr[\,\mathsf{query}\,]$ is non-negligible in k, then there exists a PPT adversary $\mathcal{B}$ that can win the existential unforgeability experiment against MAC with non-negligible probability.

Consider the adversary $\mathcal{B}$ that, given 1^k and oracle access to $\mathsf{Mac}_{\mathrm{K}}(\cdot)$ and $\mathsf{Ver}_{\mathrm{K}}(\cdot,\cdot)$ begins by generating an encryption key $\mathrm{K}_1 \leftarrow \mathsf{Gen}'(1^k)$ and runs $\mathcal{A}_1(1^k)$ as follows:

Given an encryption query $e = \langle m, r \rangle$, $\mathcal{B}$ computes $c \leftarrow \mathsf{Enc}'_{\mathrm{K}_1}(m; r)$ and queries its own Mac oracle with c, receiving t. Finally, it returns the ciphertext $\langle c, t \rangle$ to $\mathcal{A}_1$.

Given a decryption query $d = \langle c, t \rangle$, $\mathcal{B}$ queries its $\mathsf{Ver}_{\mathrm{K}}$ oracle with c and t. If the oracle returns 1 then it computes and returns $m \leftarrow \mathsf{Dec}'_{\mathrm{K}_1}(c)$ to $\mathcal{A}_1$; otherwise it returns $\perp$. Adversary $\mathcal{B}$ stores all of $\mathcal{A}_1$'s decryption queries. After polynomially many queries, $\mathcal{A}_1$ outputs $(m_0, m_1, \textsc{St})$.

$\mathcal{B}$ samples $b \xleftarrow{\$} \{0,1\}$ and computes $c^* \leftarrow \mathsf{Enc}'_{\mathrm{K}_1}(m_b)$, and queries its oracle to receive $t^* \leftarrow \mathsf{Mac}_{\mathrm{K}}(c^*)$. It then runs $\mathcal{A}_2(\textsc{St}, \langle c^*, t^* \rangle)$, and answers its queries as before. After polynomially many queries, $\mathcal{A}_2$ outputs a bit b' and halts. Let $q(k)$ be the number of decryption queries made by $\mathcal{A}$. If query has occurred by the end of the experiment (note that $\mathcal{B}$ can determine if this happens), then $\mathcal{B}$ outputs the decryption query for which this occurred.

It remains to analyze $\mathcal{B}$'s success probability. Notice that $\mathcal{B}$ will succeed in its unforgeability experiment if query occurs. Since $\mathcal{A}$'s view is identical to the view it would have in a $\mathbf{CCRA}_{\Sigma,\mathcal{A}}(k)$ experiment, the claim follows from our original assumption about $\mathcal{A}$.

□

Claim. $\Pr[\, b' = b \mid \overline{\mathsf{query}} \,] \leq 1/2 + \mathsf{negl}(k)$

We show that if there exists a PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that

$$\Pr[\, b' = b \mid \overline{\mathsf{query}} \,] \geq 1/2 + \varepsilon(k)$$

where $\varepsilon(k)$ is non-negligible, then there exists a PPT adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that succeeds in a $\mathbf{CRA}_{\Sigma',\mathcal{B}}(k)$ experiment also with non-negligible probability.

Consider $\mathcal{B}_1$ that, given 1^k, and oracle access to $\mathsf{Enc}'_{\mathrm{K}}(\cdot;\cdot)$ begins by generating a key $\mathrm{K}_2 \leftarrow \{0,1\}^k$ and runs $\mathcal{A}_1(1^k)$ as follows:

> Given an encryption query $e = \langle m, r \rangle$, $\mathcal{B}_1$ queries its oracle with $\langle m, r \rangle$ in order to receive c. It then computes $t \leftarrow \mathsf{Mac}_{\mathrm{K}_2}(c)$, and returns the ciphertext $\langle c, t \rangle$ to $\mathcal{A}_1$. It stores the tuple $\langle c, t, m \rangle$ in a table T.
>
> Given a decryption query $d = \langle c, t \rangle$, $\mathcal{B}_1$ looks up the pair $\langle c, t \rangle$ in its table and returns the corresponding plaintext m. If the pair $\langle c, t \rangle$ is not in T, then it returns $\perp$. After polynomially many queries, $\mathcal{A}_1$ outputs (m_0, m_1, St), which $\mathcal{B}_1$ also outputs.

Given a challenge ciphertext $c^* \leftarrow \mathsf{Enc}'_{\mathrm{K}}(m_b)$ and St, $\mathcal{B}_2$ computes $t^* \leftarrow \mathsf{Mac}_{\mathrm{K}_2}(c^*)$ and runs $\mathcal{A}_2(\mathrm{St}, \langle c^*, t^* \rangle)$ answering its oracle queries as before. After polynomially-many more queries, $\mathcal{A}_2$ outputs a bit b' which $\mathcal{B}$ outputs as well.

It remains to analyze $\mathcal{B}$'s success probability. First, notice that $\mathcal{B}_1$ can answer $\mathcal{A}_1$'s encryption queries perfectly. Furthermore, if query does not occur, then the only valid decryption queries $\mathcal{A}$ makes are for ciphertexts that were the result of previous

encryption queries. In this case (i.e., conditioned on $\overline{\mathsf{query}}$), $\mathcal{B}$ will also correctly answer all of $\mathcal{A}$'s decryption queries (using its table). It follows then that conditioned on $\overline{\mathsf{query}}$, the view of $\mathcal{A}$ will be identical to its view during a $\mathbf{CCRA}_{\Sigma,\mathcal{A}}(k)$ experiment. It follows then from the description of $\mathcal{B}$ that,

$$\begin{aligned}\Pr\left[\,\mathbf{CRA}_{\Sigma',\mathcal{B}}(k)=1\,\right] &\geq \Pr\left[\,b'=b \mid \overline{\mathsf{query}}\,\right] \\ &\geq \frac{1}{2}+\varepsilon(k),\end{aligned}$$

where the second inequality follows from our original assumption about $\mathcal{A}$.

□

■

3.5 Conclusions

In this chapter, we introduced and formally defined two notions of security for private-key encryption. The first, CRA-security guarantees that if a message is encrypted using a truly random key and a truly random source of bits, the message will be protected even against an adversary that will control the source in the future, and one that controlled it in the past. The second, CCRA-security, provides the same guarantee even if the adversary has access to a decryption oracle.

In addition, we also show how to construct CRA-secure schemes for fixed-length and for arbitrary-length messages from any one-way function, and give two generic

transformations converting any CPA-secure scheme into a CRA-secure scheme, and any CRA-secure scheme into a CCRA-secure scheme.

While our constructions are computationally efficient, they do require roughly twice as much randomness as the CPA-secure constructions they are based on. One interesting question is whether this increase in randomness complexity is necessary or whether CRA and CCRA-secure schemes can be built with the same amount of randomness as CPA and CCA-secure schemes. Other interesting directions for future work include whether a similar type of security can be achieved for other cryptographic primitives such as zero-knowledge proofs or oblivious transfer.

Chapter 4

Authenticating Data Stored on an Unreliable Server

4.1 Introduction

At first glance, proving possession of data might seem like a simple problem which can be solved using known cryptographic primitives like collision-resistant hash functions. As an example, the owner could store the hash of the data and ask the server to return a copy of the data so that it can verify its integrity. Unfortunately, there are many practical settings in which such a simple approach will not work. If the bandwidth available between the client and the server is limited, then transferring large amounts of data might be infeasible. Also, if the client is computationally limited (e.g., in the case of a PDA or a smart-phone) then it might not be able to process

the entire data collection. Finally, if the owner has limited storage – which, after all, is the motivation for outsourcing storage to begin with – then it might be impossible for it to maintain a copy of the data for verification purposes.

Recently, two approaches to the problem of proving possession of data were put forth [4, 45]. While these works address the same problem, the approaches and techniques used are very different. The first approach, referred to as a *proof of retrievability* (POR), is based on symmetric-key techniques and enables a server to prove to an owner that it possesses enough of the original data to allow for its efficient retrievability (i.e., in polynomial-time). On the other hand, the second approach, referred to as a *proof of data possession* (PDP), is based on public-key techniques and allows a server to prove to anyone with the owner's public-key that it possesses the original file.

Both PORs and PDPs are proof systems executed between a prover and a verifier that enables the prover to convince the verifier that he is in possession of a file F. They are generally composed of two phases: a *setup* phase where the owner of the file encodes the file and sends it to the prover; and a *challenge* phase where a verifier engages in an interactive protocol with the prover to determine if it indeed possesses the file. If only the owner is allowed to verify possession, then the system is *privately verifiable*. If, on the other hand, the verifier can be any party that possesses the owner's public-key then the system is *publicly verifiable*.

Clearly, as previously mentioned, if we allow $O(|F|)$ communication complexity,

then the verifier can store a hash of the file and the prover can simply send F as a proof. Alternatively, if the verifier is willing to store a commitment of F, then the prover can execute a zero-knowledge proof of knowledge (PoK) for F. A PoK allows a prover to convince a verifier that is in possession of a commitment, that it knows the secret under the commitment. In addition, this is done while guaranteeing that the verifier does not learn any information about the secret. For our purposes, however, where files can be very large we are interested in proof systems with communication complexity and storage at the verifier that is $O(1)$. Such systems are referred to as *compact*.

In this chapter, we present an explicit connection between proofs of knowledge and PDP systems. Concretely, we show how to compile certain 3-round public-coin PoK into PDP systems with $O(1)$ size proofs. Due to their round-efficiency these protocols are widely used in practice and many efficient constructions are known. We use our compiler on a variation of Schnorr's protocol for proving knowledge of discrete logarithms [58] to generate a privately verifiable compact PDP system. The resulting construction is the first construction based on the hardness of computing discrete logarithms, is efficient, supports an unlimited number of proofs, and works on public data.

4.1.1 Previous Work

The problem of proving data possession has been considered in the past, but previous techniques based on RSA [25, 32], erasure-codes [59, 48], or homomorphic hash functions [48, 60, 64] all incur communication complexity or client storage that is linear in the size of the data.

As mentioned, recent work on proofs of data possession include Juels and Kaliski's POR systems and Ateniese *et al.* 's PDP systems. The scheme presented in [4] is based on public-key techniques that allow any verifier in possession of the public-key to interact with the server in order to verify data possession. The interaction can be repeated any number of times, each time resulting in a fresh proof. The construction in [4] requires little extra storage for the owner and $O(1)$ communication complexity for each verification.

The POR scheme presented in [45] uses special blocks (called *sentinels*) hidden among other blocks in the data. During the verification phase, the client asks for random blocks and checks whether the sentinels are intact. If the server modifies or deletes parts of the data, then sentinels should be affected with a certain probability. This approach, however, requires that sentinels are indistinguishable from other regular blocks, which requires that the data be encrypted ahead of time (i.e., before outsourcing). Thus, unlike the PDP scheme proposed in [4] and the constructions we provide in this work, the constructions in [45] cannot be used for public data. In addition, the number of queries is limited and fixed a-priori. This is because sentinels,

and their position within the database, are revealed to the server after each query. Recently, and concurrent with our work, Shacham and Waters [61] presented new POR constructions that are compact. The first is based on pseudo-random functions and is privately verifiable, whereas the second is publicly verifiable and is based on BLS signatures [20] in bilinear groups.

Related to PORs are sub-linear authenticators considered by Naor and Rothblum [51], and based on earlier work on memory checking [16, 17]. A sub-linear authenticator encodes a file in such a way that its integrity can be verified without reading the entire encoding. To achieve this a small amount of secret storage is necessary. In addition, each verification requires the client to re-encode the file.

Proofs of knowledge were introduced by Goldwasser, Micali and Rackoff [41] and later formalized in a series of works [28, 63, 29] culminating with the now standard definition of Bellare and Goldreich [9]. There are a many classes of PoKs, but here we focus on so-called sigma-protocols, which were formalized by Cramer *et al.* [23]. These are 3-round public-coin PoKs where the prover sends the first and third messages and the verifier sends a random challenge. The most well-known examples of sigma-protocols are Schnorr's protocol for proving knowledge of a discrete logarithm [58], and the Guillou-Quisquater protocol for proving knowledge of RSA roots [43].

4.1.2 Summary of Contributions

We propose a compiler that transforms any *homomorphically verifiable* sigma-protocol into PDP system with constant-size proofs. Intuitively, a protocol is homomorphically verifiable if it has a verification algorithm for which verifying a "sum" of interactions (in section 4.2 we make precise what we mean by a sum) is equivalent to verifying each interaction separately. We show that if the protocol is *partially* statistical honest-verifier zero-knowledge (SHVZK) – which only guarantees that part of the transcript of an interaction not leak any partial information about the secret – then it yields a privately verifiable PDP system.

Finally, we propose a simple partial SHVZK protocol based on the Schnorr protocol which compiles into a privately verifiable *compact* PDP system.

The contents of this Chapter constitute preliminary work, joint with G. Ateniese and G. Tsudik.

4.2 Definitions

4.2.1 Proofs of Data Possession

Figure 4.1 describes how a PDP system works at a high-level. First, the verifier generates a public and private-key pair. It then encodes the file and hands the encoding, together with the public-key, to the prover. The encoding process also produces some state information. If the system is privately verifiable then the state is

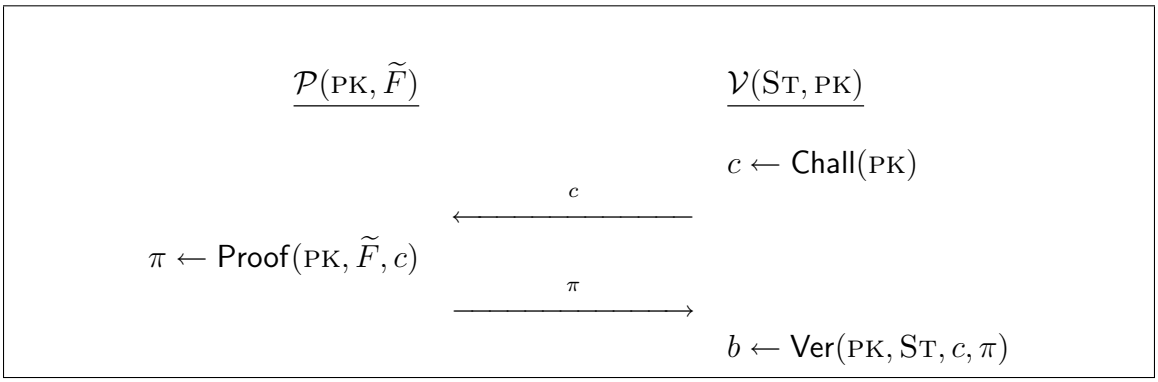

Figure 4.1: A proof of data possession system

kept private, but if the system is publicly verifiable then it is made public. At a later time when a verifier wishes to check for the availability of the file, it generates and sends a challenge to the prover who responds with a proof of data possession. The verifier then uses the public-key and the state information to verify that the proof is correct.

In the following, we provide a more formal definition of a PDP system. Although introduced by Ateniese *et al.* in [4], we follow the definitional approach of Juels and Kaliski from [45]. In particular, while the original definition of a PDP system was split into two components: a PDP scheme and a PDP system, here we collapse these into a single definition. Also, we choose to formulate the Tag algorithm from [4] as an Encode algorithm.

Definition 4.2.1 (Proof of data possession system)**.** *A PDP system between a* PPT *prover* $\mathcal{P}$ *and a* PPT *verifier* $\mathcal{V}$ *is a tuple of five polynomial-time algorithms* $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Chall}, \mathsf{Proof}, \mathsf{Ver})$ *such that,*

$(\text{PK}, \text{SK}) \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic algorithm that is run by the client to set up the scheme. It takes as input a security parameter* k*, and outputs a public and private key pair* (PK, SK)*.*

$(\text{ST}, \widetilde{F}) \leftarrow \mathsf{Encode}(\text{PK}, \text{SK}, F)$*: is a probabilistic algorithm that is run by the client in order to encode the file. It takes as input the public and secret keys* PK *and* SK*, and a* $\mathsf{poly}(k)$*-length file* F*; and outputs an encoded file* $\widetilde{F}$*, and a string* ST*. In a publicly verifiable system,* ST *can be made public, whereas in a privately verifiable system it must be kept private.*

$c \leftarrow \mathsf{Chall}(\text{PK})$*: is a probabilistic algorithm that is run by the client to generate a verification challenge for the server. It takes as input the public key* PK *, and outputs a challenge* c*.*

$\pi \leftarrow \mathsf{Proof}(\text{PK}, \widetilde{F}, c)$*: is a deterministic algorithm that takes as input the public key* PK*, an encoded file* $\widetilde{F}$ *and a challenge* c*, and outputs a proof* π*.*

$b \leftarrow \mathsf{Ver}(\text{PK}, \text{ST}, c, \pi)$*: is a deterministic algorithm that takes as input the public key* PK*, the state* ST*, a challenge* c*, and a proof* π*. It outputs a bit* b*, where '1' indicates acceptance and '0' indicates rejection. If* Ver *outputs '1' on proof* π*, then we say that* π *is* valid*.*

To be useful, a PDP system must satisfy two requirements: completeness and soundness. Intuitively, completeness requires that if a prover possesses the file F and

follows the protocol, then the verifier will accept with certainty. This intuition is formalized in the following definition.

Definition 4.2.2 (Completeness)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Chall}, \mathsf{Proof}, \mathsf{Ver})$ *be a PDP system,* $k \in \mathbb{N}$ *be the security parameter and* F *be a file. We define* $\mathbf{Comp}_{\Pi,F}(k)$ *as the following probabilistic experiment:*

$\mathbf{Comp}_{\Pi,F}(k)$
$(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k)$
$(\mathrm{ST}, \widetilde{F}) \leftarrow \mathsf{Encode}(\mathrm{PK}, \mathrm{SK}, F)$
$c \leftarrow \mathsf{Chall}(\mathrm{PK})$
$\pi \leftarrow \mathsf{Proof}(\mathrm{PK}, \widetilde{F}, c)$
$b \leftarrow \mathsf{Ver}(\mathrm{PK}, \mathrm{ST}, c, \pi)$
output b

We say that Π *is complete if for all polynomial-length strings* F*,*

$$\Pr\left[\,\mathbf{Comp}_{\Pi,F}(k) = 1\,\right] = 1,$$

where the probability is over the coins of Gen*,* Encode*,* Chall*, and* Proof*.*

The main security notion for a PDP system is *soundness* which, informally, guarantees that if the verifier accepts then the prover indeed "possesses" the entire file F. As discussed in [4, 45] this intuition can be formalized using the notion of a knowledge extractor which was introduced in the context of proofs of knowledge [31, 28, 9]. Here, we follow the definitional approaches of Juels and Kaliski [45] and Bellare and Goldreich [9], though our definition seems weaker than the one in [45]. The following definition is for privately verifiable systems. In particular, notice that the adversary is not given the state ST as input.

Definition 4.2.3 (Soundness for privately verifiable PDPs)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Chall}, \mathsf{Proof}, \mathsf{Ver})$ *be a privately verifiable PDP system,* $k \in \mathbb{N}$ *be the security parameter,* r *be a string, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be an adversary. We define* $\mathbf{Setup}_{\mathcal{A}_1,\Pi}(k)$*,* $\mathbf{Prove}_{\mathcal{A}_2,\Pi}[r](k)$ *and* $\mathbf{Ext}_{\Pi,\mathcal{A}_2,\mathcal{K}}[r](k)$ *as the following probabilistic experiments:*

$\mathbf{Setup}_{\Pi,\mathcal{A}_1}(k)$
- $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathsf{Gen}(1^k)$
- $(F, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_1^{\mathsf{Encode}(\mathrm{PK},\mathrm{SK},\cdot)}(\mathrm{PK})$
- $(\mathrm{ST}, \widetilde{F}) \leftarrow \mathsf{Encode}(\mathrm{PK}, \mathrm{SK}, F)$
- *let* $\mathrm{E} = (\mathrm{PK}, F, \widetilde{F}, \mathrm{ST}, \mathrm{ST}_{\mathcal{A}})$
- *output* E

$\mathbf{Prove}_{\Pi,\mathcal{A}_2}[r](k)$
- $\mathrm{E} \leftarrow \mathbf{Setup}_{\Pi,\mathcal{A}_1}(k)$
- $c \leftarrow \mathsf{Chall}(\mathrm{PK})$
- $\pi \leftarrow \mathcal{A}_2(\mathrm{ST}_{\mathcal{A}}, \widetilde{F}, c; r)$
- $b \leftarrow \mathsf{Ver}(\mathrm{PK}, \mathrm{ST}, c, \pi)$
- *output* b

$\mathbf{Ext}_{\Pi,\mathcal{A}_2,\mathcal{K}}[r](k)$
- $\mathrm{E} \leftarrow \mathbf{Setup}_{\Pi,\mathcal{A}_1}(k)$
- $F' \leftarrow \mathcal{K}^{\mathcal{A}_2(\mathrm{ST}_{\mathcal{A}},\widetilde{F},\cdot;r)}(\mathrm{PK}, \mathrm{ST})$
- *if* $F' = F$ *output* 1
- *otherwise output* 0

We say that Π *is sound if there exists a probabilistic expected polynomial-time knowledge extractor* $\mathcal{K}$ *such that for all* PPT $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, for all* $r \in \{0,1\}^*$*, if* $\mathcal{A}_2$ *outputs a valid proof with noticeable probability in* k *over the challenge space, then*

$$\Pr\left[\,\mathbf{Ext}_{\Pi,\mathcal{A}_2,\mathcal{K}}[r](k) = 1\,\right] \geq \Pr\left[\,\mathbf{Prove}_{\Pi,\mathcal{A}_2}[r](k) = 1\,\right] - \mathsf{negl}(k).$$

Soundness for publicly verifiable systems can be defined analogously, except that the adversary $\mathcal{A}_2$ is provided with ST in the $\mathbf{Prove}_{\Pi,\mathcal{A}_2}[r](k)$ and $\mathbf{Ext}_{\Pi,\mathcal{K},\mathcal{A}_2}[r](k)$ experiments.

A few words about this definition are in order. We refer to the values $\text{E} = (\text{PK}, F, \widetilde{F}, \text{ST}, \text{ST}_{\mathcal{A}})$ output by the $\mathbf{Setup}_{\Pi,\mathcal{A}_1}(k)$ experiment as the "environment". Intuitively, what the definition guarantees is that for most adversarially generated environments, if the adversary $\mathcal{A}_2$ is able to output a valid proof for a noticeable fraction of the possible challenges, then $\mathcal{A}_2$ indeed "knew" the file F. We formalize the intuitive notion of $\mathcal{A}_2$ "knowing" F by describing a knowledge extractor $\mathcal{K}$ that has the ability to recover F from oracle access to $\mathcal{A}_2$ in polynomial-time.

Notice, however, that $\mathcal{K}$ is given the state information ST. This makes the interpretation of Definition 4.2.3 slightly more complex than the intuition given above. Indeed, if ST is a *private* value, as it is when Π is privately verifiable, then access to ST gives the extractor an advantage over the adversary (i.e., the server). This means that the existence of a knowledge extractor implies that $\mathcal{A}_2$ possesses enough knowledge about F so that one could recover it *if* one knows ST. In particular, this *does not* necessarily imply that $\mathcal{A}_2$ "knows" F explicitly. More intuitively, this guarantees that if a server is able to generate a valid proof, then anyone with knowledge of ST can retrieve F from the server even though the server itself might not be able to.

On the other hand, if ST is public the extractor does not have any advantage over the adversary. Therefore, the existence of a knowledge extractor implies that $\mathcal{A}_2$ knows F *explicitly*. In other words if the server is able to generate a valid proof then anyone can retrieve F from the server, even the server itself.

4.2.2 Sigma-Protocols

A proof of knowledge for a binary **NP**-relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ enables a prover that is given $(x, w) \in R$ to prove his knowledge of w to a verifier that knows x. A sigma-protocol is a 3-round public-coin PoK that has the following structure: the prover sends the first message to commit to some randomness; the second message is sent by the verifier and constitutes a random challenge for the prover; the third message is sent by the prover and is a response to the challenge. Finally, given the transcript of the protocol, the verifier decides whether to accept or not.

Definition 4.2.4 (Sigma-protocol)**.** *A sigma-protocol for a binary* **NP***-relation R is a three move protocol between a* PPT *prover $\mathcal{P}$ and a* PPT *verifier $\mathcal{V}$. The protocol consists of a tuple of four polynomial-time algorithms $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Ver})$ such that,*

$(\text{CMT}, r) \leftarrow \mathsf{Comm}(x)$*: is a probabilistic algorithm that is run by the prover $\mathcal{P}$ to generate the first message. It takes as input the common input $x \in \{0,1\}^*$, and outputs a random string r and a commitment to r,* CMT*.*

$\text{CH} \leftarrow \mathsf{Chall}(1^{|x|})$*: is a probabilistic algorithm that is run by the verifier $\mathcal{V}$ to generate the second message. It takes as input the common input x, and outputs a random challenge* CH *from some associated challenge space.*

$\text{RSP} \leftarrow \mathsf{Resp}(x, w, \text{CMT}, \text{CH})$*: is a probabilistic algorithm that is run by the prover $\mathcal{P}$ to generate the third message. It takes as input the common input x, a witness*

w, a commitment CMT *and a challenge* CH*, and outputs a response* RSP *from some associated response space.*

$b \leftarrow \mathsf{Ver}(x, \text{CMT}, \text{CH}, \text{RSP})$*: is a deterministic algorithm that is run by the verifier $\mathcal{V}$ to decide whether to accept the interaction. It takes as input the common input x, a commitment* CMT*, a challenge* CH*, and a response* RSP*. It outputs a bit b, where '1' indicates acceptance and '0' indicates rejection.*

Like any proof system a sigma-protocol is only useful if it satisfies a notion of completeness and soundness. Completeness, described in Definition 4.2.5 below, guarantees that if an honest prover (i.e., one that follows the protocol) indeed "knows" a witness w such that $(x, w) \in R$, then an honest verifier will be convinced.

Definition 4.2.5 (Completeness)**.** *Let $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Dec})$ be a sigma-protocol for a binary* **NP***-relation R that is hard on average, and $k \in \mathbb{N}$ be the security parameter. We define $\mathbf{Comp}_{\Sigma}(k)$ as the following probabilistic experiment:*

$\mathbf{Comp}_{\Sigma}(k)$
$(x, w) \leftarrow \mathsf{Gen}_R(1^k)$
$(\text{CMT}, r) \leftarrow \mathsf{Comm}(x)$
$\text{CH} \leftarrow \mathsf{Chall}(1^k)$
$\text{RSP} \leftarrow \mathsf{Resp}(x, w, \text{CMT}, \text{CH})$
$b \leftarrow \mathsf{Dec}(x, \text{CMT}, \text{CH}, \text{RSP})$
output b

We say that Σ is complete if,

$$\Pr\left[\,\mathbf{Comp}_{\Sigma}(k) = 1\,\right] = 1,$$

where the probability is over the coins of Gen_R, Comm *and* Chall.

Soundness, on the other hand, guarantees that if the verifier accepts an interaction with a prover, then the prover indeed "knows" a witness w such that $(x, w) \in R$. The soundness of a sigma-protocol can be defined in a variety of ways (see [9] for a discussion), but is typically formalized using the notion of special soundness [23]. Informally, special soundness requires that one be able to efficiently "extract" a witness from the transcripts of two instances of the protocol that use the same commitment but different challenges.

In the next definition we introduce a slightly stronger variant of special soundness which requires that the witness be extractable from only the challenges and the responses of the interactions (i.e., without seeing the commitment). We refer to this as *strong special soundness*, and note that it implies special soundness. Also, most known sigma-protocols, such as the Schnorr and Guillou-Quisquater protocols, satisfy it.

Definition 4.2.6 (Strong special soundness)**.** *A sigma-protocol* $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Dec})$ *for a binary* **NP***-relation satisfies the strong special soundness property if for all* $x \in \{0,1\}^*$*, and all pairs of accepting transcripts* $(\text{CMT}, \text{CH}, \text{RSP})$ *and* $(\text{CMT}, \text{CH}', \text{RSP}')$ *on* x*, where* $\text{CH} \neq \text{CH}'$*, there exists a* PPT *knowledge extractor* $\mathcal{K}$ *such that*

$$\Pr\left[\,(x, w') \in R : w' \leftarrow \mathcal{K}(x, \text{CH}, \text{CH}', \text{RSP}, \text{RSP}')\,\right] \geq 1 - \mathsf{negl}(|x|)$$

where the probability is over the coins of $\mathcal{K}$.

Unlike PDPs, sigma-protocols must also be "zero-knowledge" in the sense that the verifier must not learn any partial information about the witness [41]. Of course there are many ways of formalizing this intuition, but in many instances the notion of *special honest-verifier zero-knowledge* (SHVZK) is sufficient. SHVZK is a relaxation of zero-knowledge that only requires the protocol to protect the witness from verifiers that do not deviate from it.

For the purpose of building privately verifiable PDP systems, however, we will see that a weaker notion than SHVZK is sufficient. We refer to this notion as *partial* special honest-verifier zero-knowledge, since it only requires that the last two messages of the interaction be "secure".

Definition 4.2.7 (Partial special honest-verifier zero-knowledge)**.** *A sigma-protocol* $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Dec})$ *for a binary* **NP***-relation* R *is partial SHVZK if there exists a* PPT *simulator* $\mathcal{S}$ *which on input* $x \in L_R$ *and a random challenge* CH*, outputs part of an accepting transcript* $(\text{CH}', \text{RSP}')$ *with the same distribution as an honest prover given* (x, w)*, where* $w \in W_R(x)$*, and a verifier given* x.

Clearly, SHVZK implies partial SHVZK. Also, it is easy to show that both strong special soundness and partial SHVZK are preserved under sequential composition.

4.2.3 Unforgeable Sigma-Protocols

We now formalize the notion of security we need from a sigma-protocol in order to build a PDP system. Intuitively we will require that, given several accepting partial transcripts of the protocol, no adversary can generate a new challenge/response pair that is accepting. We refer to this notion as *unforgeability* and we show in Theorem 4.2.9 that it is implied by partial SHVZK and strong special soundness.

Definition 4.2.8 (Unforgeability)**.** *Let* $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Dec})$ *be sigma-protocol for a binary* **NP***-relation* R *that is hard on average,* $k \in \mathbb{N}$ *be the security parameter,* $n \in \mathbb{N}$*, and* $\mathcal{A}$ *be an adversary. We define* $\mathbf{Unf}_{\Sigma,\mathcal{A},n}(k)$ *as the following probabilistic experiment:*

$$
\begin{aligned}
&\mathbf{Unf}_{\Sigma,\mathcal{A},n}(k)\\
&\quad (x, w) \leftarrow \mathsf{Gen}_R(1^k)\\
&\quad \textit{for } 1 \le i \le n,\\
&\qquad (\mathrm{CMT}_i, r) \leftarrow \mathsf{Comm}(x)\\
&\qquad \mathrm{CH}_i \leftarrow \mathsf{Chall}(1^k)\\
&\qquad \mathrm{RSP}_i \leftarrow \mathsf{Resp}(x, w, \mathrm{CMT}_i, \mathrm{CH}_i)\\
&\quad (i, \mathrm{CH}'_i, \mathrm{RSP}'_i) \leftarrow \mathcal{A}(x, \{(\mathrm{CH}_i, \mathrm{RSP}_i)\}_i)\\
&\quad b \leftarrow \mathsf{Dec}(x, \mathrm{CMT}_i, \mathrm{CH}'_i, \mathrm{RSP}'_i)\\
&\quad \textit{output } b
\end{aligned}
$$

with the restriction that $\mathrm{CH}'_i \neq \mathrm{CH}_i$*. We say that* Σ *is unforgeable if for all* PPT $\mathcal{A}$*, for all* $n = \mathsf{poly}(k)$*,*

$$\Pr\left[\,\mathbf{Unf}_{\Sigma,\mathcal{A}}(k) = 1\,\right] \le \mathsf{negl}(k),$$

where the probability is over the coins of Gen_R*,* Comm*,* Chall *and* $\mathcal{A}$*.*

Theorem 4.2.9. *If Σ is partial SHVZK and satisfies the strong special soundness property, then it is unforgeable.*

Proof. We show that if Σ satisfies strong special soundness and if there exists an $n = \mathsf{poly}(k)$ and a PPT adversary $\mathcal{A}$ that succeeds in an $\mathbf{Unf}_{\Sigma,\mathcal{A},n}(k)$ experiment with non-negligible probability, then there exists a PPT honest verifier $\mathcal{V}$ that can recover the witness w from n accepting partial transcript of Σ, also with non-negligible probability. The theorem then follows from the observation that partial SHVZK is preserved under sequential composition.

Let n be as above, and let $\{(\text{CMT}_i, \text{CH}_i, \text{RSP}_i)\}_i$, where $1 \leq i \leq n$, be a set of n accepting transcripts of Σ executed on common input x and witness w, where $(x, w) \in R$. Let $\mathcal{K}$ be the knowledge extractor whose existence is guaranteed by the strong special soundness of Σ. Consider the verifier $\mathcal{V}$ that, given the common input x and the partial transcripts $\{(\text{CH}_i, \text{RSP}_i)\}_i$, works as follows. It begins by running $\mathcal{A}(x, \{\text{CH}_i, \text{RSP}_i)\}_i)$, receiving a value $1 \leq i \leq n$ and a challenge/response pair $(\text{CH}', \text{RSP}')$. It then runs $\mathcal{K}(x, \text{CH}_i, \text{CH}', \text{RSP}_i, \text{RSP}')$, receiving a witness w' which it outputs.

Since $\mathcal{A}$'s simulated view is identical to its view in an $\mathbf{Unf}_{\Sigma,\mathcal{A},n}(k)$ experiment, by our assumption, it will output with non-negligible probability a tuple $(i, \text{CH}', \text{RSP}')$ such that $\text{CH}' \neq \text{CH}_i$ and such that $(\text{CMT}_i, \text{CH}', \text{RSP}')$ is accepting. It follows then that $\mathcal{K}$ will output a witness w' such that $(x, w') \in R$ also with non-negligible probability.

■

4.2.4 Homomorphically Verifiable Sigma-Protocols

In addition to unforgeability, we will also require the underlying sigma-protocol to be *homomorphically verifiable*. Roughly, this means that one should be able to verify the validity of multiple interactions by verifying their "sum". We provide a more formal description in Definition 4.2.10 below.

Definition 4.2.10 (Homomorphic verifier)**.** *Let $\mathbb{F}$ be a field and Σ be a sigma-protocol for a binary* **NP***-relation R with challenge and response space $\mathbb{F}$. A homomorphic verifier for Σ is a deterministic polynomial-time algorithm $\mathsf{Ver_H}$ such that,*

$b \leftarrow \mathsf{Ver_H}(x, \mathbf{a}, \mathrm{CMT}_1, \ldots, \mathrm{CMT}_n, \boldsymbol{c}, \boldsymbol{r})$*: takes as input a value $x \in \{0,1\}^*$, a vector $\mathbf{a} \in \mathbb{F}^n$, where $n = \mathsf{poly}(|x|)$, n commitments, and two elements $\boldsymbol{c}, \boldsymbol{r} \in \mathbb{F}$. It outputs a bit b, where '1' indicates acceptance and '0' indicates rejection.*

We will require that a homomorphic verifier be both complete and sound. Intuitively, we will say that $\mathsf{Ver_H}$ is complete if it always accepts when the elements $\mathbf{c}$ and $\mathbf{r}$ are generated using challenges and responses from n accepting transcripts.

Definition 4.2.11 (Completeness)**.** *Let $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Dec})$ be a sigma-protocol, $\mathsf{Ver_H}$ be a homomorphic verifier for Σ, $k \in \mathbb{N}$ be the security parameter, and $n \in \mathbb{N}$. We define $\mathbf{Comp}_{\mathsf{Ver_H},n}(k)$ as the following probabilistic experiment:*

$\mathbf{Comp}_{\mathsf{Ver}_\mathsf{H},n}(k)$
$(x, w) \leftarrow \mathsf{Gen}_R(1^k)$
for $1 \leq i \leq n$
$(\mathrm{CMT}_i, r) \leftarrow \mathsf{Comm}(x)$
$\mathrm{CH}_i \leftarrow \mathsf{Chall}(1^k)$
$\mathrm{RSP}_i \leftarrow \mathsf{Resp}(x, w, \mathrm{CMT}_i, \mathrm{CH}_i)$
$\mathbf{a} \xleftarrow{\$} \mathbb{F}^n$
$\boldsymbol{c} \leftarrow \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathrm{CH}_i$
$\boldsymbol{r} \leftarrow \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathrm{RSP}_i$
$b \leftarrow \mathsf{Ver}_\mathsf{H}(x, \mathbf{a}, \mathrm{CMT}_1, \ldots, \mathrm{CMT}_n, \boldsymbol{c}, \boldsymbol{r})$
output b

We say that Ver_H *complete if for all* $n = \mathsf{poly}(|x|)$,

$$\Pr\left[\, \mathbf{Comp}_{\mathsf{Ver}_\mathsf{H},n}(k) = 1 \,\right] = 1,$$

where the probability is over the coins of Gen_R, Comm *and* Chall, *and the choice of* $\mathbf{a}$.

Soundness, on the other hand, will guarantee that no efficient adversary is able to generate values $\mathbf{c}$ and $\mathbf{r}$ such that Ver_H will accept, unless $\mathbf{c}$ and $\mathbf{r}$ are generated using the challenges and responses of n accepting transcripts.

Definition 4.2.12 (Soundness)**.** *Let* $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Dec})$ *be a sigma-protocol,* Ver_H *be a homomorphic verifier for* Σ, $k \in \mathbb{N}$ *be the security parameter,* $\mathcal{A}$ *be an adversary, and* $n \in \mathbb{N}$. *We define* $\mathbf{Sound}_{\mathsf{Ver}_\mathsf{H},\mathcal{A},n}(k)$ *as the following probabilistic experiment:*

$\mathbf{Sound}_{\mathsf{Ver_H},\mathcal{A},n}(k)$

$(x, w) \leftarrow \mathsf{Gen}_R(1^k)$

for $1 \leq i \leq n$

$(\mathrm{CMT}_i, r) \leftarrow \mathsf{Comm}(x)$

$\mathrm{CH}_i \leftarrow \mathsf{Chall}(1^k)$

$\mathrm{RSP}_i \leftarrow \mathsf{Resp}(x, w, \mathrm{CMT}_i, \mathrm{CH}_i)$

$\mathbf{a} \xleftarrow{\$} \mathbb{F}^n$

$(\boldsymbol{c}, \boldsymbol{r}) \leftarrow \mathcal{A}\,(x, \mathbf{a}, \{(\mathrm{CH}_i, \mathrm{RSP}_i)\}_i)$

$b \leftarrow \mathsf{Ver_H}(x, \mathbf{a}, \mathrm{CMT}_1, \ldots, \mathrm{CMT}_n, \boldsymbol{c}, \boldsymbol{r})$

output b

with the restriction that either $\boldsymbol{c} \neq \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathrm{CH}_i$ *or that* $\boldsymbol{r} \neq \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathrm{RSP}_i$. *We say that* $\mathsf{Ver_H}$ *is sound if for all* $n = \mathsf{poly}(|x|)$, *for all* PPT $\mathcal{A}$,

$$\Pr\left[\,\mathbf{Sound}_{\mathsf{Ver_H},\mathcal{A},n}(k) = 1\,\right] \leq \mathsf{negl}(k),$$

where the probability is over the coins of Gen_R, Comm, Chall *and* $\mathcal{A}$, *and the choice of* $\mathbf{a}$.

We say that Σ is homomorphically verifiable if it has a homomorphic verifier that is both complete and sound.

4.3 Compiling Sigma-Protocols into PDP Systems

In this section we show how to compile any unforgeable and homomorphically verifiable sigma-protocol into a PDP system with constant size proofs. We provide a high-level description of our compiler which is described in detail in Figure 4.2.

Let Σ be a sigma-protocol as above, let R be the hard on average **NP**-relation on which Σ is defined, and let $\mathbb{F}_q$, for q prime, be its challenge and response space. We run the Gen_R algorithm to generate a pair (x, w) and set the public-key PK to be x and the secret-key SK to be the witness w. Given a file F we first parse it into n blocks $(f_1, \ldots, f_n)$. For every block f_i, we then run the Comm algorithm in order to generate a randomness/commitment pair (r_i, CMT_i). Next, for each file block we generate a *tag* in the following manner. We run the Resp algorithm using CMT_i and f_i as the commitment and challenge, respectively. This results in a response RSP_i, which is the tag for the block f_i. The encoded file is composed of all the file blocks together with their tags, and the state is the set of commitments.

As a challenge, the client sends the key to a pseudo-random function with range $\mathbb{F}_q$. This is only to save bandwidth, however, and is equivalent to sending a vector of n randomly chosen values in $\mathbb{F}_q$. To prove possession of a file, the honest server returns as proof the dot products of the challenge vector with the file blocks, and with the tags. Finally, the client verifies the proof using the homomorphic verifier.

Theorem 4.3.1. *Let Σ be a sigma-protocol for a binary* **NP***-relation that is hard on average. If Φ is a pseudo-random function, if Σ is unforgeable and homomorphically verifiable, then Π as described in Figure 4.2 is a sound privately verifiable PDP system.*

Proof Sketch: Consider the PDP system $\widetilde{\Pi} = (\mathsf{Gen}, \mathsf{Encode}, \widetilde{\mathsf{Chall}}, \widetilde{\mathsf{Proof}}, \widetilde{\mathsf{Ver}})$ in which $\widetilde{\mathsf{Chall}}$ samples a vector $\mathbf{a} \xleftarrow{\$} \mathbb{F}_q^n$ as the challenge, and $\widetilde{\mathsf{Proof}}$ and $\widetilde{\mathsf{Ver}}$ are defined in the

natural way according to the scheme described above. We analyze the security of $\widetilde{\Pi}$ and note that the security of Π can be derived by a standard argument.

We describe a probabilistic expected polynomial-time knowledge extractor $\mathcal{K}$ that satisfies Definition 4.2.3. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a PPT adversary, $r \in \{0,1\}^*$, and $\mathrm{E} = (\mathrm{PK}, F, \widetilde{F}, \mathrm{ST}, \mathrm{ST}_{\mathcal{A}})$ be an environment output by $\mathbf{Setup}_{\Pi,\mathcal{A}_1}(k)$, such that

$$\delta(k) \stackrel{\text{def}}{=} \Pr\left[\mathsf{Ver}(\mathrm{PK}, \mathrm{ST}, \mathbf{a}, \pi) = 1 : \mathbf{a} \xleftarrow{\$} \mathbb{F}_q^n; \pi \leftarrow \mathcal{A}_2(\mathrm{ST}_{\mathcal{A}}, \widetilde{F}, \mathbf{a}; r)\right] \geq 1/\mathsf{poly}(k).$$

Consider the extractor $\mathcal{K}$ that works as follows during an $\mathbf{Ext}_{\widetilde{\Pi},\mathcal{A}_2,\mathcal{K}}[r](k)$ experiment. Given inputs $(\mathrm{PK}, \mathrm{ST})$ and oracle access to $\mathcal{A}_2(\mathrm{ST}_{\mathcal{A}}, \mathrm{ST}, \cdot; r)$, $\mathcal{K}$ does the following:

1. it queries $\mathcal{A}_2$ on a vector $\mathbf{a}_1$ selected uniformly at random from $\mathbb{F}_q^n$, receiving a proof $\pi_1' = (\mathbf{f}_1', \mathbf{t}_1')$. It then computes $\mathsf{Ver}_{\mathsf{H}}(\mathrm{PK}, \mathbf{a}_1, \mathrm{CMT}_1, \ldots, \mathrm{CMT}_n, \mathbf{f}_1', \mathbf{t}_1')$ which outputs a bit b. If $b = 1$, we say that the proof π_1 is *valid*. If π_1 is valid it continues, otherwise it halts.

2. for $2 \leq i \leq n$, it queries $\mathcal{A}_2$ on a vector selected uniformly at random from the set $\mathsf{indep}(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1})$ until it receives a valid proof $\pi_i' = (\mathbf{f}_i', \mathbf{t}_i')$, where $\mathsf{indep}(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1})$ is the set of vectors in $\mathbb{F}_q^n$ that are linearly independent from the vectors $(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1})$. We denote by $\mathbf{a}_i$ be the vector that leads to the valid proof π_i'.

3. it sets up the following sets of linear congruences:

$$\left\{\sum_{j=1}^{n} \mathbf{a}_i[j] \cdot f_j^* \equiv \mathbf{f}_i' \bmod p\right\}_i \text{ and } \left\{\sum_{j=1}^{n} \mathbf{a}_i[j] \cdot t_j^* \equiv \mathbf{t}_i' \bmod p\right\}_i,$$

and solves for $(f_1^*, \ldots, f_n^*)$ and $(t_1^*, \ldots, t_n^*)$.

4. it outputs $F^* = \langle f_1^*, \ldots, f_n^* \rangle$

Claim. $\mathcal{K}$ runs in expected polynomial-time.

Let N be the number of vectors in $\mathbb{F}_q^n$ on which $\mathcal{A}_2(\text{ST}_{\mathcal{A}}, \text{ST}, \widetilde{F}, \cdot; r)$ outputs a valid proof. We have two cases: (1) π_1 is invalid and $\mathcal{K}$ halts; and (2) π_1 is valid and $\mathcal{K}$ executes steps 2 and 3. Since step 3 can be done in deterministic polynomial-time we restrict our attention to step 2. For each $2 \leq i \leq n$, notice that

$$|\mathsf{indep}(\mathbf{a}_1, \ldots, \mathbf{a}_{i-1})| = \frac{N - q^{i-1}}{q^n - q^{i-1}}.$$

Therefore, for $2 \leq i \leq n$, the expected number of iterations until $\mathcal{A}_2$ outputs π_i is,

$$\frac{q^n - q^{i-1}}{N - q^{i-1}}.$$

It follows then that the expected number of iterations until $\mathcal{A}_2$ outputs $(\pi_2, \ldots, \pi_n)$ is

$$\sum_{i=2}^{n} \left(\frac{q^n - q^{i-1}}{N - q^{i-1}} \right) \leq n \left(\frac{q^n - q^{n-1}}{N - q^{n-1}} \right),$$

which is polynomial in k since q and n are constant, and

$$\begin{aligned} N &= q^n \cdot \delta(k) \\ &\geq 1/\mathsf{poly}(k). \end{aligned}$$

□

In the following, we only provide a high level intuition as to the correctness of our extractor. Clearly, the probability that $\mathcal{K}$ succeeds is at least the probability that all the recovered blocks $(f_1^*, \ldots, f_n^*)$ are equal to original blocks $(f_1, \ldots, f_n)$. If $f_i^* = f_i$, where $1 \leq i \leq n$, we say that f_i^* is *correct*.

Since the only probabilistic step $\mathcal{K}$ takes is Step 1, it follows that its success probability is the probability that all the recovered blocks are correct conditioned on π_1 being valid. In addition, conditioned on the latter occurring, all the proofs $(\pi_1, \ldots, \pi_n)$ are valid with certainty. But if the proofs are valid, then the soundness of the homomorphic verifier implies that the proofs were generated by summing accepting transcripts of Σ, which, by the unforgeability of Σ, implies that the recovered blocks will be correct.

■

4.4 Concrete Instantiations Based on the Schnorr Protocol

We now focus on concrete instantiations of PDP systems based on our compiler. In particular on constructions that are based on the Schnorr protocol for proving knowledge of discrete logarithms [58]. The Schnorr protocol is known to be SHVZK and specially sound [23]. We now show that it is also homomorphically verifiable.

Theorem 4.4.1. *The Schnorr protocol is homomorphically verifiable.*

Proof. Let $(\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Ver})$ be the Schnorr protocol as defined in Figure 4.3. We describe a homomorphic verifier for Schnorr, $\mathsf{Ver}_{\mathsf{H}}$, that satisfies Definition 4.2.10. Note that in the Schnorr protocol the challenge and response space is the field $\mathbb{Z}_q$.

$\mathsf{Ver}_{\mathsf{H}}(x, \mathbf{a}, R_1, \ldots, R_n, \mathbf{c}, \mathbf{t})$: if $g^{\mathbf{t}} = \left(R_1^{\mathbf{a}[1]} \times \cdots \times R_n^{\mathbf{a}[n]}\right) \cdot x^{\mathbf{c}} \mod p$ then output 1 otherwise 0.

In the following claims we show that $\mathsf{Ver}_{\mathsf{H}}$ is both complete and sound.

Claim. $\mathsf{Ver}_{\mathsf{H}}$ is complete.

Let $\mathbf{a} \in \mathbb{Z}_q^n$, and let $\{(R_i, c_i, t_i)\}_{1 \leq i \leq n}$ be a set of n accepting transcripts. Suppose

$$\mathbf{c} = \sum_{i=1}^{n} \mathbf{a}[i] \cdot c_i \mod p \text{ and } \mathbf{t} = \sum_{i=1}^{n} \mathbf{a}[i] \cdot t_i \mod p.$$

It follows then that,

$$\begin{aligned} g^{\mathbf{t}} &= \Pi_{i=1}^{n} \left(g^{t_i}\right)^{\mathbf{a}[i]} \\ &= \Pi_{i=1}^{n} \left(R_i \cdot x^{c_i}\right)^{\mathbf{a}[i]} \\ &= \left(R_1^{\mathbf{a}[1]} \times \cdots \times R_n^{\mathbf{a}[n]}\right) \cdot x^{\mathbf{a}[1] \cdot c_1 + \cdots + \mathbf{a}[n] \cdot c_n} \\ &= \left(R_1^{\mathbf{a}[1]} \times \cdots \times R_n^{\mathbf{a}[n]}\right) \cdot x^{\mathbf{c}}. \end{aligned}$$

□

Claim. $\mathsf{Ver}_{\mathsf{H}}$ is sound.

We show that if there exists an $n = \mathsf{poly}(k)$ and a PPT adversary $\mathcal{A}$ that succeeds in a $\mathbf{Sound}_{\mathsf{Ver}_{\mathsf{H}}, \mathcal{A}, n}(k)$ experiment with non-negligible probability, then there exists a PPT

adversary $\mathcal{B}$ that succeeds in a $\mathbf{Unf}_{\Sigma,\mathcal{B}}(k)$ experiment also with non-negligible probability. Since the Schnorr protocol is SHVZK and satisfies strong special soundness, by Theorem 4.2.9 it is unforgeable and the claim will follow.

Consider the adversary $\mathcal{B}$ that, given x and a set of n partial transcripts $\{(c_i, t_i)\}_{1 \leq i \leq n}$, works as follows:

1. it samples $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n$ and computes $(\mathbf{c}, \mathbf{t}) \leftarrow \mathcal{A}(x, \mathbf{a}, \{(c_i, t_i)\}_i)$
2. it computes $\mathbf{t}' = \sum_{i=1}^{n} \mathbf{a}[i] \cdot t_i \mod q$ and $w' = (\mathbf{t} - \mathbf{t}')/(\mathbf{c} - \sum_{i=1}^{n} \mathbf{a}[i] \cdot c_i) \mod q$
3. it chooses an arbitrary $1 \leq i \leq n$, and computes $r_i' = t_i - c_i \cdot w' \mod q$. It then chooses an arbitrary value c' different than all $(c_1, \ldots, c_n)$, computes $t' = r_i' + c' \cdot w' \mod q$ and outputs the pair (c', t')

It remains to analyze $\mathcal{B}$'s success probability. Let valid be the event that $\mathsf{Ver}_{\mathsf{H}}(x, \mathbf{a}, R_1, \ldots, R_n, \mathbf{c}, \mathbf{t})$ outputs 1. Conditioned on valid occurring, it follows that

$$\mathbf{t} = w \cdot \mathbf{c} + \sum_{i=1}^{n} r_i \cdot \mathbf{a}[i] \mod q,$$

and that

$$\begin{aligned} \mathbf{t}' - \mathbf{t} &= w \cdot \mathbf{c} + \sum_{i=1}^{n} r_i \cdot \mathbf{a}[i] - w \cdot \sum_{i=1}^{n} \mathbf{a}[i] \cdot c_i - \sum_{i=1}^{n} \mathbf{a}[i] \cdot r_i \mod q \\ &= w \cdot \left(\mathbf{c} - \sum_{i=1}^{n} \mathbf{a}[i] \cdot c_i \right) \mod q. \end{aligned}$$

However, recall that $\mathbf{c} \neq \sum_{i=1}^{n} \mathbf{a}_i \cdot c_i \mod q$ so w' in Step 2 will be equal to the witness w. It follows then that (c', t') will be such that $\Sigma.\mathsf{Ver}(x, R_i, c', t')$ outputs 1.

Since valid occurs with non-negligible probability, it follows that $\mathcal{B}$ will succeed in its $\mathbf{Unf}_{\Sigma,\mathcal{B}}(k)$ experiment also with non-negligible probability.

□

■

Since the Schnorr protocol is SHVZK, satisfies special soundness and has a homomorphic verifier, by Theorem 4.3.1, our compiler will yield a privately verifiable PDP system with short proofs. Unfortunately, the resulting system has the limitation that the size of the state St is linear in n, the number of blocks.

4.4.1 A Compact Privately Verifiable PDP System

In this section we describe a protocol that, when compiled, yields a privately verifiable *compact* PDP system. The protocol is a simple modification of the Schnorr protocol so it inherits some of its properties including its homomorphic verifier. The protocol, Γ_S, is described in detail in Figure 4.4.

We observe that Γ_S is partial SHVZK, satisfies strong special soundness and is homomorphically verifiable. This follows directly from the facts that (1) the Schnorr protocol satisfies these properties; and that (2) the challenge and response algorithms of both protocols are identical.

By Theorem 4.3.1, it then follows that the PDP system that results from compiling Γ_S will be privately verifiable and will have constant-size proofs. We observe, however,

that Γ_S can be made compact as follows. Notice that it's Comm algorithm outputs a single fixed commitment K. Therefore, the state information will be composed of n copies of K which can obviously be compressed to a single K. For completeness, we include the description of Π_S in figure 4.5.

4.5 Conclusions

In this chapter we showed how to compile sigma-protocols into PDP systems, establishing an explicit connection between the two. In particular, we showed that if the protocol possesses certain homomorphic properties, then it can be compiled into a PDP system with $O(1)$-size proofs. We also identified sufficient security properties for a sigma-protocol to yield a privately verifiable PDP system under our compiler. Finally, we describe a simple protocol based on the Schnorr protocol for proving knowledge of discrete logarithms [58] which, when compiled, generates a privately verifiable *compact* PDP system, i.e., it has communication and storage complexity at the verifier that is $O(1)$. In addition to being efficient, our construction allows for an unlimited number of proofs and supports public data.

An interesting open problem resulting from this work is to identify a sufficient set of properties for sigma-protocol to yield a publicly verifiable compact PDP system under our compiler. Also, since partial SHVZK is sufficient but does not seem necessary to construct PDPs, a precise formulation of the minimal security requirements needed for our compiler to achieve private and public-verifiability would be useful. In

addition, this might lead to more efficient PDP constructions. Another interesting direction is to establish an explicit connection between PDP systems and identification schemes.

Let R be an **NP**-relation that is hard on average. Let $\Sigma = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Ver})$ be a sigma-protocol for R that is homomorphically verifiable over a finite field $\mathbb{F}_q$. Let $\Phi : \{0,1\}^k \times \mathbb{Z}_n \rightarrow \mathbb{F}_q$ be a pseudo-random function. $\Pi = (\mathsf{Gen}, \mathsf{Encode}, \mathsf{Chall}, \mathsf{Proof}, \mathsf{Ver})$ is the PDP system defined as follows.

$\mathsf{Gen}(1^k)$: compute $(x, w) \leftarrow \mathsf{Gen}_R(1^k)$, and output $\text{PK} = x$ and $\text{SK} = w$.

$\mathsf{Encode}(\text{PK}, \text{SK}, F)$:

1. parse the file F into n blocks $F = \langle f_1, \ldots, f_n \rangle$ such that each f_i is in the challenge space.
2. for $1 \leq i \leq n$
   a. $(\text{CMT}_i, r_i) \leftarrow \Sigma.\mathsf{Comm}(x)$
   b. $t_i \leftarrow \Sigma.\mathsf{Resp}(x, w, r_i, f_i)$
3. output $\widetilde{F} = \langle f_1, \ldots, f_n, t_1, \ldots, t_n \rangle$ and $\text{ST} = \langle \text{CMT}_1, \ldots, \text{CMT}_n \rangle$.

$\mathsf{Chall}(\text{PK})$: sample and output $\text{K} \xleftarrow{\$} \{0,1\}^k$

$\mathsf{Proof}(\text{PK}, \widetilde{F}, c)$:

1. for $1 \leq i \leq n$, compute $a_i \leftarrow \Phi_{\text{K}}(i)$
2. compute $\mathbf{f} = a_1 \cdot f_1 + \cdots + a_n \cdot f_n$
3. compute $\mathbf{t} = a_1 \cdot t_1 + \cdots + a_n \cdot t_n$
4. output $\langle \mathbf{f}, \mathbf{t} \rangle$

$\mathsf{Ver}(\text{PK}, \text{ST}, c, \pi)$:

1. for $1 \leq i \leq n$, compute $a_i \leftarrow \Phi_{\text{K}}(i)$
2. compute and output $b \leftarrow \Sigma.\mathsf{Ver}_{\mathsf{H}}(\text{PK}, a_1, \ldots, a_n, \text{CMT}_1, \ldots, \text{CMT}_n, \mathbf{f}, \mathbf{t})$

Figure 4.2: Compiling a sigma-protocol into a PDP system

Let $p, q \in \mathsf{primes}(k)$ such that $p = 2q + 1$. Choose $g \in \mathbb{Z}_p^*$ such that g has order q. Sample $w \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $x = g^w \mod p$. Make (p, q, g) public.

$\mathsf{Comm}(x)$: sample $r \xleftarrow{\$} \mathbb{Z}_q$. Compute $R \leftarrow g^r \mod p$; and output (R, r).

$\mathsf{Chall}(1^k)$: sample and output $c \xleftarrow{\$} \mathbb{Z}_q$

$\mathsf{Resp}(x, w, R, c)$: compute and output $t \leftarrow r + c \cdot w \mod q$

$\mathsf{Ver}(x, R, c, t)$: if $g^t \equiv R \cdot x^c \mod p$ output 1, otherwise output 0

Figure 4.3: The Schnorr protocol

Let $p, q \in \mathsf{primes}(k)$ such that $p = 2q + 1$. Choose $g \in \mathbb{Z}_p^*$ such that g has order q. Sample $w \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $x = g^w \mod p$. Make (p, q, g) public. Let $\Phi : \{0,1\}^k \times \{0,1\}^k \to \mathbb{Z}_q$ be a pseudo-random function and let $\mathrm{K} \xleftarrow{\$} \{0,1\}^k$. $\Gamma_S = (\mathsf{Comm}, \mathsf{Chall}, \mathsf{Resp}, \mathsf{Ver})$ is the protocol defined as follows.

$\mathsf{Comm}(x)$: let i be a counter initialized to 0 and K be "hardwired". Compute $r \leftarrow \Phi_{\mathrm{K}}(i)$ and output (K, r).

$\mathsf{Chall}(1^k)$: sample and output $c \xleftarrow{\$} \mathbb{Z}_q$

$\mathsf{Resp}(x, w, \mathrm{K}, c)$: let i be a counter initialized to 0 and K be "hardwired". Compute and output $t \leftarrow \Phi_{\mathrm{K}}(i) + c \cdot w \mod q$

$\mathsf{Ver}(x, \mathrm{K}, c, t)$: let i be a counter initialized to 0. Compute $r \leftarrow \Phi_{\mathrm{K}}(i)$ and $R \leftarrow g^r \mod p$. If $g^t \equiv R \cdot x^c \mod p$ output 1, otherwise output 0

Figure 4.4: A Schnorr-based protocol

$\mathsf{Gen}(1^k)$: Choose $p \xleftarrow{\$} \mathsf{primes}(k)$ and q such that $p = 2q + 1$, $g \in \mathbb{Z}_p^*$ such that g has order q, and $w \xleftarrow{\$} \mathbb{Z}_q$. Let $\Phi : \{0,1\}^k \times \mathbb{Z}_n \to \mathbb{Z}_q$ be a pseudo-random function. Compute $x = g^w \mod p$, and set $\text{SK} = w$ and $\text{PK} = (p, q, g, x)$.

$\mathsf{Encode}(\text{PK}, \text{SK}, F)$:

1. parse the file F into n blocks $F = \langle f_1, \ldots, f_n \rangle$ such that $f_1 + \cdots + f_n < q$.
2. sample $\text{K}_1 \xleftarrow{\$} \{0,1\}^k$
3. for $1 \le i \le n$
   a. compute $r_i \leftarrow \Phi_{\text{K}_1}(i)$, and let $R_i = g^{r_i} \mod p$
   b. compute $t_i = r_i + f_i \cdot w \mod q$
4. output $\widetilde{F} = \langle f_1, \ldots, f_n, t_1, \ldots, t_n \rangle$ and $\text{ST} = \text{K}_1$.

$\mathsf{Chall}(\text{PK})$: sample and output $\text{K}_2 \xleftarrow{\$} \{0,1\}^k$

$\mathsf{Proof}(\text{PK}, \widetilde{F}, c)$:

1. for $1 \le i \le n$, compute $a_i \leftarrow \Phi_{\text{K}_2}(i)$
2. compute $\mathbf{f} = a_1 \cdot f_1 + \cdots + a_n \cdot f_n \mod q$
3. compute $\mathbf{t} = a_1 \cdot t_1 + \cdots + a_n \cdot t_n \mod q$
4. output $\pi = \langle \mathbf{f}, \mathbf{t} \rangle$

$\mathsf{Ver}(\text{PK}, \text{ST}, c, \pi)$:

1. for $1 \le i \le n$,
   a. compute $r_i \leftarrow \Phi_{\text{K}_1}(i)$
   b. compute $a_i \leftarrow \Phi_{\text{K}_2}(i)$
2. output 1 if and only if $g^{\mathbf{t}} = (R_1^{a_1} \times \cdots \times R_n^{a_n}) \cdot x^{\mathbf{f}} \mod p$.

Figure 4.5: A privately verifiable compact PDP system

Chapter 5

Storing Private Data on an Untrusted Server

5.1 Introduction

As discussed in the Introduction, the motivations for storage outsourcing are numerous. By outsourcing, organizations and individuals can concentrate on their core tasks rather than incurring the substantial hardware, software and personnel costs involved in maintaining data "in house". However, when the data to be outsourced is sensitive, such as hospital records or even emails, outsourcing raises many privacy concerns.

To address this, private-key encryption can be used to encrypt the data before storage. This approach, however, can be very inefficient when applied to large amounts

of data. Indeed, consider the case of a hospital which stores all its patient records encrypted using a private-key encryption scheme. Note that in such a setting, even the existence of a record for a given patient should be kept private (i.e., the untrusted server should not even learn that a given person is a patient at the hospital). In this case, in order to recover a single record from the server, a staff member will have to download the entire encrypted data collection, decrypt it and search for the appropriate record. Clearly, this solution is inefficient both in terms of communication complexity and in terms of the computation performed by the server and the client.

Motivated by this problem, several techniques have been proposed for provisioning private-key encryption with search capabilities [62, 35, 13, 18, 22]. The resulting primitive is typically called *searchable encryption*.

5.1.1 Previous work

Before providing a comparison to existing work, we put our work in context by providing a classification of the various models for privacy-preserving searches, including searching on *public-key* encrypted data; single-database *private information retrieval* (PIR); and searching on *private-key* encrypted data (which is the subject of this work). Common to all three models is a server (sometimes called the "database") that stores data, and a user that wishes to access, search, or modify the data while revealing as little as possible to the server. There are, however, important differences between these three settings.

Searching on public-key encrypted data. In the setting of searching on public-key encrypted data, users who encrypt the data can be different from the owner of the decryption key. In a typical application, a user publishes a public key while multiple senders send e-mails to the mail server [18, 3]. Anyone with access to the public key can encrypt emails, but only the owner of the private key can generate "trapdoors" to test for the occurrence of a keyword. Although the original work on public-key encryption with keyword search (PEKS) by Boneh *et al.* [18] reveals the user's access pattern, Boneh, Kushilevitz, Ostrovsky and Skeith [19] have shown how to build a public-key encryption scheme that hides even the access pattern. This construction, however, has an overhead in search time that is proportional to the square root of the database size, which is far less efficient then the best private-key solutions.

Bellare, Boldyreva and O'Neill introduced the notion of asymmetric efficiently searchable encryption (ESE) and proposed three constructions in the random oracle model [7]. Similarly to PEKS, asymmetric ESE schemes allow anyone with access to a user's public-key to encrypt messages. On the other hand, unlike PEKS it allows *anyone* in possession of the public-key to generate trapdoors in order to search. Like our own constructions, asymmetric ESE schemes achieve optimal search time. We note, however, that in order to achieve this level of efficiency asymmetric ESE is inherently deterministic, and therefore provides security guarantees that are significantly weaker than the ones considered in our work.

Searching on public data. In single-database private information retrieval (PIR), introduced by Kushilevitz and Ostrovsky [49], they show how a user can retrieve data from a server containing *unencrypted* data without revealing the access pattern and with total communication less then the data size. This was extended to keyword searching, including searching on streaming data by Ostrovsky and Skeith [54]. We note, however, that since the data in PIR is always unencrypted, any scheme that tries to hide the access pattern must touch all data items. Otherwise, the server learns information: namely, that the untouched item was not of interest to the user. Thus, PIR schemes require work which is linear in the database size.

Searching on private-key encrypted data. In the setting of searching on private-key-encrypted data the user himself encrypts the data so he can organize it in an arbitrary way before encryption and include additional data structures to allow for efficient access. The data and the additional data structures can then be encrypted and stored on the server so that only someone with the private key can access it. In this setting, the initial work for the user (i.e., for preprocessing the data) is at least as large as the data, but subsequent work (i.e., for accessing the data) is very small relative to the size of the data for both the user and the server.

Searchable symmetric encryption can be achieved securely in its full generality using the work of Ostrovsky and Goldreich on oblivious RAMs [53, 39]. While oblivious RAMs hide all information (including the access pattern) from a remote and potentially malicious server, this comes at the cost of a logarithmic (in the of number

of documents) amount of rounds of interaction for each read and write. The authors also give a 2-round solution, but it incurs a considerably larger storage overhead. Therefore, the recent work on searchable encryption can be viewed as achieving more efficient solutions (typically in one or two rounds) by weakening the privacy guarantees.

In [62], Song, Wagner and Perrig showed that a solution for searchable encryption was possible for a weaker security model. Specifically, they achieve SSE by crafting, for each keyword, a special two-layered encryption construct. Given a trapdoor, the server can strip the outer layer and assert whether the inner layer is of the correct form. This construction, however, has some limitations: while it is proven to be a secure encryption scheme, it is not proven to be a secure *searchable* encryption scheme; the distribution of the underlying plaintexts is vulnerable to statistical attacks; and searching is linear in the length of the document collection.

Other than the current work, two definitions of security have been considered for SSE: indistinguishability against chosen-keyword attacks (CKA2-security), introduced by Goh [35][1], and a simulation-based definition introduced by Chang and Mitzenmacher [22]. We note that, unlike the latter and our own definitions, CKA2-security applies to indexes that are built for individual documents as opposed to indexes built from entire document collections.

Intuitively, the security guarantee that CKA2-security achieves can be described

[1] Goh also defines a weaker notion, CKA-security, that allows an index to leak the number of keywords in the document.

as follows: given access to a set of indexes, the adversary (i.e., the server) is not able to learn any partial information about the underlying document that he cannot learn from using a trapdoor that was given to him by the client, and this holds even against adversaries that can "trick" the client into generating indexes and trapdoors for documents and keywords of its choice (i.e., chosen-keyword attacks). We remark that Goh's work addresses a larger problem than searchable encryption, namely that of secure indexes, which are secure data structures that have many uses, only one of which is searchable encryption. And as Goh remarks (*cf.* Note 1, p. 5 of [35]), CKA2-security does not explicitly require that trapdoors be secure since this is not a requirement for all applications of secure indexes.

Regarding existing simulation-based definitions, in [22] Chang and Mitzenmacher provide a definition of security for SSE that is intended to be stronger than CKA2-security in the sense that it requires a scheme to output secure trapdoors. Unfortunately, as we point out, this definition can be trivially satisfied by any SSE scheme, even one that is insecure. Moreover, this definition is inherently *non-adaptive*, in the sense that it only guarantees security against adversaries that perform keyword searches without seeing the outcome of previous searches.

Both works propose constructions that associate an "index" to each document in a collection. As a result, the server has to search each of these indexes, and the amount of work required for a query is linear in the number of documents in the collection.

Naturally, SSE can also be viewed as an instance of secure two/multi-party compu-

tation [65, 38, 14]. However, the weakening and refinement of the privacy requirement as well as efficiency considerations mandate a specialized treatment of the problem, both at the definitional and construction levels.

5.1.2 Summary of Contributions

We start by revisiting previous security definitions for SSE and pointing out several of their limitations. Additionally, we show that the current definitions only achieve what we call *non-adaptive* security, while the more natural usage of searchable encryption calls for *adaptive* security (a notion we make precise in Section 5.3). We propose new security definitions for both the non-adaptive and adaptive cases, and present efficient constructions for both based on any one-way function.

Our first construction is the most efficient non-adaptive SSE scheme to date in terms of computation on the server, and incurs a minimal (i.e., constant) cost for the user. Our second construction achieves adaptive security, which was not previously achieved by any constant-round solution.

We also extend the problem of SSE to the multi-user setting, where a client wishes to allow an authorized group of users to search through its document collection.

The contents of this Chapter appear in [24].

5.2 Preliminaries

Document collections. Let $\Delta = \{w_1, \ldots, w_d\}$ be a dictionary of d words, and 2^{Δ} be the set of all possible documents. If $k \in \mathbb{N}$ is the security parameter, we assume $d = \mathsf{poly}(k)$ and that all words $w \in \Delta$ are of length polynomial in k. Furthermore, let $\mathbf{D} \subseteq 2^{\Delta}$ be a collection of $n = \mathsf{poly}(k)$ documents $\mathbf{D} = (D_1, \ldots, D_n)$, each containing $\mathsf{poly}(k)$ words. Let $\mathsf{id}(D)$ be the identifier of document D, where the identifier can be any string that uniquely identifies a document, such as a memory location. We denote by $\mathbf{D}(w)$ the lexicographically ordered list consisting of the identifiers of all documents in $\mathbf{D}$ that contain the word w.

Model. The participants in a single-user searchable encryption scheme include a user that wishes to store an encrypted document collection $\mathbf{D}$ on an honest-but-curious server S, while preserving the ability to search through them. We note that while we choose, for ease of exposition, to limit searches to be over documents, any SSE scheme can be trivially extended to search over lists of arbitrary keywords associated with the documents.

The participants in a multi-user searchable encryption scheme include a trusted owner O, an honest-but-curious server S, and a set of users N. O owns $\mathbf{D}$ and wants to grant and revoke searching privileges to a subset of users in N. We let $\mathsf{G} \subseteq \mathsf{N}$ be the set of users allowed to search. We assume that non-revoked users behave honestly.

More formally, throughout this paper, authorized users are modeled as probabilis-

tic polynomial-time Turing machines, while adversaries are modeled as non-uniform Turing machines.

5.3 Defining Security for Searchable Symmetric Encryption

We begin by reviewing the formal definition of a SSE scheme.

Definition 5.3.1 (Searchable symmetric encryption scheme)**.** *A SSE scheme is a collection of four polynomial-time algorithms* $\Sigma = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trapdoor}, \mathsf{Search})$ *such that,*

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic key generation algorithm that is run by the user to setup the scheme. It takes as input a security parameter* k*, and outputs a secret key* K*.*

$\mathrm{I} \leftarrow \mathsf{Index}(\mathrm{K}, \mathbf{D})$*: is a probabilistic algorithm run by the user to generate indexes. It takes as input a secret key* K *and a document collection* $\mathbf{D}$*, and outputs an index* I*. We sometimes write this as* $\mathrm{I} \leftarrow \mathsf{Index}_{\mathrm{K}}(\mathbf{D})$*.*

$\mathrm{T}_w \leftarrow \mathsf{Trapdoor}(\mathrm{K}, w)$*: is run by the user to generate a trapdoor for a given keyword. It takes as input a secret key* K *and a keyword* w*, and outputs a trapdoor* T_w*. We sometimes write this as* $\mathrm{T}_w \leftarrow \mathsf{Trapdoor}_{\mathrm{K}}(w)$*.*

$X \leftarrow \mathsf{Search}(\mathrm{I}, \mathrm{T}_w)$*: is run by the server in order to search for the documents in* $\mathbf{D}$ *that contain a keyword* w*. It takes as input an index* I *for a collection* $\mathbf{D}$ *and a trapdoor* T_w *for keyword* w*, and outputs a set* X *of document identifiers.*

Let $\mathbf{D}$ be a document collection and let w be a keyword. An SSE scheme is complete if, given an index for $\mathbf{D}$ and a trapdoor for w (generated under the same key used to generate the index), the search algorithm returns the identifiers of the documents in $\mathbf{D}$ that contain w.

Definition 5.3.2 (Completeness)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trapdoor}, \mathsf{Search})$ *be a SSE scheme,* $k \in \mathbb{N}$ *be the security parameter,* $\mathbf{D}$ *be a document collection and* $w \in \mathbf{D}$ *be a keyword. We define* $\mathbf{Comp}_{\Sigma,\mathbf{D},w}(k)$ *as the following probabilistic experiment:*

$\mathbf{Comp}_{\Sigma,\mathbf{D},w}(k)$
$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$
$\mathrm{I} \leftarrow \mathsf{Index}_{\mathrm{K}}(\mathbf{D})$
$\mathrm{T}_w \leftarrow \mathsf{Trapdoor}_{\mathrm{K}}(w)$
$X \leftarrow \mathsf{Search}(\mathrm{I}, \mathrm{T}_w)$
if $X = \mathbf{D}(w)$*, output* 1
otherwise output 0

We say that Σ *is complete if for all dictionaries* Δ*, all document collections* $\mathbf{D} \subseteq 2^{\Delta}$*, all keywords* $w \in \Delta$*,*

$$\Pr\left[\,\mathbf{Comp}_{\Sigma,\mathbf{D},w}(k) = 1\,\right] = 1,$$

where the probability is over the coins of Gen *and* Index*.*

So far, establishing correct security definitions for searchable encryption has been elusive. Clearly, as we have discussed, one could use the general definitions from

oblivious RAMs , but subsequent work (including ours) examines if more efficient schemes can be achieved by revealing *some* information. The first difficulty seems to be in correctly capturing this intuition in a security definition. In the literature, this has typically been characterized as the requirement that "nothing be leaked beyond the access pattern" , i.e., the identifiers of the documents that contain a sequence of queried keywords. However, we are not aware of any previous work on SSE that satisfies this intuition. In fact, with the exception of oblivious RAMs, in addition to the access pattern all the constructions in the literature leak what we refer to as the *search pattern*. Informally, the search pattern is any information that can be derived from knowing whether two searches were performed for the same keyword or not. This is clearly the case for the schemes presented in [62, 35, 22] since their trapdoors are deterministic. Therefore, a more accurate characterization of the security notion achieved (or rather, sought) for SSE is that nothing should be leaked beyond the access and the search patterns of a sequence of searches.

The second issue seems to be in appropriately capturing the adversary's power. In fact, while Song *et al.* prove their construction secure, the definition implicitly used in their work is that of a classical encryption scheme, where the adversary is not allowed to perform searches. This was partly rectified by Goh [35], whose security definition for secure indexes gives the adversary access the to Index and Trapdoor oracles. While much work on searchable encryption uses CKA2-security as a security definition [42, 55, 5], we note that it was never intended as such. In fact, Goh notes

that CKA2-security does not explicitly require trapdoors to be secure, which we deem is an important requirement for any searchable encryption scheme.

In [22], Chang and Mitzenmacher propose a simulation-based definition that aims to guarantee privacy for indexes *and* trapdoors. Similarly to the classical definition of semantic security for encryption [40], they require that anything that can be computed from the index and the trapdoors for various queries, can be computed from the outcome of those queries (i.e., the identifiers of the documents that contain the keywords). However, while the intuition seems correct, in the case of searchable encryption one must also take care in describing *how* the search queries are generated. In particular, whether they can be made adaptively (i.e., after seeing the outcome of previous queries) or non-adaptively (i.e., without seeing the outcome of any queries). This distinction is important because it leads to security definitions that achieve different privacy guarantees. Indeed, while non-adaptive definitions only guarantee security to clients who generate all their queries at once, adaptive definitions guarantee privacy even to clients who generate queries as a function of previous search outcomes.

We now address the above issues. Before stating our definitions for SSE, we introduce three auxiliary notions which we will make use of. First, we note that an interaction between the client and the server will be determined by a document collection and a set of keywords that the client wishes to search for and that we wish to hide from the adversary. We call an instantiation of such an interaction a *history*.

Definition 5.3.3 (History)**.** *Let* Δ *be a dictionary and* $\mathbf{D} \subseteq 2^{\Delta}$ *be a document collection over* Δ*. A* q*-query* history *over* $\mathbf{D}$ *is a tuple* $\mathrm{H} = (\mathbf{D}, w_1, \ldots, w_q)$ *that includes the document collection* $\mathbf{D}$ *and* q *keywords* $(w_1, \ldots, w_q)$*.*

Definition 5.3.4 (Access Pattern)**.** *Let* Δ *be a dictionary and* $\mathbf{D} \subseteq 2^{\Delta}$ *be a document collection over* Δ*. The access pattern induced by a* q*-query history* $\mathrm{H} = (\mathbf{D}, w_1, \ldots, w_q)$*, is the tuple* $\alpha(\mathrm{H}) = (\mathbf{D}(w_1), \ldots, \mathbf{D}(w_q))$*.*

Definition 5.3.5 (Search Pattern)**.** *Let* Δ *be a dictionary and* $\mathbf{D} \subseteq 2^{\Delta}$ *be a document collection over* Δ*. The search pattern induced by a* q*-query history* $\mathrm{H} = (\mathbf{D}, w_1, \ldots, w_q)$*, is a symmetric binary matrix* $\sigma(\mathrm{H})$ *such that for* $1 \leq i, j \leq q$*, the element in the* i^{th} *row and* j^{th} *column is* 1 *if* $w_i = w_j$*, and* 0 *otherwise.*

The final notion is that of the *trace* of a history, which consists of exactly the information we are willing to leak about the history and nothing else. More precisely, this should include the identifiers of the documents that contain each keyword in the history (i.e., the outcome of each search), and information that describes which trapdoors in the view correspond to the same underlying keywords in the history (i.e., the pattern of the searches). According to our intuitive formulation of security, this should be no more than the access and search patterns. However, since in practice the encrypted documents will also be stored on the server, we can assume that the document sizes and identifiers will also be leaked. Therefore we choose to include these in the trace.[2]

[2] On the other hand, if we wish not to disclose the size of the documents, this can be trivially

Definition 5.3.6 (Trace)**.** *Let* Δ *be a dictionary and* $\mathbf{D} \subseteq 2^{\Delta}$ *be a document collection over* Δ*. The trace induced by a* q*-query history* $\mathrm{H} = (\mathbf{D}, w_1, \ldots, w_q)$ *over* $\mathbf{D}$*, is a sequence* $\tau(\mathrm{H}) = (|D_1|, \ldots, |D_n|, \alpha(\mathrm{H}), \sigma(\mathrm{H}))$ *comprised of the lengths of the documents in* $\mathbf{D}$*, and the access and search patterns induced by* H*.*

5.3.1 Our Definitions

In this section we present our definitions of security for SSE. We are now ready to state our first security definition for SSE. First, we assume that the adversary generates its history "at once". In other words, it is not allowed to see the index of the document collection or the trapdoors of any keywords it chooses before it has generated the entire history. We call such an adversary *non-adaptive*.

Definition 5.3.7 (Non-adaptive semantic security)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trapdoor}, \mathsf{Search})$ *be a* SSE *scheme,* $k \in \mathbb{N}$ *be the security parameter,* $\mathcal{A}$ *be an adversary,* $\mathcal{S}$ *be a simulator, and* z *be a string. We define* $\mathbf{Real}_{\Sigma,\mathcal{A},z}(k)$ *and* $\mathbf{Sim}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)$ *as the following two probabilistic experiments:*

achieved by "padding" each plaintext document such that all documents have a fixed size and omitting the document sizes from the trace.

$\mathbf{Real}_{\Sigma,\mathcal{A},z}(k)$

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$

$(\mathrm{H}, \mathrm{ST}) \leftarrow \mathcal{A}(1^k, z)$

parse H *as* $\mathrm{H} = (\mathbf{D}, w_1, \ldots, w_q)$

$\mathrm{I} \leftarrow \mathsf{Index}_{\mathrm{K}}(\mathbf{D})$

for $1 \leq i \leq q$,

$\quad \mathrm{T}_i \leftarrow \mathsf{Trapdoor}_{\mathrm{K}}(w_i)$

output $\mathrm{V} = (\mathrm{I}, \mathrm{T}_1, \ldots, \mathrm{T}_q)$ *and* ST

$\mathbf{Sim}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)$

$(\mathrm{H}, \mathrm{ST}) \leftarrow \mathcal{A}(1^k, z)$

$\mathrm{V} \leftarrow \mathcal{S}(1^k, \tau(\mathrm{H}), z)$

output V *and* ST

We say that Σ *is semantically secure if for all non-uniform polynomial-time adversaries* $\mathcal{A}$*, there exists a non-uniform polynomial-time simulator* $\mathcal{S}$ *such that, for all polynomial-length strings* z*, and all non-uniform polynomial-time distinguishers* $\mathcal{D}$*,*

$$|\Pr[\,\mathcal{D}(\mathrm{V}, \mathrm{ST}, z) = 1 : (\mathrm{V}, \mathrm{ST}) \leftarrow \mathbf{Real}_{\Sigma,\mathcal{A},z}(k)\,] -$$

$$\Pr[\,\mathcal{D}(\mathrm{V}, \mathrm{ST}, z)) = 1 : (\mathrm{V}, \mathrm{ST}) \leftarrow \mathbf{Sim}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)\,]|$$

is negligible in k*, where the probabilities are over the coins of* Gen *and* Index*.*

We now turn to our adaptive security definition. It is similar to Definition 5.3.7 with the exception that we allow the adversary to choose its history adaptively and we require the simulator to simulate the history as it is generated.

Definition 5.3.8 (Adaptive semantic security)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trapdoor}, \mathsf{Search})$ *be a SSE scheme,* $k \in \mathbb{N}$ *be the security parameter,* $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$ *be an adversary such that* $q \in \mathbb{N}$*,* $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$ *be a simulator, and* z *be a string. We define* $\mathbf{Real2}_{\Sigma,\mathcal{A},z}(k)$ *and* $\mathbf{Sim2}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)$ *as the following probabilistic experiments:*

$\mathbf{Real2}_{\Sigma,\mathcal{A},z}(k)$

$\mathrm{K} \leftarrow \mathsf{Gen}(1^k)$
$(\mathbf{D}, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_0(1^k)$
$\mathrm{I} \leftarrow \mathsf{Index}_{\mathrm{K}}(\mathbf{D})$
$(w_1, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathrm{ST}_{\mathcal{A}}, \mathrm{I})$
$\mathrm{T}_1 \leftarrow \mathsf{Trapdoor}_{\mathrm{K}}(w_1)$
for $2 \leq i \leq q$,
  $(w_i, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_i(\mathrm{ST}_{\mathcal{A}}, \mathrm{I}, \mathrm{T}_1, \ldots, \mathrm{T}_{i-1})$
  $\mathrm{T}_i \leftarrow \mathsf{Trapdoor}_{\mathrm{K}}(w_i)$
output $\mathrm{v} = (\mathrm{I}, \mathrm{T}_1, \ldots, \mathrm{T}_q)$ *and* $\mathrm{ST}_{\mathcal{A}}$

$\mathbf{Sim2}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)$

$(\mathbf{D}, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_0(1^k)$
$(\mathrm{I}, \mathrm{ST}_{\mathcal{S}}) \leftarrow \mathcal{S}_0(1^k, \tau(\mathbf{D}), z)$
$(w_1, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_1(\mathrm{ST}_{\mathcal{A}}, \mathrm{I})$
$(\mathrm{T}_1, \mathrm{ST}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(\mathrm{ST}_{\mathcal{S}}, \tau(\mathbf{D}, w_1), z)$
for $2 \leq i \leq q$,
  $(w_i, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathcal{A}_i(\mathrm{ST}_{\mathcal{A}}, \mathrm{I}, \mathrm{T}_1, \ldots, \mathrm{T}_{i-1})$
  $(\mathrm{T}_i, \mathrm{ST}_{\mathcal{S}}) \leftarrow \mathcal{S}_i(\mathrm{ST}_{\mathcal{S}}, \tau(\mathbf{D}, w_1, \ldots, w_i), z)$
output $\mathrm{v} = (\mathrm{I}, \mathrm{T}_1, \ldots, \mathrm{T}_q)$ *and* $\mathrm{ST}_{\mathcal{A}}$

We say that Σ *is adaptively semantically secure if for all non-uniform polynomial-time adversaries* $\mathcal{A} = (\mathcal{A}_0, \ldots, \mathcal{A}_q)$ *such that* $q = \mathsf{poly}(k)$*, there exists a non-uniform polynomial-time simulator* $\mathcal{S} = (\mathcal{S}_0, \ldots, \mathcal{S}_q)$*, such that for all polynomial-length strings* z*, and all non-uniform polynomial-time distinguishers* $\mathcal{D}$*,*

$$|\Pr[\mathcal{D}(\mathrm{v}, z, \mathrm{ST}_{\mathcal{A}}) = 1 : (\mathrm{v}, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathbf{Real2}_{\Sigma,\mathcal{A},z}(k)] -$$

$$\Pr[\mathcal{D}(\mathrm{v}, z, \mathrm{ST}_{\mathcal{A}}) = 1 : (\mathrm{v}, \mathrm{ST}_{\mathcal{A}}) \leftarrow \mathbf{Sim2}_{\Sigma,\mathcal{S},z}(k)]|$$

is negligible in k*, where the probabilities are over the coins of* Gen*, and* Index*.*

5.4 Our Constructions

We now present our SSE constructions, and state their security in terms of the definitions presented in Section 5.3. We start by introducing some additional notation and the data structures used by the constructions. Let $\delta(\mathbf{D}) \subseteq \Delta$ be the set of distinct keywords in the document collection $\mathbf{D}$, and $\delta(D) \subseteq \Delta$ be the set of distinct keywords

in the document $D \in \mathbf{D}$. We assume that keywords in Δ can be represented using at most ℓ bits. Also, recall that $\mathbf{D}(w)$ is the set of identifiers of documents in $\mathbf{D}$ that contain keyword w ordered in lexicographic order.

We use several data structures, including arrays, linked lists and look-up tables. Given an array $\mathtt{A}$, we refer to the element at address i in $\mathtt{A}$ as $\mathtt{A}[i]$, and to the address of element x relative to $\mathtt{A}$ as $\mathsf{addr}_{\mathtt{A}}(x)$. So if $\mathtt{A}[i] = x$, then $\mathsf{addr}_{\mathtt{A}}(x) = i$. In addition, a linked list $\mathtt{L}$ of n nodes that is stored in an array $\mathtt{A}$ is a sequence of nodes $\mathtt{N}_i = \langle v_i, \mathsf{addr}_{\mathtt{A}}(\mathtt{N}_{i+1})\rangle$, where $1 \leq i \leq n$, and where v_i is an arbitrary string and $\mathsf{addr}_{\mathtt{A}}(\mathtt{N}_{i+1})$ is the memory address of the next node in the list. We denote by $\#\mathtt{L}$ the number of nodes in the list $\mathtt{L}$.

5.4.1 An Efficient Non-Adaptively Secure Construction

We first give an overview of our one-round non-adaptively secure SSE construction. We associate a single index I with a document collection $\mathbf{D}$. The index I consists of two data structures:

$\mathtt{A}$: an array in which for all $w \in \delta(\mathbf{D})$, we store an encryption of the set $\mathbf{D}(w)$.

$\mathtt{T}$: a look-up table in which which for all $w \in \delta(\mathbf{D})$, we store information that enables one to locate and decrypt the appropriate element from $\mathtt{A}$.

For each distinct keyword $w_i \in \delta(\mathbf{D})$, we start by creating a linked list $\mathtt{L}_i$ where each node contains the identifier of a document in $\mathbf{D}(w_i)$. We then store all the nodes

of all the lists in the array A "scrambled" in a random order and encrypted with randomly generated keys. Before encryption, the j-th node of $\mathtt{L}_i$ is augmented with a pointer (with respect to A) to the $(j+1)$-th node of $\mathtt{L}_i$, together with the key used to encrypt it. In this way, given the location in A and the decryption key for the first node of a list $\mathtt{L}_i$, the server will be able to locate and decrypt all the nodes in $\mathtt{L}_i$. Note that by storing the nodes of all lists $\mathtt{L}_i$ in a random order, the length of each individual $\mathtt{L}_i$ is hidden.

We then build a look-up table T that allows one to locate and decrypt the first node of each list $\mathtt{L}_i$. Each entry in T corresponds to a keyword $w_i \in \Delta$ and consists of a pair <address,value>. The field value contains the location in A and the decryption key for the first node of $\mathtt{L}_i$. value is itself encrypted using the output of a pseudo-random function. The other field, address, is simply used to locate an entry in T. The look-up table T is managed using *indirect addressing* (described below).

The client generates both A and T based on the plaintext document collection $\mathbf{D}$, and stores them on the server together with the *encrypted* documents. When the user wants to retrieve the documents that contain keyword w_i, it computes the decryption key and the address for the corresponding entry in T and sends them to the server. The server locates and decrypts the given entry of T, and gets a pointer to and the decryption key for the first node of $\mathtt{L}_i$. Since each node of $\mathtt{L}_i$ contains a pointer to the next node, the server can locate and decrypt all the nodes of $\mathtt{L}_i$, revealing the identifiers in $\mathbf{D}(w_i)$.

Efficient storage and access of sparse tables. We describe the indirect addressing method that we use to efficiently manage look-up tables. The entries of a look-up table $\mathtt{T}$ are tuples $\mathtt{<address,value>}$ in which the $\mathtt{address}$ field is used as a *virtual address* to locate the entry in $\mathtt{T}$ that contains some $\mathtt{value}$ field. Given a parameter ℓ, a virtual address is from a domain of exponential size, i.e., from $\{0,1\}^\ell$. However, the maximum number of entries in a look-up table will be polynomial in ℓ, so the number of virtual addresses that are used is $\mathsf{poly}(k)$. If, for a table $\mathtt{T}$, the $\mathtt{address}$ field is from $\{0,1\}^\ell$, the $\mathtt{value}$ field is from $\{0,1\}^v$ and there are at most s entries in $\mathtt{T}$, then we say $\mathtt{T}$ is a $(\{0,1\}^\ell \times \{0,1\}^v \times s)$ look-up table.

Let Addr be the set of virtual addresses that are used for entries in a look-up table $\mathtt{T}$. We can efficiently store $\mathtt{T}$ such that, when given a virtual address, it returns the associated $\mathtt{value}$ field. We achieve this by organizing Addr in a so-called *FKS dictionary* [33], an efficient data structure for storage of sparse tables that requires $O(|\mathsf{Addr}|)$ storage and $O(1)$ look-up time. In other words, given some virtual address a, we are able to tell if $a \in \mathsf{Addr}$ and if so, return the associated $\mathtt{value}$ in constant look-up time. Addresses that are not in Addr are considered undefined.

Padding. Consistent with our security definitions, our construction reveals only the access pattern, the search pattern, the total size of the encrypted document collection, and the number of documents it contains. To achieve this, a certain amount of padding to the array and the table are necessary. To see why, recall that the array $\mathtt{A}$ stores a collection of linked lists $(\mathtt{L}_1, \ldots, \mathtt{L}_{|\delta(\mathbf{D})|})$, where each $\mathtt{L}_i$ contains the identifiers

of all the documents that contain the keyword $w_i \in \delta(\mathbf{D})$. Note that the number of non-empty cells in $\mathtt{A}$, denoted by $\#\mathtt{A}$, is equal to the total number of nodes contained in all the lists. In other words,

$$\#\mathtt{A} = \sum_{w_i \in \delta(\mathbf{D})} \#\mathtt{L}_i.$$

Notice, however, that this is also equal to the sum (over all the documents) of the number of distinct keywords found in each document. In other words,

$$\#\mathtt{A} = \sum_{w_i \in \delta(\mathbf{D})} \#\mathtt{L}_i = \sum_{i=1}^{n} |\delta(D_i)|.$$

Let $\#\mathbf{D}$ be the number of (non-distinct) words in the document collection $\mathbf{D}$. Clearly, if

$$\sum_{i=1}^{n} |\delta(D_i)| < \#\mathbf{D},$$

then there exists at least one document in $\mathbf{D}$ that contains a certain word more than once. Our goal, therefore, will be to pad $\mathtt{A}$ so that this leakage does not occur.

In practice, the adversary (i.e., the server) will not know $\#\mathbf{D}$ explicitly, but it can approximate it as follows using the encrypted documents it stores. Let s be the total size of the encrypted document collection in "min-units", where a min-unit is the smallest possible size for a keyword (e.g., one byte). Also, let s' be total size of the encrypted document collection in "max-units", where a max-unit is the largest possible size for a keyword (e.g., ten bytes). It follows then that

$$s' \leq \#\mathbf{D} \leq s.$$

From the previous argument, it follows that $\mathtt{A}$ must be padded so that $\#\mathtt{A}$ is at least s'. Note, however, that setting $\#\mathtt{A} = s'$ is not sufficient since an adversary will know that in all likelihood $\#\mathbf{D} > s'$. We therefore pad $\mathtt{A}$ so that $\#\mathtt{A} = s$.

We follow the same line of reasoning for the look-up table $\mathtt{T}$, which has at least one entry for each distinct keyword in $\mathbf{D}$. To avoid revealing the number of distinct keywords in $\mathbf{D}$, we add an additional $|\Delta| - |\delta(\mathbf{D})|$ entries in $\mathtt{T}$ filled with random values so that the total number of entries is always equal to $|\Delta|$.

Our construction in detail. We are now ready to proceed to the details of the construction. Let k be security parameters and let $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure private-key encryption scheme with $\mathsf{Enc} : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^*$. In addition, we make use of a pseudo-random function f and two pseudo-random permutations Ψ and Φ with the following parameters:

$$f : \{0,1\}^k \times \{0,1\}^\ell \rightarrow \{0,1\}^{k+\log_2(s)};$$

$$\Psi : \{0,1\}^k \times \{0,1\}^\ell \rightarrow \{0,1\}^\ell;$$

$$\Phi : \{0,1\}^k \times \{0,1\}^{\log_2(s)} \rightarrow \{0,1\}^{\log_2(s)},$$

where s is as above [3]. Let $\mathtt{A}$ be an array with s non-empty cells, and let $\mathtt{T}$ be a $(\{0,1\}^\ell \times \{0,1\}^{k+\log_2(s)} \times |\Delta|)$ look-up table, managed using indirect addressing as described previously. Our construction is described in Fig. 5.1.

[3] If the documents are not encrypted with a length preserving encryption scheme or if they are compressed before encryption, then s is the maximum between the total size of the plaintext $\mathbf{D}$ and the total size of the encrypted $\mathbf{D}$.

Theorem 5.4.1. *If f is a pseudo-random function, if Ψ and Φ are pseudo-random permutations, and if Σ is CPA-secure, then the construction described in Figure 5.1 is non-adaptively secure.*

Proof. We describe a non-uniform polynomial-time simulator $\mathcal{S}$ such that for all non-uniform polynomial-time adversaries $\mathcal{A}$, for all polynomial-length strings z, the outputs of $\mathbf{Real}_{\Sigma,\mathcal{A},z}(k)$ and $\mathbf{Sim}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)$ are computationally indistinguishable.

Let $\mathcal{A}$ and z be as above and consider the simulator $\mathcal{S}$ that given z and the trace of a q-query history H, generates a string $\mathrm{v}^* = (\mathrm{I}^*, \mathrm{T}_1^*, \ldots, \mathrm{T}_n^*, r^*)$, where $\mathrm{I}^* = (\mathtt{A}^*, \mathtt{T}^*)$, as follows:

1. (Simulating $\mathtt{A}^*$) if $q = 0$ then for $1 \leq i \leq s$, $\mathcal{S}$ sets $\mathtt{A}^*[i]$ to a string of length $\log_2(n) + k + \log_2(s)$ selected uniformly at random. If $q \geq 1$, it sets $|\delta(\mathbf{D})| = q$ and runs Step 4 of the Index algorithm on the sets $\mathbf{D}(w_1)$ through $\mathbf{D}(w_q)$ using different random strings of size $\log_2(s)$ instead of $\Phi(\mathsf{ctr})$. Note that $\mathcal{S}$ knows $\mathbf{D}(w_1)$ through $\mathbf{D}(w_q)$ from the trace it receives.

2. (Simulating $\mathtt{T}^*$) if $q = 0$ then for $1 \leq i \leq |\Delta|$, $\mathcal{S}$ generates pairs (a_i^*, c_i^*) such that the a_i^* are distinct strings of length ℓ chosen uniformly at random, and the c_i^* are strings of length $\log_2(s) + k$ also chosen uniformly at random. If $q \geq 1$, then for $1 \leq i \leq q$, $\mathcal{S}$ generates random values β_i^* of length $\log_2(s) + k$ and a_i^* of length ℓ, and sets

$$\mathtt{T}^*[a_i^*] = \langle \mathsf{addr}_{\mathtt{A}^*}(\mathtt{N}_{i,1}) || \mathrm{K}_{i,0} \rangle \oplus \beta_i^*.$$

It then inserts dummy entries into the remaining entries of $\mathtt{T}^*$. So, in other words, $\mathcal{S}$ runs Step 6 of the Index algorithm with $|\delta(\mathbf{D})| = q$, using $\mathtt{A}^*$ instead of $\mathtt{A}$, and using β_i^* and a_i^* instead of $f_y(w_i)$ and $\Psi_z(w_i)$, respectively.

3. (Simulating T_i^*) it sets $\mathrm{T}_i^* = (a_i^*, \beta_i^*)$

It follows by construction that searching on I^* using trapdoors T_i^* will yield the expected search outcomes.

Let v be the outcome of a $\mathbf{Real}_{\Sigma,\mathcal{A},z}(k)$ experiment. We now claim that no non-uniform polynomial-time $\mathcal{D}$ can distinguish between the distributions v^* and v, otherwise, by a standard hybrid argument, $\mathcal{D}$ could distinguish between at least one of the elements of v and its corresponding element in v^*. We argue that this is not possible by showing that each element of v^* is computationally indistinguishable from its corresponding element in v.

1. ($\mathtt{A}$ and $\mathtt{A}^*$) Recall that $\mathtt{A}$ consists of s' semantically secure ciphertexts and $s - s'$ random strings of the same size. If $q = 0$, $\mathtt{A}^*$ consists of all random strings. While if $q \geq 1$, $\mathtt{A}^*$ consists of q semantically secure ciphertexts and $s - q$ random strings of the same size. In either case, it follows that each element in $\mathtt{A}^*$ will be indistinguishable from its counterpart in $\mathtt{A}$.

2. ($\mathtt{T}$ and $\mathtt{T}^*$) Recall that $\mathtt{T}$ consists of $|\delta(\mathbf{D})|$ ciphertexts, c_i, generated by XORing a message with the output of f, and of $|\Delta| - |\delta(\mathbf{D})|$ random values of size $k + \log_2(s)$. If $q = 0$, $\mathtt{T}^*$ consists of all random values. While if $q \geq 1$, $\mathtt{T}^*$ consists

of q ciphertexts generated by XOR-ing a message with a random string β_i^* of length $k + \log_2(s)$, and $|\Delta| - q$ random strings of the same length. It follows that, in either case, each element of $\mathtt{T}$ indistinguishable from its counterpart in $\mathtt{T}^*$.

3. (T_i and T_i^*) for $1 \leq i \leq q$, T_i is indistinguishable from T_i^*, otherwise one could distinguish between the output of Ψ and a random string of size ℓ or between the output of f and a random string of size $k + \log_2(s)$.

■

Regarding efficiency, we remark that each query takes only one round, and $O(1)$ message size. In terms of storage, the demands are $O(1)$ on the user and $O(s)$ on the server; more specifically, in addition to the encrypted $\mathbf{D}$, the server stores the index I, which has size $O(s)$, and the look-up table $\mathtt{T}$, which has size $O(|\Delta|)$. Since the size of the encrypted documents is $O(s)$, accommodating the auxiliary data structures used for searching does not change (asymptotically) the storage requirements for the server. The user spends $O(1)$ time to compute a trapdoor, while for a query for keyword w, the server spends time proportional to $|\mathbf{D}(w)|$.

5.4.2 An Adaptively Secure Construction

While our first construction is efficient, it is only proven secure against non-adaptive adversaries. We now show a second construction which achieves semantic

security against adaptive adversaries at the price of requiring higher communication size per query and more storage on the server. Asymptotically, however, the costs are the same.

The difficulty of proving our first construction secure against an adaptive adversary stems from the difficulty of simulating in advance an index for the adversary that will be consistent with future unknown queries. Given the intricate structure of the construction, with each keyword having a corresponding linked list whose nodes are stored encrypted and in a random order, building an index that allows for such a simulation seems challenging. We circumvent this problem as follows.

For a keyword w and an integer j, we derive a *label* for w by concatenating w with j, where j is first converted to a string of characters. So, for example, if w is the keyword "coin" and $j = 1$, then $w||j$ is the string "coin1". We define the *family* of a keyword $w \in \delta(\mathbf{D})$ to be the set of labels $\text{FAM}_w = \{w||j : 1 \leq j \leq |\mathbf{D}(w)|\}$. So if the keyword "coin" appears in three documents, then $\text{FAM}_w = \{$"coin1", "coin2", "coin3"$\}$. Note that the maximum size of a keyword's family is n, i.e., the number of documents in the collection. We associate with the document collection $\mathbf{D}$ an index I, that consists of a look-up table `T`. For each label in a keyword's family, we add an entry in `T` whose `value` field is the identifier of the document that contains an instance of w. So for each $w \in \delta(\mathbf{D})$, instead of keeping a list we simply derive the family FAM_w and enter into the table each label together with the identifier of a document in $\delta(\mathbf{D})$. So if "coin" is contained in documents (D_5, D_8, D_9), then we enter

the tuples <"coin1",5>, <"coin2",8>, <"coin3",9>. In order to hide the number of distinct keywords in each document, we pad the look-up table so that the identifier of each document appears in the same number of entries. To search for the documents that contain w, it now suffices to search for all the labels in w's family. Since each label is unique, a search for it "reveals" a single document identifier. Translated to the proof, this will allow the simulator to construct an index for the adversary that is indistinguishable from a real index, even before it knows any of the adversary's queries.

Recall that $\delta(\mathbf{D})$ is the set of distinct keywords that exist in $\mathbf{D}$. Let max be the maximum number of distinct keywords that can fit in the largest document in $\mathbf{D}$ (assuming the minimum size for a keyword is one byte). We give an algorithm to determine max, given the size (in bytes) of the largest document in $\mathbf{D}$, which we denote MAX. In step 1 we try to fit the maximum number of distinct 1-byte keywords; there are 2^8 such keywords, which gives a total size of 256 bytes ($2^8 \cdot 1$ bytes). If MAX > 256, then we continue to step 2. In step 2 we try to fit the maximum number of distinct 2-byte keywords; there are 2^{16} such keywords, which gives a total size of 131328 bytes ($2^8 \cdot 1 + 2^{16} \cdot 2$ bytes). In general, in step i we try to fit the maximum number of distinct i-byte keywords, which is $2^{8 \cdot i}$. We continue similarly until step i when MAX becomes smaller than the total size accumulated so far. Then we go back to step $i-1$ and try to fit as many $(i-1)$-byte distinct keywords as possible in a document of size MAX. For example, when the largest document in $\mathbf{D}$ has size MAX $= 1$ MByte, we

can fit at most $\texttt{max} = 355349$ distinct keywords (2^8 distinct 1-byte keywords + 2^{16} distinct 2-byte keywords + 289557 distinct 3-byte keywords). Note that $\texttt{max}$ cannot be larger than $|\Delta|$; thus, if we get a value for $\texttt{max}$ (using the previously described algorithm) that is larger than $|\Delta|$, then we set $\texttt{max} = |\Delta|$.

Let k be security parameter and $s = \texttt{max} \cdot n$, where n is the number of documents in $\mathbf{D}$. Recall that keywords in Δ can be represented using at most ℓ bits. We use a pseudo-random permutation $\Psi : \{0,1\}^k \times \{0,1\}^{p+\log_2(n+\texttt{max})} \rightarrow \{0,1\}^{p+\log_2(n+\texttt{max})}$. Let $\texttt{T}$ be a $\left(\{0,1\}^{p+\log_2(n+\texttt{max})} \times \{0,1\}^{\log_2(n)} \times s\right)$ look-up table, managed using indirect addressing. The construction is described in Figure 5.2.

Theorem 5.4.2. *If Ψ is a pseudo-random permutation, then the construction described in Figure 5.2 is adaptively secure.*

Proof. We describe a non-uniform polynomial-time simulator $\mathcal{S} = (\mathcal{S}_0, \ldots \mathcal{S}_q)$ such that for all non-uniform polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_q)$, for all polynomial-length strings z, the outputs of $\mathbf{Real2}_{\Sigma,\mathcal{A},z}(k)$ and $\mathbf{Sim2}_{\Sigma,\mathcal{A},\mathcal{S},z}(k)$ are computationally indistinguishable.

Let $\mathcal{A}$ and z be as above and consider the simulator $\mathcal{S}$ that works as follows.

$\mathcal{S}_0(1^k, \tau(\mathbf{D}), z)$: it computes $\texttt{max}$ using the algorithm described above. Note that it can do this since it knows the size of all the documents from the trace of $\mathbf{D}$. It then sets I^* to be a $\left(\{0,1\}^{\ell+\log_2(n+\texttt{max})} \times \{0,1\}^{\log_2(n)} \times s\right)$ look-up table, where $s = \texttt{max} \cdot n$, with $\texttt{max}$ copies of each document's identifier inserted at

random locations. $\mathcal{S}$ then includes I^* in $\mathrm{ST}_{\mathcal{S}}$ and outputs $(\mathrm{I}^*, \mathrm{ST}_{\mathcal{S}})$. Clearly, I^* is indistinguishable from a real index otherwise one could distinguish between the output of Ψ and a random string of size $\ell + \log_2(n + \mathtt{max})$.

$\mathcal{S}_1(\mathrm{ST}_{\mathcal{S}}, \tau(\mathbf{D}, w_1), z)$: Recall that $\mathbf{D}(w_i) = (\mathbf{D}(w_i||1), \ldots, \mathbf{D}(w_i||n))$. Note that each $\mathbf{D}(w_i||j)$, for $1 \leq j \leq n$, contains only one document identifier which we refer to as $\mathsf{id}(D_{i,j})$. For all $1 \leq j \leq n$, $\mathcal{S}$ randomly picks an address $\mathtt{addr}_j$ from I^* such that $\mathrm{I}^*[\mathtt{addr}_j] = \mathsf{id}(D_{i,j})$, making sure that all $\mathtt{addr}_j$ are pairwise distinct. It then sets $\mathrm{T}_i^* = (\mathtt{addr}_1, \ldots, \mathtt{addr}_n)$. Also, $\mathcal{S}$ remembers the association between T_t^* and w_i by including it in $\mathrm{ST}_{\mathcal{S}}$. It then outputs $(\mathrm{T}_1^*, \mathrm{ST}_{\mathcal{S}})$.

It is easy to see that trapdoor T_i^* is indistinguishable from the trapdoors T_i, otherwise one could distinguish between the output of Ψ and a random string of size $\ell + \log_2(n + \mathtt{max})$.

$\mathcal{S}_i(\mathrm{ST}_{\mathcal{S}}, \tau(\mathbf{D}, w_1, \ldots, w_i), z)$ for $2 \leq i \leq q$: first $\mathcal{S}$ checks whether (the unknown) w_i has appeared before. This can be done done by checking whether there exists a $1 \leq j \leq i-1$ such that $\sigma[i, j] = 1$. If w_i has not previously appeared, then $\mathcal{S}$ generates a trapdoor the same way $\mathcal{S}_1$ does. On the other hand, if w_i did previously appear, then $\mathcal{S}$ retrieves the trapdoor previously used for w_i and uses it as T_i^*. $\mathcal{S}$ outputs $(\mathrm{T}_i^*, \mathrm{ST}_S)$, which is clearly indistinguishable from T_i.

■

Just like our non-adaptively secure scheme, this construction requires one round

of communication for each query and an amount of computation on the server proportional with the number of documents that contain the query (i.e., $O(|\mathbf{D}(w)|)$. Similarly, the storage and computational demands on the user are $O(1)$. The communication is equal to n and the storage on the server is increased by a factor of `max` when compared to out first construction . We note that the communication cost can be reduced if in each entry of `T` corresponding to an element in some keyword w's family, we also store $|\mathbf{D}(w)|$ in encrypted form. In this way, after searching for a label in w's family, the user will know $|\mathbf{D}(w)|$ and can derive FAM_w. The user can then send in a single round all the trapdoors corresponding to the remaining labels in w's family.

5.4.3 Secure updates

We allow for secure updates to the document collection in the sense defined by Chang and Mitzenmacher [22]: each time the user adds a new set ζ of encrypted documents, ζ is considered a separate document collection. Old trapdoors cannot be used to search newly submitted documents, as the new documents are part of a collection indexed using different secrets. If we consider the submission of the original document collection an update, then after u updates, there will be u document collections stored on the server. In the previously proposed solution [22], the user sends a pseudo-random seed for each document collection, which implies that the trapdoors have length $O(u)$. We propose a solution that achieves better bounds

for the length of trapdoors (namely $O(\log u)$) and for the amount of computation at the server. For applications where the number of queries dominates the number of updates, our solution may significantly reduce the communication size and the server's computation.

When the user performs an update, i.e., submits a set ζ^a of new documents, the server checks it there exists (from previous updates) a document collection ζ^b, such that $|\zeta^b| \leq |\zeta^a|$. If so, the server sends back ζ^b and the user combines ζ^a and ζ^b into a single collection ζ^c with $|\zeta^a| + |\zeta^b|$ documents. The user then computes an index for ζ^c. The server stores the combined document collection ζ^c and its index I_c, and deletes the document collections ζ^a, ζ^b and their indexes $\mathrm{I}_a, \mathrm{I}_b$. Note that ζ^c and its index I_c will not reveal anything more than what was already revealed by the ζ^a, ζ^b and their indexes $\mathrm{I}_a, \mathrm{I}_b$, since one can trivially reduce the security of the combined collection to the security of the composing collections.

Next, we analyze the number of document collections that results after u updates using the method proposed above. Without loss of generality, we assume that each update consists of one new document. Then, it can be shown that after u updates, the number of document collections is given by the the Hamming weight of $f(u)$. Note that $f(u) \in [1, \lfloor \log(u+1) \rfloor]$. This means that after u updates, there will be at most $\log(u)$ document collections, thus the queries sent by the user have size $O(\log u)$ and the search can be done in $O(\log u)$ by the server (as opposed to $O(u)$ in [22]).

5.5 Multi-User Searchable Encryption

In this section we consider a natural extension of SSE to the setting where a user owns a document collection, but an arbitrary group of users can submit queries to search the collection. A familiar question arises in this new setting, that of managing access privileges while preserving privacy with respect to the server. We first present a definition of a multi-user searchable encryption scheme (MSSE) and some of its desirable security properties, followed by an efficient construction which, in essence, combines a single-user SSE scheme with a broadcast encryption [30] (BE) scheme. Let N denote the set of all possible users, and $\mathsf{G} \subseteq \mathsf{N}$ the set of users that are currently authorized to search.

Definition 5.5.1 (Multi-user searchable symmetric encryption scheme)**.** *A multi-user SSE scheme is a collection of seven polynomial-time algorithms* $\Omega = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Add}, \mathsf{Revoke}^O, \mathsf{Revoke}^S, \mathsf{Trapdoor}, \mathsf{Search})$ *such that,*

$\mathrm{K}_O \leftarrow \mathsf{Gen}(1^k)$*: is a probabilistic key generation algorithm that is run by the owner* O *to set up the scheme. It takes as input a security parameter* k*, and outputs an owner secret key* K_O*.*

$\mathrm{I} \leftarrow \mathsf{Index}(\mathrm{K}_O, \mathbf{D})$*: is run by the owner* O *to generate indexes. It takes as input the owner's secret key* SK_O *and a document collection* $\mathbf{D}$*, and outputs an index* I*.*

$\mathrm{K}_U \leftarrow \mathsf{Add}(\mathrm{K}_O, U)$*: is run by* O *whenever it wishes to add a user to the group* G*. It takes as input the owner's secret key* K_O *and a user* U*, and outputs* U*'s secret*

key, K_U*.*

$m \leftarrow \mathsf{Revoke}^O(\mathrm{K}_O, U)$*: is run by* O *server whenever it wishes to revoke a user from* G*. It takes as input the owner's secret key* K_O *and a user* U*, and outputs a revocation message* m*.*

$b \leftarrow \mathsf{Revoke}^S(U, m)$*: is run by the server* S *in order to revoke a user from* G*. It takes as input a user* U *and message* m*, and outputs a bit* b *that signifies success or failure.*

$\mathrm{T}_{U,w} \leftarrow \mathsf{Trapdoor}(\mathrm{K}_U, w)$*: is run by a user (including* O*) in order to generate a trapdoor for a keyword. It takes as input a user* U*'s secret key* K_U *and a keyword* w*, and outputs a trapdoor* $\mathrm{T}_{U,w}$*.*

$X \leftarrow \mathsf{Search}(\mathrm{I}, \mathrm{T}_{U,w})$*: is run by the server* S *in order to search for the documents in* $\mathbf{D}$ *that contain keyword* w*. It takes as input an index* I *and a trapdoor* $\mathrm{T}_{U,w}$ *for keyword* w*, and outputs a set* X *of document identifiers.*

We briefly discuss here notions of security that a multi-user SSE scheme should achieve. It should be clear that the (semantic) security of a multi-user scheme can be reduced to the semantic security of the underlying single-user scheme. The reason is that in the multi-user case, just like in the single-user case, we are only concerned with providing security against the server. One distinct property in this new setting is that of *revocation*, which essentially states that a revoked user no longer be able to perform searches on the owner's documents.

Definition 5.5.2 (Revocation)**.** *Let* $\Omega = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Add}, \mathsf{Revoke}^O, \mathsf{Revoke}^S, \mathsf{Trapdoor}, \mathsf{Search})$ *be a multi-user SSE scheme,* $k \in \mathbb{N}$ *be the security parameter, and* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *be an adversary. We define* $\mathbf{Rev}_{\Omega,\mathcal{A}}(k)$ *as the following probabilistic experiment.*

$\mathbf{Rev}_{\Omega,\mathcal{A}}(k)$
$\mathrm{K}_O \leftarrow \mathsf{Gen}(1^k)$
$(\mathrm{ST}, \mathbf{D}) \leftarrow \mathcal{A}_1(1^k)$
$\mathrm{K}_{\mathcal{A}} \leftarrow \mathsf{Add}(\mathrm{K}_O, \mathcal{A})$
$\mathrm{I} \leftarrow \mathsf{Index}(\mathrm{K}_O, \mathbf{D})$
$\mathrm{ST} \leftarrow \mathcal{A}_2^{\mathsf{Search}(\mathrm{I},\cdot)}(\mathrm{ST}, \mathrm{K}_{\mathcal{A}})$
$m \leftarrow \mathsf{Revoke}^O(\mathrm{K}_O, \mathcal{A})$
$b \leftarrow \mathsf{Revoke}^S(\mathcal{A}, m)$
$(w, T') \leftarrow \mathcal{A}_3(\mathrm{ST})$
$X \leftarrow \mathsf{Search}(\mathrm{I}, \mathrm{T}')$
if $X = \mathbf{D}(w)$ *output* 1
else output 0

We say that Ω *achieves revocation if for all non-uniform polynomial-time adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\Pr\left[\,\mathbf{Rev}_{\Omega,\mathcal{A}}(k) = 1\,\right] \leq \mathsf{negl}(k),$$

where the probability is over the coins of Gen*,* Add*,* Revoke^O *and* Index*.*

Our construction makes use of a single-user SSE scheme and a broadcast encryption scheme. Let $\mathsf{BE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a broadcast encryption scheme. Recall that in broadcast encryption, a center starts out by running the Gen algorithm to generate long-lived secrets which it distributes to the set of authorized users. It encrypts a message m to a group G of privileged users that are allowed to access the message.

We denote this by $c \leftarrow \mathsf{Enc}_{\mathsf{G}}(m)$. The group G can be dynamically changing, as users can be added or removed from G. Although the encrypted message can be received by a set $\mathsf{N} \supseteq \mathsf{G}$ of receivers, only the users in G can recover the message. We denote this by $m \leftarrow \mathsf{Dec}_{\mathsf{G}}(c)$. When a user joins the system, it receives a set of secrets, referred to as *long-lived* secrets. The long-lived secrets are distinct for each user. Given an encrypted message, the long-lived secrets allow a user to decrypt it only if the user was non-revoked at the time the message was encrypted. We use off-the-shelf broadcast encryption as a building block in our multi-user secure index construction in order to efficiently manage user revocation.

Let $\Sigma = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Trapdoor}, \mathsf{Search})$ be a single-user SSE scheme and $\mathsf{BE} = (\mathsf{Gen}, \mathsf{Enc}_{\mathsf{G}}, \mathsf{Dec}_{\mathsf{G}})$ be a broadcast encryption scheme. We require standard security notions for broadcast encryption: namely, that in addition to being CPA-secure it provide revocation-scheme security against a coalition of all revoked users and that its key assignment algorithm satisfies key indistinguishability. Let N denote the set of all users, and $\mathsf{G} \subseteq \mathsf{N}$ the set of users (currently) authorized to search, and R denote the set of revoked users. Let Φ be a pseudo-random permutation such that $\Phi : \{0,1\}^k \times \{0,1\}^t \rightarrow \{0,1\}^t$, where t is the size of a trapdoor in the underlying single-user SSE scheme. Φ can be constructed using the techniques proposed by Black and Rogaway [15], which describe how to build pseudo-random permutations over domains of arbitrary size. Our multi-user construction $\Omega = (\mathsf{Gen}, \mathsf{Index}, \mathsf{Add}, \mathsf{Revoke}^O, \mathsf{Revoke}^S, \mathsf{Trapdoor}, \mathsf{Search})$ is described in detail in Figure 5.3.

The owner key is composed of a key K for the underlying single-user scheme, and a key r for the pseudo-random permutation Φ. To create an index, the owner O simply runs the single-user indexing algorithm, $\Sigma.\mathsf{Index}$, on the data collection and stores the index, together with the broadcast encryption of r on the server. To add a user U, the owner sends it the pair (K, r) together with the long-lived secrets for the broadcast encryption scheme.

To search for a keyword w, an authorized user U first retrieves $\mathsf{Enc}_{\mathsf{N}\setminus\mathsf{R}}(r)$ from the server and recovers r. It generates a single-user trapdoor T_w, encrypts it using Φ keyed with r, and sends it to the server. The server, upon receiving $\Phi_r(\mathrm{T}_w)$, recovers the trapdoor by computing $\mathrm{T}_w = \Phi_r^{-1}(\Phi_r(\mathrm{T}_w))$. The key r currently used for Φ is only known by the owner, by the set of currently authorized users and by the server. Each time a user is revoked, the owner picks a new r' and stores it on the server encrypted such that only non-revoked users can decrypt it. The server will use the new r' when inverting Φ for all subsequent queries. Since revoked users will not be able to recover r', with overwhelming probability, their queries will not yield a valid trapdoor after the server applies $\Phi_{r'}^{-1}$.

Notice that to give a user U permission to search through $\mathbf{D}$, the owner sends it all the secret information needed to perform searches in a single-user context. Note that O should possess an additional secret that will not be shared with U and that allows him to perform authentication with the server when he wants to update $\mathbf{D}$. This guarantees that only O can perform updates to $\mathbf{D}$. However, the extra layer

given by the pseudo-random permutation Φ, together with the guarantees offered by the broadcast encryption scheme and the assumption that the server is honest-but-curious, is what prevents users from performing successful searches once they are revoked.

We point out that users receive their long-lived secrets for the broadcast encryption scheme only when they are given authorization to search. So while a user U that has not joined the system yet could retrieve $\mathsf{Enc}_{\mathsf{N}\setminus\mathsf{R}}(r)$ from the server, since it does not know the long-lived secrets, it will not be able to recover r. Similarly, when a revoked user U retrieves $\mathsf{Enc}_{\mathsf{N}\setminus\mathsf{R}}(r)$ from the server, it cannot recover r because $U \in \mathsf{R}$. Moreover, even though a revoked user which has been re-authorized to search could recover (old) values of r that were used while he was revoked, these values are no longer of interest. The fact that backward secrecy is not needed for the BE scheme makes the Add algorithm more efficient, since it does not require the owner to send a message to the server.

Our multi-user construction is very efficient on the server side: when given a trapdoor, the server only needs to evaluate a pseudo-random permutation in order to determine if the user is revoked. If access control mechanisms were used instead for this step, a "heavier" authentication protocol would be required.

5.6 Conclusions

In this chapter, we considered the problem of symmetric searchable encryption, revisiting previous security definitions and pointing out some of their limitations. More precisely, we show that the current definitions only achieve what we call *non-adaptive* security, while the more natural usage of searchable encryption requires *adaptive* security. To address this we proposed two new definitions for SSE, one non-adaptive and one adaptive.

We also presented efficient constructions for both security notions based on any one-way function. Our first construction is the most efficient non-adaptive SSE scheme to date in terms of computation on the server, and incurs minimal cost for the user. Our second construction achieves adaptive security, which was not previously achieved by any constant-round solution.

Finally, we extend the problem of SSE to the multi-user setting, where a client wishes to allow an authorized group of users to search through its document collection. For the case of multi-user SSE we introduce an efficient transformation based on pseudo-random permutations and broadcast encryption that turns any single-user scheme into a multi-user scheme.

While our adaptively secure construction is asymptotically optimal in terms of computational complexity at the server, the underlying constants are prohibitive. Therefore an interesting open question is whether more efficient adaptively secure SSE schemes can be achieved. Other interesting directions for future work include

provisioning our constructions with more flexible types of queries such as: conjunctive queries which allow one to searches for keywords w_1 and w_2; disjunctive queries that allow one to search for w_1 or w_2; and fuzzy queries which allow one to search for any keyword "close" to w.

$\mathsf{Gen}(1^k)$: sample $\mathrm{K}_1, \mathrm{K}_2, \mathrm{K}_3 \xleftarrow{\$} \{0,1\}^k$ and output $\mathrm{K} = \langle \mathrm{K}_1, \mathrm{K}_2, \mathrm{K}_3 \rangle$.

$\mathsf{Index}_{\mathrm{K}}(\mathbf{D})$:

Initialization:

1. scan $\mathbf{D}$ and generate the set of distinct keywords $\delta(\mathbf{D})$
2. for all $w \in \delta(\mathbf{D})$, generate $\mathbf{D}(w)$
3. initialize a global counter $\mathsf{ctr} = 1$

Building the array $\mathtt{A}$:

4. for $1 \leq i \leq |\delta(\mathbf{D})|$, build a list $\mathtt{L}_i$ as follows
   (a) sample a key $\mathrm{K}_{i,0} \xleftarrow{\$} \{0,1\}^k$
   (b) for $1 \leq j \leq |\mathbf{D}(w_i)|$:
   - let $\mathsf{id}(D_{i,j})$ be the j^{th} identifier in $\mathbf{D}(w_i)$
   - sample a key $\mathrm{K}_{i,j} \xleftarrow{\$} \{0,1\}^k$ and create a node

$$\mathtt{N}_{i,j} = \langle \mathsf{id}(D_{i,j}) \| \mathrm{K}_{i,j} \| \Phi_{k_1}(\mathsf{ctr}+1) \rangle$$

   - encrypt the node $\mathtt{N}_{i,j}$ under key $\mathrm{K}_{i,j-1}$ and store it in $\mathtt{A}[\Phi_{\mathrm{K}_1}(\mathsf{ctr})]$
   - set $\mathsf{ctr} = \mathsf{ctr} + 1$

   (c) for the last node of $\mathtt{L}_i$ set the address of the next node to NULL
5. let $s' = \sum_{w_i \in \delta(\mathbf{D})} |\mathbf{D}(w_i)|$. If $s' < s$, then set remaining $s - s'$ entries of $\mathtt{A}$ to random values of the same size as the existing s' entries of $\mathtt{A}$.

Building the look-up table $\mathtt{T}$:

6. for all $w_i \in \delta(\mathbf{D})$, set $\mathtt{T}[\Psi_{\mathrm{K}_3}(w_i)] = \langle \mathsf{addr}_{\mathtt{A}}(\mathtt{N}_{i,1}) || \mathrm{K}_{i,0} \rangle \oplus f_y(w_i)$
7. if $|\delta(\mathbf{D})| < |\Delta|$, set the remaining $|\Delta| - |\delta(\mathbf{D})|$ entries to random values
8. output $\mathrm{I} = (\mathtt{A}, \mathtt{T})$.

$\mathsf{Trapdoor}_{\mathrm{K}}(w)$: output $\mathrm{T}_w = (\Psi_{\mathrm{K}_3}(w), f_{\mathrm{K}_2}(w))$.

$\mathsf{Search}(\mathrm{I}, \mathrm{T}_w)$:

1. Parse T_w as (γ, η), and retrieve $\theta = \mathtt{T}[\gamma]$. Parse $\theta \oplus \eta$ as $\langle \alpha || \mathrm{K}' \rangle$
2. Use the key K' to decrypt the list $\mathtt{L}$ starting with the node stored at address α in $\mathtt{A}$
3. Output the list of document identifiers contained in $\mathtt{L}$.

Figure 5.1: A non-adaptively secure SSE scheme

$\mathsf{Gen}(1^k)$: sample and output $\mathrm{K} \xleftarrow{\$} \{0,1\}^k$

$\mathsf{Index}_{\mathrm{K}}(\mathbf{D})$:

Initialization:

1. scan $\mathbf{D}$ and generate the set of distinct keywords $\delta(\mathbf{D})$
2. for all $w \in \delta(\mathbf{D})$, generate $\mathbf{D}(w)$ (i.e., the set of documents that contain w)

Building the look-up table I:

3. for $1 \leq i \leq |\delta(\mathbf{D})|$ and $1 \leq j \leq |\mathbf{D}(w_i)|$,
   (a) let $\mathsf{id}(D_{i,j})$ be the j^{th} identifier in $\mathbf{D}(w_i)$
   (b) set $\mathrm{I}[\Psi_{\mathrm{K}}(w_i||j)] = \mathsf{id}(D_{i,j})$
4. let $s' = \sum_{w_i \in \delta(\mathbf{D})} |\mathbf{D}(w_i)|$
5. if $s' < s$, then set values for the remaining $(s - s')$ entries in I such that for all documents $D \in \mathbf{D}$, the identifier $\mathsf{id}(D)$ appears exactly `max` times.
6. output I

$\mathsf{Trapdoor}_{\mathrm{K}}(w)$: output $\mathrm{T}_w = (\mathrm{T}_{w_1}, \ldots, \mathrm{T}_{w_n}) = (\Psi_{\mathrm{K}}(w||1), \ldots, \Psi_{\mathrm{K}}(w||n))$.

$\mathsf{Search}(\mathrm{I}, \mathrm{T}_w)$: for all $1 \leq i \leq n$, retrieve and output $\mathsf{id} = \mathrm{I}[\mathrm{T}_{w_i}]$

Figure 5.2: An adaptively secure SSE scheme

$\mathsf{Gen}(1^k)$: compute $\mathrm{K} \leftarrow \Sigma.\mathsf{Gen}(1^k)$ and sample $r \xleftarrow{\$} \{0,1\}^k$. Output $\mathrm{K}_O = (\mathrm{K}, r)$.

$\mathsf{Index}(\mathrm{K}_O, \mathbf{D})$: run $\mathrm{I} \leftarrow \Sigma.\mathsf{Index}(\mathrm{K}, \mathbf{D})$. Initialize BE and set $\mathsf{R} = \{\varnothing\}$. Send r and $\mathsf{Enc}_{\mathsf{N}}(r)$ to the server, and output I.

$\mathsf{Add}(\mathrm{K}_O, U)$: send $\mathrm{K}_U = (\mathrm{K}, r)$ to user U, where r is the current key used for Φ. Also send it the long-lived secrets needed for the BE scheme.

$\mathsf{Revoke}^O(\mathrm{K}_O, U)$: set $\mathsf{R} = \mathsf{R} \cup \{U\}$ and sample $r \xleftarrow{\$} \{0,1\}^k$. Output $m = \langle r, enc_{\mathsf{N}\setminus\mathsf{R}}(r)\rangle$.

$\mathsf{Revoke}^s(U, m)$: parse m into $\langle r', \mathsf{Enc}_{\mathsf{N}\setminus\mathsf{R}}(r')\rangle$ and overwrite the old values r and $\mathsf{Enc}_{\mathsf{N}\setminus\mathsf{R}}(r)$ with the new ones.

$\mathsf{Trapdoor}(\mathrm{K}_U, w)$: compute $T_w \leftarrow \Sigma.\mathsf{Trapdoor}(\mathrm{K}, w)$. Retrieve $\mathsf{Enc}_{\mathsf{N}\setminus\mathsf{R}}(r)$ from the server and use the long-lived BE secrets to recover r. Output $\Phi_r(T_w)$.

$\mathsf{Search}(\mathrm{I}, T_{U,w})$: compute $T_w \leftarrow \Phi_r^{-1}(T_{U,w})$. Run $\mathsf{Search}(\mathrm{I}, T_w)$ and return its output.

Figure 5.3: A multi-user SSE scheme

Chapter 6

Conclusion

The assumption that the computational devices used by honest parties have access to "secured" resources (i.e., resources that are outside of the adversary's control) does not hold in many practical settings. In fact, cryptographic protocols that are implemented on smartcards, or other embedded devices, are often broken by manipulating the device's source of randomness. Also, there are many settings in which the honest parties may wish to use resources from parties they do not fully trust. This is the case, for example, when users outsource their storage.

In this dissertation we explore various security problems in settings where the honest parties do not fully trust the resources available to them. In particular, we focus on fundamental computational resources: namely, randomness and storage.

First, we consider the problem of privately encrypting data using malicious randomness. We introduce and formally define two notions of security for private-key

encryption that guarantee that if a message is encrypted using a truly random key and a truly random source of randomness, the message will be protected even against an adversary that will control the source in the future, and one that controlled it in the past. We also provide several efficient constructions that meet these new notions of security.

Second, we consider the problem of authenticating data stored on an unreliable server. In particular, we show how to compile any 3-round public-coin zero-knowledge proof of knowledge with certain homomorphic properties into an efficient proof of data possession system. Our compiler yields PDP systems with $O(1)$-size proofs, and we describe a simple protocol that, when compiled, generates a privately verifiable PDP system that also has $O(1)$ storage complexity at the verifier.

Finally, we consider the problem of storing private data on an untrusted server. Specifically, we examine the problem of symmetric searchable encryption and observe that the current definitions only achieve what we call non-adaptive security, while the more natural usage of searchable encryption requires adaptive security. To address this, we propose two new definitions for SSE: one non-adaptive and one adaptive. We also present efficient constructions for both security notions based on any one-way function. Our first construction is the most efficient non-adaptive SSE scheme to date in terms of computation on the server, and incurs minimal cost for the user. Our second construction achieves adaptive security, which was not previously achieved by any constant-round solution. Finally, we extend the problem of SSE to the multi-user

setting, where a client wishes to allow an authorized group of users to search through its document collection. For the case of multi-user SSE we introduce an efficient transformation based on pseudo-random permutations and broadcast encryption that turns any single-user scheme into a multi-user scheme.

Bibliography

[1] Health Insurance Portability and Accountability Act. http://www.hhs.gov/ocr/hipaa, 1996.

[2] Gramm-Leach-Bliley Act. http://www.ftc.gov/privacy/privacyinitiatives/glbact.html, 1999.

[3] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. M. Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In V. Shoup, editor, *Advances in Cryptology – CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. Syverson, editors, *ACM Conference on Computer and Communication Security (CCS '07)*. ACM Press, 2007.

[5] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword

searches over encrypted data. In S. Qing, W. Mao, J. Lopez, and G. Wang, editors, *Seventh International Conference on Information and Communication Security (ICICS '05)*, volume 3783 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2005.

[6] B. Barak and S. Halevi. A model and architecture for pseudo-random generation and applications to /dev/random. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security (CCS '05)*. ACM, 2005.

[7] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, Lecture Notes in Computer Science, pages 535–552. Springer, 2007.

[8] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Symposium on Foundations of Computer Science (FOCS '97)*, pages 394–405. IEEE Computer Society, 1997.

[9] M. Bellare and O. Goldreich. On defining proofs of knowledge. In E. Brickell, editor, *Advances in Cryptology – CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer-Verlag, 1992.

[10] M. Bellare, J. Killian, and P. Rogaway. The security of cipher block chaining. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer, 1994.

[11] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology – ASIACRYPT '00*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[12] M. Bellare and B. Yee. Forward-security in private-key cryptography. In M. Joye, editor, *The Cryptographers' Track at the RSA Conference (CT-RSA '03)*, volume 2612 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2003.

[13] S. Bellovin and W. Cheswick. Privacy-enhanced searches using encrypted Bloom filters. Technical Report 2004/022, IACR ePrint Cryptography Archive, 2004.

[14] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *ACM Symposium on the Theory of Computation (STOC '88)*, pages 1–10. ACM, 1988.

[15] J. Black and P. Rogaway. Ciphers with arbitrary finite domains. In B. Preneel, editor, *The Cryptographers' Track at the RSA Conference (CT-RSA '02)*, volume 2271 of *Lecture Notes in Computer Science*, pages 114–130. Springer-Verlag, 2002.

[16] M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *IEEE Symposium on Foundations of Computer Science (FOCS '91)*, pages 90–99. IEEE Computer Society, 1991.

[17] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

[18] D. Boneh, G. di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[19] D. Boneh, E. Kushilevitz, R. Ostrovsky, and W. Skeith. Public-key encryption that allows PIR queries. In A. Menezes, editor, *Advances in Cryptology – CRYPTO '07*, volume 4622 of *Lecture Notes in Computer Science*, pages 50–67. Springer, 2007.

[20] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT '01*, volume 2248 of *Lecture Notes in Computer Science*, pages 514–32. Springer, 2001.

[21] C. Bosley and Y. Dodis. Does privacy require true randomness? In S. Vadhan, editor, *Theory of Cryptography Conference (TCC '07)*, volume 4392 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2007.

[22] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In J. Ioannidis, A. Keromytis, and M. Yung, editors, *Applied Cryptography and Network Security (ACNS '05)*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005.

[23] R. Cramer, I. Damgrard, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *Advances in Cryptology – CRYPTO '94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 1994.

[24] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In A. Juels, R. Wright, and S. D. C. di Vimercati, editors, *ACM Conference on Computer and Communications Security (CCS '06)*, pages 79–88. ACM, 2006.

[25] Y. Deswarte, J.-J. Quisquater, and A. Saidane. Remote integrity checking. In M. Gertz, editor, *Integrity and Internal Control in Information Systems (IICIS' 03)*, 2003.

[26] Y. Dodis, S. J. Ong, M. Prabhakaran, and A. Sahai. On the (im)possibility of cryptography with imperfect randomness. In *IEEE Symposium on Foundations of Computer Science (FOCS '04)*, pages 196–205. IEEE Computer Society, 2004.

[27] Y. Dodis and J. Spencer. On the (non)universality of the one-time pad. In *IEEE Symposium on Foundations of Computer Science (FOCS '02)*, pages 376–. IEEE Computer Society, 2002.

[28] U. Feige, A. Fiat, and A. Shamir. Zero knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, 1988.

[29] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *ACM Symposium on Theory of Computing (STOC '90)*, pages 416–426. ACM, 1990.

[30] A. Fiat and M. Naor. Broadcast encryption. In D. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[31] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[32] D. Filho and P. Baretto. Demonstrating data possession and uncheatable data transfer. Technical Report 2006/150, IACR ePrint Cryptography Archive, 2006.

[33] M. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *Journal of the ACM*, 31(3):538–544, 1984.

[34] K. Fu, S. Kamara, and T. Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In *Network and Distributed System Security Symposium (NDSS 2006)*. The Internet Society, 2006.

[35] E.-J. Goh. Secure indexes. Technical Report 2003/216, IACR ePrint Cryptography Archive, 2003. See http://eprint.iacr.org/2003/216.

[36] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *IEEE Symposium on the Foundations of Computer Science (FOCS '84)*, pages 464–479. IEEE Computer Society, 1984.

[37] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *ACM Symposium on the Theory of Computation (STOC '87)*, pages 218–229. ACM, 1987.

[38] O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In A. Odlyzko, editor, *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 171–185. Springer, 1987.

[39] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

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

[41] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *ACM Symposium on Theory of Computing (STOC '85)*, pages 291–304. ACM, 1985.

[42] P. Golle, J. Staddon, and B. Waters. Secure conjunctive keyword search over encrypted data. In M. Jakobsson, M. Yung, and J. Zhou, editors, *Applied Cryp-*

tography and Network Security Conference (ACNS '04), volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004.

[43] L. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory. In C. Günther, editor, *Advances in Cryptology – EUROCRYPT '88*, volume 330 of *Lecture Notes in Computer Science*, pages 123–128. Springer, 1988.

[44] J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[45] A. Juels and B. Kaliski. PORs: Proofs of retrievability for large files. In P. Ning, S. D. C. di Vimercati, and P. Syverson, editors, *ACM Conference on Computer and Communication Security (CCS '07)*. ACM, 2007.

[46] S. Kamara and J. Katz. How to encrypt with a malicious random number generator. In K. Nyberg and S. Vaudenay, editors, *Fast Software Encryption (FSE '08)*, Lecture Notes in Computer Science. Springer, 2008. To Appear.

[47] J. Katz and M. Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In B. Schneier, editor, *Fast Software Encryption (FSE '00)*, volume 1978 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2000.

[48] M. Krohn, M. Freedman, and D. Mazieres. On-the-fly verification of rateless

erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy*, pages 226–240. IEEE Computer Society, 2004.

[49] E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *IEEE Symposium on Foundations of Computer Science (FOCS '97)*, pages 364–373. IEEE Computer Society, 1997.

[50] J. McInnes and B. Pinkas. On the impossibility of private key cryptography with weakly random keys. In A. Menezes and S. Vanstone, editors, *Advances in Cryptology – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 421–435. Springer, 1990.

[51] M. Naor and G. Rothblum. The complexity of online memory checking. In *IEEE Symposium on Foundations of Computer Science (FOCS '05)*, pages 573–584. IEEE Computer Society, 2005.

[52] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. In *ACM Symposium on Theory of Computing (STOC '90)*, pages 427–437. ACM, 1990.

[53] R. Ostrovsky. Efficient computation on oblivious RAMs. In *ACM Symposium on Theory of Computing (STOC '90)*, pages 514–523. ACM, 1990.

[54] R. Ostrovsky and W. Skeith. Private searching on streaming data. In V. Shoup,

editor, *Advances in Cryptology – CRYPTO '05*, volume 3621 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2005.

[55] D. Park, K. Kim, and P. Lee. Public key encryption with conjunctive field keyword search. In C. H. Lim and M. Yung, editors, *Workshop on Information Security Applications (WISA '04)*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004.

[56] C. Rackoff and D. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576, pages 433–444. Springer, 1991. Lecture Notes in Computer Science.

[57] O. Rogaway. Nonce-based symmetric encryption. In B. Roy and W. Meier, editors, *Fast Software Encryption (FSE '04)*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004.

[58] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[59] T. Schwarz and E. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. *IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, page 12, 2006.

[60] F. Sebe, A. Martinez-Balleste, Y. Deswarte, J. Domingo-Ferrer, and J.-J.

Quisquater. Time-bounded remote file integrity checking. Technical Report 04429, LAAS, 2004.

[61] H. Shacham and B. Waters. Compact proofs of retrievability. Cryptology ePrint Archive, Report 2008/073, 2008.

[62] D. Song, D. Wagner, and A. Perrig. Practical techniques for searching on encrypted data. In *IEEE Symposium on Research in Security and Privacy*, pages 44–55. IEEE Computer Society, 2000.

[63] M. Tompa and H. Woll. Random self-reducibility and zero knowledge interactive proofs of possession of information. In *IEEE Symposium on Foundations of Computer Science (FOCS '87)*, pages 472–482. IEEE Computer Society, 1987.

[64] G. Yamamoto, S. Oda, and K. Aoki. Fast integrity for large data. In *Software Performance Enhancement for Encryption and Decryption (SPEED '07)*, 2007.

[65] A. Yao. Protocols for secure computations. In *IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164. IEEE Computer Society, 1982.

Curriculum Vitae

Seny Kamara was born on April 25^{th}, 1978 in Noisy-le-Sec, France. He earned a B.Sc. in Computer Science from Purdue University in December 2001, and a M.S.E. in Computer Science from the Johns Hopkins University in 2006. He was the recipient of a Bell Labs Graduate Fellowship, a Phillips and Camille Bradford fellowship and was a visiting fellow at the UCLA Institute for Pure and Applied Mathematics during the Fall of 2006.

Seny has published peer-reviewed scientific papers on various topics including cryptography, biometrics and network security.