© 2016 Muhammad Naveed

SECURE AND PRACTICAL COMPUTATION ON ENCRYPTED DATA

BY

MUHAMMAD NAVEED

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Doctoral Committee:

Professor Nikita Borisov
Professor Carl Gunter, Chair
Professor Manoj Prabhakaran, Chair
Professor Elaine Shi, Cornell University
Professor XiaoFeng Wang, Indiana University

Abstract

Because of the importance of computing on data with privacy protections, the cryptographic community has developed both theoretical and practical solutions to compute on encrypted data. On the one hand, theoretical schemes, such as fully homomorphic encryption and functional encryption, are secure but extremely inefficient. On the other hand, practical schemes, such as property-preserving encryption, gain efficiency by accepting significant reductions in security. In this thesis, we first study the security of popular property-preserving encryption schemes that are being used by companies such as Microsoft and Google. We show that such schemes are unacceptably insecure for key target applications such as electronic medical records. Second, we propose new models to compute on encrypted data and develop efficient constructions and systems. We propose a new cryptographic primitive called Blind Storage and show how it can be used to realize symmetric searchable encryption, which is much more secure than property-preserving encryption. Finally, we propose a new cryptographic model called Controlled Functional Encryption and develop two efficient schemes in this model.

To Abu, Ami, Iqra, and Zoha.

Acknowledgments

I am extremely grateful to my advisors Professor Carl Gunter and Professor Manoj Prabhakaran. I have been very fortunate to have two advisors working in different areas, which made my PhD experience amazingly unique and exciting. They were very supportive of my collaborations with external researchers, allowed me to visit different universities and industry labs, and continued to work with me remotely when I was away.

I would like to express my deepest gratitude to Professor XiaoFeng Wang from the Indiana University. I collaborated extensively with him throughout my PhD work, and we published many papers together. He taught me how to conduct systems security research and the art of publishing in the top-tier security conferences.

I would like to offer my special thanks to Professor Elaine Shi from the Cornell University. She invited me to spend the last year of my PhD working with her at the Cornell University. She has been very supportive throughout my stay at Cornell.

I am very grateful to Prof. Jean-Pierre Hubaux from the École polytechnique fédérale de Lausanne (EPFL). I spent one semester working with him and his group at EPFL. I would also like to thank Prof. Erman Ayday, whom I collaborated extensively during my EPFL visit. Prof. Hubaux, his group, and EPFL staff made my stay in Lausanne very productive and enjoyable.

A special thanks go to Prof. Seny Kamara from the Brown University. I interned with him when he was at Microsoft Research Redmond. My work with him proved to be very impactful. He also gave helpful suggestions and encouragement on my other work.

I am extremely grateful to all my lab mates at the Illinois Security Lab: Vincent Bindschaedler, Ji Young Chun, Soteris Demetriou, Eric Duffy, Siddharth Gupta, Dongjing He, Avesta Hojjati, Fariba Khan, Gaurav Lahoti, Michael LeMay, Yunhui Long, Xun Lu, Whitney Merrill, Tony Michalos, Se Eun Oh, Ravinder Shankesi, Igor Svecs, Güliz Seray Tuncay, Qi Wang, Ting Wu, Wei Yang, and Aston Zhang.

They made my stay at Illinois very enjoyable.

I was extremely lucky to work with several outstanding researchers during my PhD. I would like to thank all of my collaborators: Fardin Abdi, Shashank Agrawal, Erman Ayday, Vincent C. Bindschaedler, Gabriela Ciocarlie, Ellen W. Clayton, Soteris Demetriou, David Evans, Jacques Fellay, Chris Fletcher, Ashish Gehani, Paul Grubbs, Carl Gunter, Dongjing He, Yan Huang, Jean-Pierre Hubaux, Seny Kamara, Yeonjoon Lee, Tongxin Li, Bradley A. Malin, Whitney Merrill, Klara Nahrstedt, Xiaorui Pan, Manoj Prabhakaran, Mariana Raykova, Ling Ren, Kevin Sekniqi, Elaine Shi, Igor Svecs, XiaoFeng Wang, Charles V. Wright, Luyi Xing, Nan Zhang, and Xiaoyong Zhou.

I would like to thank all the staff at CS@Illinois, especially Indria Clay, Mary Beth Kelley, Colin Robertson, and Andrea Whitesell, for their help.

A special thanks go to Google, Sohaib Abbasi and his wife Sara Abbasi. My research was partially supported by the Sohaib and Sara Abbasi fellowship and the Google fellowship. I am also grateful to the Office of the National Coordinator for Health IT in the Department of Health and Human Services and the National Science Foundation; my research was partially supported by the grants from these agencies.

I am extremely grateful to my parents for supporting me through all these years. I could not thank them enough for everything they have done for me. I am also very grateful to my sisters, Aaisha and Faiza, and my brother, Hassan, for their love and support.

I would like to thank my lovely wife, Iqra, from the bottom of my heart; it simply would not be possible without her love and support. A very special thanks go to my little angel, Zoha.

Chapters 3, 4, 5 of this thesis are directly taken from my papers [1, 2, 3].

Table of Contents

Chapter 1 Introduction 1

Chapter 2 Background 8

Chapter 3 A Critical Analysis of Property-Preserving Encryption 17
- 3.1 Threat Model 22
- 3.2 Attacking DTE Columns 24
- 3.3 Attacking OPE Columns 27
- 3.4 Simulating a Medical EDB 30
- 3.5 Experimental Setup 34
- 3.6 Experimental Results 36

Chapter 4 A Practical Model for Searching on Encrypted Data 41
- 4.1 Overview 43
- 4.2 Blind Storage 46
- 4.3 Searchable Symmetric Encryption 63
- 4.4 Implementation Details 70
- 4.5 Searchable Encryption Evaluation 71
- 4.6 Efficacy of Oblivious RAM in Searchable Encryption 79

Chapter 5 A Practical Model for Computing on Encrypted Data 97
- 5.1 Overview 99
- 5.2 Controlled Functional Encryption 105
- 5.3 Constructions 108
- 5.4 Alternate General Construction 115
- 5.5 Implementation and Evaluation 116
- 5.6 Applications 122

Chapter 6 Conclusion 125

References 129

Chapter 1

Introduction

Cryptography has been used to secure communication for millennia. Historically, it was an art to construct secure encryption schemes; however, this approach failed miserably, sooner or later all such encryption schemes were broken. Only in the last few decades scientific foundations for cryptography have been developed and thankfully now we know how to develop cryptography with mathematical proofs of security. While difficulties remain, we know how to secure our data at rest and in flight both in theory and practice. However, securing data at rest and in flight does not necessarily protect it from ill-intentioned people.

The encrypted data need to be used at some point and the straightforward approach of decrypting the data for computation is susceptible to threats such as data breaches, insider threats, and cyber espionage. A promising approach to prevent such threats is to enable computation on encrypted data such that only the results from the computation are revealed to the authorized user.

Computation on Encrypted Data

Enabling computation on encrypted data is the holy grail of modern cryptography. The sole purpose of encrypting the data is to make it unintelligible, and therefore, computation on encrypted data seems counter-intuitive. Nonetheless, we know how to compute on encrypted. In fact, we know many different ways to compute on encrypted data, and it has been an active area of research for several decades. Some of the cryptographic primitives that allow computation on encrypted data are described below:

- **Property-Preserving Encryption** preserves some property of the plaintext in the ciphertext that allows limited computation on the encrypted data. For example, an equality-preserving encryption scheme [4], also known as

deterministic encryption, preserves the equality in the ciphertexts and allows lookup queries. An order-preserving encryption scheme [5, 6] preserves the order in the ciphertexts and allows order and range queries. Property-preserving encryption schemes are very popular in industry, because they do not require changes in the applications and database software, supports a large class of queries, are efficient, and provide some level of security. In Chapter 3, we show that property-preserving encryption is unacceptably insecure and in some cases not much better than not encrypting at all.

- **Symmetric Searchable Encryption** schemes allow efficient searching on encrypted data. The state-of-the-art schemes are efficient and expressive supporting Boolean [7] and range queries [8]. However, the efficiency comes at the cost of some leakage. Islam, et al., [9] have shown that such leakage can reveal search queries if the adversary knows plaintext for all the encrypted data, a subset of the queries, and the access pattern. Since in most of the interesting applications it is unrealistic to assume that the adversary would know all the plaintext, the attack is not realistic in such scenarios. However, searchable symmetric encryption leaks significant information and it is best to develop techniques with minimum leakage. In Chapter 4, we develop a novel symmetric searchable encryption scheme which is the first scheme that is compatible with commercial cloud services. We also show that preventing information leakage in symmetric searchable is non-trivial and would require developing novel solutions.

- **Homomorphic Encryption** allows a party to compute an encrypted result on encrypted data such that the result can only be decrypted by the client holding the secret key. Gentry developed the first fully homomorphic encryption schemes which is capable of computing any arbitrary function on the encrypted [10]. There has been significant development in this area and efficiency has been improved by many orders of magnitude; however, fully homomorphic encryption is still 5 orders of magnitude slower than the plaintext computation.

- **Functional Encryption** allows a party to compute a *plaintext* result on encrypted data [11]. Note the distinction from fully homomorphic encryption, which allows a party to compute *encrypted* result that can only be decrypted

by the party holding the key. Functional Encryption is an active area of research and there are many interesting results, however, it is still very far from being practical. In Chapter 5, we propose a new model, called Controlled Functional Encryption, which allows for the development of very efficient schemes.

- **Secure Computation** enables two or more parties to compute any function of their inputs without revealing anything but the output of the computation and whatever can be learned from output itself. There has been a lot of work on secure computation in the last three decades starting from the ground-breaking work on Yao [12]. At this point in time, two-party computation is mature and could be used in applications with relatively small data. While there are many interesting theoretical multi-party computation protocols, there is still a lot of work needed to make it practical.

The cryptographic community has approached the problem of computation on encrypted data from both theoretical and practical perspectives. On the one hand, we have schemes, such as homomorphic encryption, that are secure but extremely inefficient. On the other, practical schemes such as, property-preserving encryption, are practical and widely deployed but leak significant information.

In this thesis, we first study the security of deployed encryption schemes, namely deterministic encryption and order-preserving encryption that allow computation on encrypted data. We believe that such cryptanalysis is crucial for the understanding of the systems people use and developing secure schemes and systems. Second, we develop the first symmetric searchable encryption scheme that is compatible with cloud storage services such as Dropbox. We also discuss that using techniques such as Oblivious-RAM makes searchable encryption very inefficient. Third, we develop a novel cryptographic model called, Controlled Functional Encryption, that allows construction of very efficient schemes to allow an authorized party to directly compute on encrypted data.

A Critical Analysis of Property-Preserving Encryption

Over the past several decades cryptographers have developed many techniques to compute on encrypted data, however, they are still far from being practically efficient. These schemes aim for high levels of security at the cost of efficiency.

Therefore, to design *efficient* schemes to compute on encrypted data an approach that is less secure but more efficient has been explored over the past several years. This approach does not lead to perfectly secure schemes and deliberately leaks information to gain efficiency. The most popular cryptographic primitives developed in this fashion are equality-preserving encryption, also known as deterministic encryption, and order-preserving encryption. The equality-preserving encryption and order-preserving encryption are instances of a more general class of encryption schemes called property-preserving encryption. The equality-preserving and order-preserving encryption are very useful to query SQL databases. The equality-preserving encryption enables look-up queries on encrypted data, e.g., looking up people with their first names such as "John". Any such encryption scheme would always encrypt the same message to the same ciphertext and as a result look up queries can be easily performed without decrypting the data. Order-preserving encryption enables sorting (e.g., sorting a list of people by first names) and range-queries (e.g., querying people between age 10 and 40).

Traditionally, encryption schemes are considered secure only if they do not preserve any information in the ciphertext except the size of the message. However, to design efficient schemes to enable lookup and order queries, property-preserving encryption schemes inherently leak information. While it is well-understood that such schemes leak information, it is not clear what such leakage mean for real applications such as electronic medical records. We address this question in Chapter 3.

The question now arises, **why should we study the security of property- preserving encryption**. First, such schemes are very popular in industry and are used by companies such as Microsoft, Google, SAP, Skyhigh Networks, Ciphercloud, and many others. In fact, Microsoft SQL Server 2016 will be shipping with such a system called "Always Encrypted", which is advertised as one of the flagship benefits[1]. The reason behind the property-preserving encryption's popularity is that the systems that use it promise *(i)* security, *(ii)* no change to applications or database servers, *(iii)* minimal performance overhead, and *(iv)* support for a large class of SQL queries. Second, there is a push towards using such encryption schemes for sensitive data such as electronic medical records and financial data. It is crucial to understand the implications of using such encryption schemes in such applications before deploying them to protect real people's data.

[1]For example, see here or here (video demo: starts at 18:32).

5for such applications.

We analyzed encrypted database systems using property-preserving encryption using electronic medical records (EMR) as a concrete application. Electronic medical records is a real application where security and privacy are critical, is representative of many other applications, and is a prime market for such systems. We used well known and novel techniques to design inference attacks against these systems using real data from 200 U.S. hospitals. An inference attack uses ciphertexts along with auxiliary information to recover plaintexts. Our attacks use real auxiliary data publicly available over the Internet. In short, our attack framework is the least any attacker can do and uses the weakest threat model such systems are designed for. Our attacks demonstrate that an alarming amount of information is revealed from these systems: *one attack recovered more than* 80% *of patient records for all* 200 *hospitals*. This shows that property-preserving encryption and systems using it are not secure enough for applications, such as electronic medical records.

Our attacks are just the tip of the iceberg because we are assuming a very weak adversary: one that only have access to the encrypted data. However, in reality adversary can be much more powerful and could use active attacks and have access to the query patterns. Moreover, we assume ideal leakage for equality-preserving and order-preserving encryption, but practical schemes leak significantly more information which can be used to learn even more information.

A Practical Model for Searching on Encrypted Data

As mentioned above and explained in detail in Chapter 3, property-preserving encryption is far from being secure for real applications even against the weakest and most common threat where the adversary only have access to the encrypted data. A different cryptographic primitive called Symmetric Searchable Encryption, which is increasingly getting attention from the industry, is completely secure against this common threat. While Symmetric Searchable Encryption does have leakage, this leakage is due to queries and unless adversary has query information, it cannot learn useful information. Nonetheless, the leakage is significant and can lead to successful attacks. Symmetric Searchable Encryption is also useful for outsourcing data to cloud. Cloud storage is an economical, reliable, and universally accessible alternative to local storage. However, the lack of confidentiality from

the cloud provider prevents its widespread adoption. Standard encryption provides confidentiality but renders data unsearchable. There has been a lot of work on Symmetric Searchable Encryption (SSE) that enables efficient search on encrypted data. The existing schemes are incompatible with cloud storage APIs. All prior schemes required both cloud storage and compute services, which increases cost, attack surface, and response time. More importantly, due to high latency and monetary cost of outbound data transfer, it limits customers to cloud providers offering both storage and compute services, such as Amazon, and excludes storage-only services, such as Dropbox. We developed the first SSE system that is compatible with cloud storage APIs [2]. We designed a novel secure storage primitive, called Blind Storage, and used it as a black box to construct an SSE scheme. Blind Storage is designed to protect information against the cloud service providers and adversaries that steals the data. The system is very efficient, based on standard cryptographic primitives (SHA256 and AES), secure in the standard model, and simple to implement. Despite all the computation is done by the client, and both client and server do computation in other schemes, Our SSE scheme is one of the most efficient SSE schemes for single keyword queries; it has less than 10% overhead over plaintext search. It supports additions and deletions of documents. The system can also be used to protect local storage infrastructure to avoid data breaches. We built Blind Storage system and SSE on top of it in C++ and the code is open source.

A Practical Model for Computing on Encrypted Data

Many practical applications require more than just searching on the encrypted data. We may want to determine the susceptibility of an individual to a particular disease using his genomic data, find out similarity between genomes of two individuals, or compute any arbitrary function of the encrypted data. We developed a new model called Controlled Functional Encryption, that allows the construction of practically efficient schemes to allow computation on the encrypted data. We built a system that enables patients to securely outsource their genomic data to hospitals so that medical professionals can only compute personalized medicine tests allowed by the patients and hackers who break in learn nothing. A personalized medicine test determines the disease risk of an individual by computing the inner-product of genome variants and disease marker vectors. Along with a physician and

genomicist, we formulated requirements for a practical solution. After studying existing approaches, we realized that new tools are required for the solution. We describe our new cryptographic model called Controlled Functional Encryption (CFE) in Chapter 5, which is a relaxation of Functional Encryption (FE). Both CFE and FE are cryptographic models for computing on encrypted data. In FE, the party computing the function uses the same key to compute a function on multiple ciphertexts; in CFE, a fresh key is required for each new ciphertext. For many real applications, this change in model is acceptable and enables development of efficient schemes based on standard cryptographic assumptions, several magnitudes faster than state-of-the-art FE schemes. We developed a scheme that computes the inner-product of two vectors by cleverly using public key encryption and additive secret-sharing. We also developed a scheme for arbitrary functions based on a careful combination of public key encryption and Yao's garbled circuits. Using our Java implementation (open source), a laptop computes personalized medicine tests on full scale human genome data *in less than a second*. CFE is not limited to genomic applications and can be used in many other applications.

Chapter 2

Background

In this chapter, we provide some background information necessary to understand the material in the rest of the chapters.

Relational databases. A relational database is a collection of tables where each row corresponds to an entity (e.g., a customer or an employee) and each column corresponds to an attribute (e.g., age, height, salary). For any given attribute, we refer to the set of all possible values that it can take as its *space*. The *attribute space* of a column is the space of that column's attribute. If a column supports equality or range queries, then we refer to it as an equality or range column. The structured query language (SQL) is a special-purpose language for querying relational databases.

Datasets. A dataset $\mathbf{d} = (d_1, \ldots, d_n)$ is a sequence of elements from a universe $\mathbb{D}$. We assume, without loss of generality, that every space $\mathbb{D}$ is totally ordered. We view the *histogram* of a dataset $\mathbf{d}$ as a $|\mathbb{D}|$-dimensional vector over $\mathbb{N}_{\geq 0}$ with, at position i, the number of times the ith element of $\mathbb{D}$ appears in $\mathbf{d}$. We denote by $\mathsf{Hist}(\mathbf{d})$ the operation that computes the histogram of a dataset $\mathbf{d}$. The *cumulative distribution function* (CDF) of a dataset $\mathbf{d}$ is a $|\mathbb{D}|$-dimensional vector over $\mathbb{N}_{\geq 0}$ with, at position i, the number of times the first through ith elements of $\mathbb{D}$ that appear in $\mathbf{d}$. We denote by $\mathsf{CDF}(\mathbf{d})$ the operation that computes the CDF of a dataset $\mathbf{d}$. The CDF of $\mathbf{d}$ is the vector $\mathbf{f}$ such that for all $i \in [n]$, $f_i = \sum_{j=1}^{i} h_j$, where $\mathbf{h} = (h_1, \ldots, h_n)$ is the histogram of $\mathbf{d}$.

We denote by $\mathsf{Unique}(\mathbf{d})$ the dataset that results from removing all duplicates in $\mathbf{d}$ (i.e., from keeping only the first occurrence of every element in $\mathbf{d}$). The rank of an element $d \in \mathbb{D}$ in a dataset $\mathbf{d}$ is the position of its first occurrence in $\mathbf{d}$ if $d \in \mathbf{d}$ and 0 if $d \notin \mathbf{d}$. We denote the rank of d in $\mathbf{d}$ by $\mathsf{Rank}_{\mathbf{d}}(d)$.

We will often need to sort datasets. The result of sorting $\mathbf{d}$ by value is the sequence $\mathbf{d}' = (d_{i_1}, \ldots, d_{i_n})$ such that $d_{i_1} < \cdots < d_{i_n}$. We denote this operation $\mathbf{d}' \leftarrow \mathsf{vSort}(\mathbf{d})$. When dealing with the histogram $\mathbf{h}$ of a dataset $\mathbf{d}$ over $\mathbb{D}$, we

identify the coordinates of $\mathbf{h}$ with the elements of $\mathbb{D}$.

Encryption. A symmetric encryption scheme $\mathsf{SKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a tuple of three algorithms that work as follows. Gen takes as security parameter as input and returns a secret key K; Enc takes as input a key K and a message m and returns a ciphertext c; and Dec takes as input a key K and a ciphertext c and returns a message m. The standard notion of security for encryption is security against chosen-plaintext attacks (CPA). We refer the reader to [13] for a detailed description of this notion. Here, we only mention that it is well-known that for symmetric-key encryption, CPA-security can only be achieved if Enc is either stateful or randomized.

IND-CCA2 Secure Encryption. IND-CCA2 encryption provides security against adaptive chosen-ciphertext attacks. It has the following useful non-malleability property: given an encryption of a message m, it is infeasible for a computationally bounded adversary to create an encryption of a message related to m. The security requirement is formally captured via an indistinguishability based security *game* which we briefly describe here. An adversary outputs two messages m_0 and m_1 and is given a ciphertext c – an encryption of either m_0 or m_1. Even with access to a decryption oracle which can be used to decrypt any ciphertext except c itself, adversary should not be able to tell which message c corresponds to. Very efficient IND-CCA2 encryption schemes are known in the random oracle model, for instance RSA-OAEP [14]. A detailed discussion of the relationship among different notions of security for public-key encryption can be found in [15].

Deterministic encryption. A symmetric DTE scheme $\mathsf{DTE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme for which Enc is not randomized; that is, each message m is mapped by Enc to a *single* ciphertext under a key K.

Order-preserving encryption. A symmetric OPE scheme $\mathsf{OPE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme with the following property: if $m_1 > m_2$ then $\mathsf{Enc}_K(m_1) > \mathsf{Enc}_K(m_2)$; if $m_1 = m_2$ then $\mathsf{Enc}_K(m_1) = \mathsf{Enc}_K(m_2)$; and if $m_1 < m_2$ then $\mathsf{Enc}_K(m_1) < \mathsf{Enc}_K(m_2)$.

Additively homomorphic encryption. A symmetric additively homomorphic encryption (AHE) scheme $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a symmetric encryption scheme with the added property that: $\mathsf{Dec}_K\big(\mathsf{Enc}_K(m_1) \otimes \mathsf{Enc}_K(m_2)\big) = m_1 + m_2$, where $\otimes$ is an operation over the ciphertext space of AHE and not necessarily addition.

Join encryption. The CryptDB system supports two kinds of Join operations: equi-joins and range-joins. Equi-joins are supported using a scheme $\mathsf{EJOIN} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ which is a combination of DTE and hashing. Range-joins are supported using an encryption scheme $\mathsf{RJOIN} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ based on OPE. We note that after a join query (of either kind), two joined columns are left encrypted under the same key.

Searchable encryption. The systems also make use of searchable encryption scheme $\mathsf{SRCH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Token}, \mathsf{Dec})$ for keyword search operations. In CryptDB, this is instantiated with a variant of the scheme of the scheme of Song, Wagner and Perrig [16].

Onion encryption. Popa *et al.*use the term *onion* to refer to the composition of encryption schemes. For example, given two encryption schemes $\mathsf{SKE}^1 = (\mathsf{Gen}^1, \mathsf{Enc}^1, \mathsf{Dec}^1)$ and $\mathsf{SKE}^2 = (\mathsf{Gen}^2, \mathsf{Enc}^2, \mathsf{Dec}^2)$ the $\mathsf{SKE}^1 \circ \mathsf{SKE}^2$ encryption of a message m is defined as

$$\mathrm{ct} = \mathsf{Enc}^1_{K_1}\left(\mathsf{Enc}^2_{K_2}(m)\right).$$

Encrypted Database Systems Using Property-Preserving Encryption We recall the high-level architecture of the CryptDB system. The system is composed of three entities: an application App, a proxy Prx, and a server Srv. The application and proxy are trusted while the server is untrusted. To create an encrypted database EDB from a database DB, the proxy generates a master secret key msk and uses it to encrypt each table as follows. First, an anonymized schema is created where the attributes for each column are replaced with random labels. The mapping between the attributes and their labels is stored at the proxy. Then, each cell is encrypted using four different *onions*. More specifically, the following four onions are used

- *Equality onion*: encrypts a string s as

$$\mathrm{ct} = \mathsf{Enc}^{\mathsf{SKE}}_{K_S}\left(\mathsf{Enc}^{\mathsf{DTE}}_{K_D}\left(\mathsf{Enc}^{\mathsf{EJOIN}}_{K_J}(s)\right)\right);$$

- *Order onion*: encrypts a string s as

$$\mathrm{ct} = \mathsf{Enc}^{\mathsf{SKE}}_{K_S}\left(\mathsf{Enc}^{\mathsf{OPE}}_{K_O}\left(\mathsf{Enc}^{\mathsf{RJOIN}}_{K_{OJ}}(s)\right)\right);$$

- *Search onion*: encrypts a keyword w as

$$\text{ct} = \mathsf{Enc}^{\mathsf{SRCH}}_{K_S}(w);$$

- *Add onion*: encrypts an integer i as

$$\text{ct} = \mathsf{Enc}^{\mathsf{AHE}}_{K_A}(i);$$

To support queries on encrypted data, the encrypted cells in the EDB are decrypted down to a certain layer. This process is referred to as *peeling* in [17] and every cell in a given column is peeled to the same level. The proxy keeps track of the layer at which each column is peeled.

To query an encrypted database the application issues a SQL query that is rewritten by the proxy before being sent to the server. In the new query, each column name is replaced with its random label and each constant is replaced with a ciphertext determined as a function of the semantics of the query. More precisely, for each type of operation the proxy does the following:

- *equality*: v is replaced with $\text{ct} = \mathsf{DTE.Enc}_K(v)$;
- *range*: v is replaced with $\text{ct} = \mathsf{OPE.Enc}_K(v)$;
- *search*: v is replaced with $\text{tk} = \mathsf{SRCH.Token}_K(v)$;
- *addition*: v is replaced with $\text{ct} = \mathsf{AHE.Enc}_K(v)$;
- *join*: v is replaced with $\text{ct} = \mathsf{EJOIN.Enc}_K(v)$;

After re-writing the query, the proxy checks the onion levels of the relevant columns to determine if they need to be peeled further. If so, it sends the appropriate decryption keys to the server so that it peels the columns down to the appropriate layer. Note that Cipherbase does not use onions and encrypts columns directly with the PPE scheme needed to support the query.

Column labeling. We note that the CryptDB system allows users to label attributes so that they are only encrypted using either CCA2- or CPA-secure encryption schemes, i.e., they are never "peeled" down to OPE or DTE. In such a case, however, the system cannot support order or equality operations on these attributes unless the columns are downloaded to the client so that they can be executed locally.

Since, we are interested in the leakage that occurs when running EMR applications on top of an *outsourced* EDB system, it follows that the attributes we analyze (see Section 3.4) cannot be labeled.

Oblivious transfer. Oblivious transfer (OT) is one of the most widely studied fundamental primitives in cryptography. It is a two party protocol between a sender, who has two strings x_0 and x_1, and a chooser with choice bit b. While sender does not learn anything about the bit b (chooser's security) in the protocol, chooser only learns the value of x_b (sender's security).

In the common random string (CRS) model, Piekert et al. [18] give an efficient, one-round (first message from chooser to sender, next one from sender to chooser), UC-secure protocol for OT under the well-studied DDH assumption (as well as a variety of other hardness assumptions). We use this protocol, denoted by Π_{OT}, in our general construction in Section 5.3.2.

Garbled Circuits. The main component of our general construction is garbled circuits (GC), introduced by Yao in [12]. See [19] for a recent treatment of this tool. Several implementations of GC already exist [20, 21], and lately, much work has been going into making it more secure and efficient [22, 23, 24, 25, 26, 27].

Related Work

The CryptDB system [17] was the first to support a large fraction of SQL on encrypted data. Other PPE-based systems include Cipherbase [28]. Akin and Sunar [29] describe attacks that enable a malicious database administrator to recover plaintexts from CryptDB through a combination of passive monitoring and active tampering with the EDB.

Frequency analysis was first described by the Arab philosopher and mathematician al-Kindi in the ninth century [30]. Techniques for recovering plaintext encrypted with substitution ciphers using language statistics are well-known (see for example pp.245–250 of [31]). Brekne, Ârnes and Øslebø[32] describe frequency attacks for recovering IP addresses anonymized under a prefix-preserving encryption scheme [33].

Islam *et al.* [9] described the first inference attack against an encrypted search solution. This attack, referred to as the IKK attack, exploits the access pattern leakage of SSE constructions together with auxiliary information about the fre-

quencies of keyword pairs and knowledge of a subset of client queries (in plaintext) to recover information about the remaining queries. In comparison, the attacks we consider here: (1) recover the *database*, as opposed to queries; and (2) require no knowledge of any queries (neither in plaintext nor encrypted). Furthermore, a recent study by Cash, Grubbs, Perry and Ristenpart [34] shows that the accuracy of the IKK attack is so low that it is not usable in practice (unless the adversary already knows most of the underlying data). In contrast, the attacks considered in this thesis are highly-accurate and very efficient. The fact that our attacks are more powerful than the IKK attack is natural since PPE schemes leak considerably more than SSE schemes.

Sanamrad, Braun, Kossman and Venkatesan [35] also consider the security of OPE schemes in the context of encrypted databases. They propose a set of security definitions and discuss previously-known attacks (e.g., frequency analysis and sorting). Unlike standard security definitions, however, the security notions proposed in [35] are attack-specific (e.g., they define security only against frequency analysis) and guarantee only one-wayness; as opposed to standard cryptographic notions which guarantee that *partial* information is protected. Finally, [35] also proposes deterministic and probabilistic OPE variants. The deterministic variant is still vulnerable to our attacks (albeit requiring larger encrypted columns).

There is an extensive literature on OPE variants including probabilistic OPE, modular OPE, etc. [36, 37, 38, 39, 40, 41]. As far as we know, none of these constructions are used in any EDB system.

The problem of searching on encrypted data has received increasing attention from the security and cryptography community, with the growing importance of cloud storage and cloud computation. One of the major hurdles in outsourcing data storage and management for businesses has been security and privacy concerns [42, 43, 44]. Theoretical cryptography literature offers an extremely powerful and highly secure solution in the form of Oblivious Random Access Memory (ORAM) [45, 46], which addresses almost all of the security concerns related to storing data in an untrusted server. However, this solution remains very inefficient for several important applications, despite significant recent improvements [47, 48, 49]. The notion of Symmetric Searchable Encryption (SSE) — investigated in a long line of works including [50, 51, 52, 53, 54, 55, 56, 57, 58, 59], among others — attempts to strike a different balance between efficiency and security, by letting the server learn just the *pattern* of data access (and ideally, nothing more), in return for a simpler and faster construction; further, one often settles for security

against passively corrupt (honest-but-curious) servers. The scheme of [59] also provides a notion of forward privacy, which prevents leaking whether a newly added document contains the keywords the user has already searched for.

The approach in [53] formed the basis for many subsequent works. The basic idea is to use an index that maps each search keyword to the list of documents that contains it. This list is kept as an encrypted linked list, with each node containing the key to decrypt next node. The nodes of all the linked lists are kept together, randomly sorted. Until the head of a linked list is decrypted, it is virtually invisible to the server; in particular, the number of linked lists and their lengths remain hidden from the server. This construction provided non-adaptive security (which assumes that all the search queries are generated at once); efficiently achieving adaptive security has been the subject of much research starting with [53].

An important aspect of SSE is whether it is *dynamic* or not: i.e., whether the client can update the document collection after starting to search on it. Dynamic SSE schemes were presented in [51, 54, 57, 58, 60, 59].

Finally, we mention a few variants of the SSE problem that are *not* considered in this thesis. One could require security against actively corrupt servers, rather than just honest-but-curious servers. Another variant requires more expressive searches, involving multiple keywords (e.g., [61, 7, 60]). One could also require that many clients can perform searches on a document collection created by a single data-owner [62]. While we do not consider these problems in this thesis, the main new tool we build — namely, a Blind Storage system — is a general-purpose tool and is likely to be useful for expressive search queries. Indeed, it could be used to implement components like the "T-sets" of [7] more efficiently.

The Chapter 5 on Controlled Functional Encryption draws inspiration from the notion of functional encryption. Functional encryption is a significantly more challenging problem in comparison with controlled functional encryption, as it does not allow the authority to issue a separate key for each ciphertext being decrypted. However, the solutions for functional encryption are currently highly inefficient [63, 64]. Gorbunov et al. proposed a functional encryption scheme for a bounded number of functions [63] and with ciphertext size depending on the size of the circuit [63]. Goldwasser et al. recently proposed a functional encryption scheme for a bounded number of functions and with succint ciphertexts, i.e., ciphertext size depends upon the depth of the circuit [64]. Solutions using tamper-proof hardware tokens are also inefficient [65].

Sahai et al. proposed single-query functional encryption scheme that supports

any polynomial-time computation on the ciphertext *only once* [66]. Their construction is based on Yao's garbled circuits [12]. In the encryption phase, a garbled circuit is generated with input of the encryption party embedded in the circuit. Each input wire label of this garbled circuit is encrypted using a different public key. The garbled circuit and encryption of the wire labels are sent to the decryption party. Decryption party asks the authority for the decryption keys of the wire labels corresponding to the function to be computed and it decrypts the wire labels using these keys After obtaining the wire labels the decryption party can evaluate the garbled circuit to compute the function. As they need to encrypt each wire label with a different public key, packing is not possible and ciphertext size blows up significantly. Typically, 80 bit wire labels are used in Yao's garbled circuit implementation and encrypting each wire label with public key will blow up the ciphertext size by $\frac{keysize}{80}\times$ the plaintext size. Moreover, the limitation of computing only single function makes the scheme not useful for many practical applications. In fact, the scheme was presented as a public key encryption scheme, where encryptor can encrypt ciphertext without worrying about the credentials of the receiver. The receiver can only decrypt if it has appropriate credentials. The limit of computing single function comes from the reusability issue of Yao's garbled circuit. Feasibility of reusable garbled circuit has been shown but it is very inefficient to be of any practical use [64].

Gorbunov et al. proposed a scheme based on Sahai et al. scheme that supports computation of q-functions over the ciphertext, where q depends on different parameters and increasing q affects the overall efficiency of the scheme [67]. They addresses the reusability issue of garbled circuit in Sahai et al. construction but this makes the scheme very inefficient. Even with overwhelming overhead the scheme supports limited number of functions to be computed. While theoretically, interesting the scheme is very far from being practical and would take hundreds of years even for very small values of q (e.g., 100).

Chung et al. proposed a functional encryption scheme based on stateless hardware tokens that are identical for all users [65]. However, they rely on computationally intensive operations (fully homomorphic encryption schemes) as well as a powerful tamper proof token carrying out significant computation (signature, non-interactive zero knowledge proofs (NIZK) and succinct non-interactive arguments (SNARGs) verification). Moreover, it has been shown that secrets can be easily stolen from tamper-proof hardware [68], so that a single token can be attacked to compromise the entire system. Overall, it is not clear if token based approach

would be practical in various applications.

Chapter 3

A Critical Analysis of Property-Preserving Encryption

As an increasing amount of private data is being collected and stored by corporations and governments, database security has become a critical area in both research and industry. High-profile data breaches like the Anthem breach in which a database of 80 million healthcare records was compromised or the Community Health Systems breach in which 4.5 million HIPAA protected (non-medical) records were stolen have fueled interest in database encryption techniques.

While encryption could offer some protections—particularly when the database is exfiltrated from disk—it also has serious limitations. In particular, since an encrypted database cannot be queried, it has to be decrypted in memory which means the secret key and the database are vulnerable to adversaries with memory access. In cloud settings, where a customer outsources the storage and management of its database, encryption breaks any service offered by the provider.

Encrypted search. Motivated by these limitations of traditional encryption, the area of encrypted search has emerged as one of the most active and potentially impactful areas of cryptography research. Encrypted search is concerned with the design and analysis of cryptographic techniques for searching on encrypted data; including both structured and unstructured data. There are various approaches to search on encrypted data including searchable symmetric encryption (SSE) [16, 69], fully-homomorphic encryption (FHE) [10], oblivious RAMs (ORAM) [70], functional encryption [11], and property-preserving encryption (PPE) [4, 6]. All these approaches achieve different trade-offs between security, query expressiveness, and efficiency.

Leakage and inference attacks. The most secure encrypted search solutions are based on FHE and ORAM but are currently too inefficient to be of practical interest. Therefore, all known practical solutions leak some information. This leakage comes in two forms: setup leakage, which is revealed by the encrypted database (EDB) itself; and query leakage which is revealed from the EDB and the

query protocol.

To better understand the impact of this leakage, an important research direction in encrypted search, initiated by Islam, Kuzu and Kantarcioglu [9], is the design of *inference attacks* which try to recover information about the data or queries by combining leakage with publicly-available information (e.g., census data or language statistics). The most well-known example of an inference attack is frequency analysis which is used to break classical ciphers. Another example is the query-recovery attack of Islam *et al.* against searchable symmetric encryption (SSE) schemes [9].

PPE-based EDBs. In the context of structured data and, in particular, of relational databases, the state-of-the-art encrypted search solutions are based on PPE schemes like deterministic and order-preserving encryption. Roughly speaking, a PPE scheme is an encryption scheme that leaks a certain property of the plaintext. For example, an order-preserving encryption (OPE) scheme encrypts a set of messages in such a way that their ciphertexts reveal the order of the messages (i.e., the *order* property). A deterministic encryption (DTE) scheme encrypts a set of messages in such a way that their ciphertexts reveal whether they are equal or not (i.e., the *equality* property).

The CryptDB system [17] first showed how to use PPE to construct an encrypted database system that supports a subset of SQL. CryptDB can be used without PPE but then comparison and equality operations on columns cannot be outsourced and must to be executed at the client/proxy. In particular, this means the client/proxy has to incur the computation and communication overhead of downloading the entire column, of decrypting it and of querying it locally. In this work, when referring to CryptDB, we implicitly mean the variant where the operations in question are *outsourced* to the database server and *not* executed at the client. The Cipherbase system [28], which uses both DTE and OPE, supports all of SQL. At a very high-level, in Cipherbase, each DB operation can be either done in a secure co-processor or over encrypted data using an approach similar to CryptDB. In this work, when referring to Cipherbase, we implicitly mean the variant where the operations in question are *not* executed in the secure co-processor. Other PPE-based EDB systems include the encrypted BigQuery demo [71] and Always Encrypted [72], both of which uses DTE but not OPE.

PPE-based EDBs have several advantages and have received a lot of interest from Industry. In particular, they are competitive with real-world relational database

systems and they require a minimal number of changes to the standard/legacy database infrastructure. The key to their efficiency and "legacy-friendliness" is the use of PPE which, roughly speaking, allows them to operate on encrypted data in the same way as they would operate on plaintext data. This enables fast operations on encrypted data and the use of standard database algorithms and optimizations.

The use of PPE has important consequences on the security of encrypted database systems. Specifically, since PPE schemes leak a non-trivial amount of information, it is well-known that PPE-based designs like CryptDB and its variants are vulnerable to inference attacks. The extent to which these systems are vulnerable, however, has never been investigated.

In this work, we study concrete inference attacks against EDBs based on the CryptDB design. At a very high-level, these systems encrypt each DB column with layers of different encryption schemes. When queried, the system decrypts the layers until it reaches a layer that supports the necessary operation. In particular, this means that columns that support either range or equality queries are left encrypted with OPE or DTE, respectively. With this in mind, we consider inference attacks that take as input an OPE- or DTE-encrypted column and an auxiliary and public dataset and return a mapping from ciphertexts to plaintexts.

We stress that EDB systems are not designed to provide privacy but the much stronger requirement of *confidentiality*. As such, for an attack to be successful against an EDB it is not required to de-identify the records of the DB as would be the case, say, against a differentially-private DB [73]. In the setting of EDBs, an attack is successful if it recovers even *partial* information about a *single* cell of the DB. As we will see later, our attacks recover a lot more.

Concrete attacks. We study the effectiveness of four different attacks. Two are well-known and two are new:

- *frequency analysis:* is a well-known attack that decrypts DTE-encrypted columns given an auxiliary dataset that is "well-correlated" with the plaintext column. The extent of the correlation needed, however, is not significant and many publicly-available datasets can be used to attack various kinds of encrypted columns with this attack.

- *ℓ_p-optimization:* is a new *family* of attacks we introduce that decrypts DTE-encrypted columns. The family is parameterized by the ℓ_p-norms and is based on combinatorial optimization techniques.

- *sorting attack:* is an attack that decrypts OPE-encrypted columns. This folklore attack is very simple but, as we show, very powerful in practice. It is applicable to columns that are "dense" in the sense that every element of the message space appears in the encrypted column. While this may seem like a relatively strong assumption, we show that it holds for many real-world datasets.

- *cumulative attack:* is a new attack we introduce that decrypts OPE-encrypted columns. This attack is applicable even to low-density columns and also makes use of combinatorial optimization techniques.

Evaluating inference attacks. As discussed above, most inference attacks need an auxiliary source of information and their success depends on how well-correlated the auxiliary data is with the plaintext column. The choice of auxiliary data is therefore an important consideration when evaluating an inference attack. A strongly correlated auxiliary dataset may yield better results but access to such a dataset may not be available to the adversary. On the other hand, misjudging which datasets are available to the adversary can lead to overestimating the security of the system. An additional difficulty is that the "quality" of an auxiliary dataset is application-dependent. For example, census data may be well-correlated with a demographic database but poorly correlated with a medical database.

So the question of how to empirically evaluate inference attacks is non-trivial. In this work, we use the following methodology: (1) we choose a real-world scenario where the use of EDBs is *well-motivated*; (2) we consider encrypted columns from real-world data for the scenario under consideration; and (3) we apply the attack on the encrypted column using any relevant *publicly*-available auxiliary dataset.

Empirical results. For our empirical analysis, we chose databases for electronic medical records (EMRs) as our motivating scenario. Such medical DBs store a large amount of private and sensitive information about both patients and the hospitals that treat them. As such they are a primary candidate for the real-world use of EDBs and appear frequently as motivation in prior work.

To evaluate our attacks, we consider DTE- and OPE-encrypted columns for several attributes using real patient data from the U.S. hospitals provided by the National Inpatient Sample (NIS) database of the Healthcare Cost and Utilization Project (HCUP). [1]

[1] We stress that we strictly adhered to the HCUP data use agreement. In particular, our study is

Following are the highlights of our results:

- ℓ_2-*optimization* (vs. DTE-encrypted columns): the attack recovered the mortality risk and patient death attributes for 100% of the patients for at least 99% of the 200 largest hospitals. It recovered the disease severity for 100% of the patients for at least 51% of those same hospitals.

- *frequency analysis* (vs. DTE-encrypted columns): the attack had the same results as ℓ_2-optimization.

- *sorting attack* (vs. OPE-encrypted columns): the attack recovered the admission month and mortality risk of 100% of patients for at least 90% of the 200 largest hospitals.

- *cumulative attack* (vs. OPE-encrypted columns): the attack recovered disease severity, mortality risk, age, length of stay, admission month, and admission type of at least 80% of the patients for at least 95% of the largest 200 hospitals. For 200 small hospitals, the attack recovered admission month, disease severity, and mortality risk for 100% of the patients for at least 99.5% of the hospitals.

Discussion. Our experiments show that the attacks considered in this work can recover a large fraction of data from a large number of PPE-based medical EDBs. In light of these results it is clear that these systems *should not be used in the context of EMRs*. One may ask, however, how the attacks would perform against non-medical EDBs, e.g., against human resource DBs or accounting DBs. We leave this as important future work but conjecture that the attacks would be at least as successful considering that much of the data stored in such DBs is also stored in medical DBs (e.g., demographic information).

We also note that even though the attacks can already recover a considerable amount of information from the EDBs, the results presented in this work should be viewed as a *lower bound* on what can be extracted from PPE-based EDBs. The first reason is that the attacks only make use of leakage from the EDB and do not exploit the considerable amount of leakage that occurs from the queries to the EDB. The second reason is that our attacks do not even target the weakest encryption schemes used in these systems (e.g., the schemes used to support equi- and range-joins).

not concerned with the problem of de-anonymization. The data was not de-anonymized nor any attempt was made to do so.

3.1 Threat Model

An EDB system should protect a database against a variety of threats. In this Section, we describe some of these threats and propose an adversarial model that captures them. In defining such a model, we make two things explicit: (1) the goal of the attack; and (2) the information the adversary holds when carrying out the attack.

3.1.1 Adversarial Goals

There are at least two kinds of attacks on EDBs which we refer to as *individual* attacks and *aggregate* attacks.

Individual attacks. In an individual attack, the adversary is concerned with recovering information about a row in the database. For example, if the EDB is a medical database where each row corresponds to a patient, then the goal of the attack would be to recover information about a specific patient, e.g., its age or name.

Aggregate attacks. In an aggregate attack, the adversary wants to recover statistical information about the entire database. Again, in the context of a medical database, this could be information such as the total number of patients with a particular disease or the number of patients above a certain age. We note that, depending on the context, aggregate attacks can be extremely harmful. For example, hospitals do not disclose the number of cancer patients they treat so as not to signal anything about the quality of their cancer treatments.

3.1.2 Adversarial Information

PPE-based EDBs like [17, 28] are designed to protect against a semi-honest adversary that corrupts the server. Intuitively, this means that the adversary has access to everything the server sees but cannot influence it—in particular, it cannot make it deviate from the prescribed protocol. Since the adversary has complete access to what the server sees, it holds the encrypted database and can see the queries generated by the proxy.

Ciphertext-only. In this work, we focus on a considerably weaker adversary which has access to the encrypted database but not to the queries. We stress that

this is a much weaker adversary than what is typically considered in the literature and captures all the threats that database customers are typically concerned with. This includes internal threats like malicious database administrators and employees, and external threats like hackers, nation states, and organized crime.

Steady state EDBs. We assume the adversary has access to the encrypted database in *steady state*, which means that the onions of each cell are peeled down to the lowest layer needed to support the queries generated by the application. Intuitively, one can think of the steady-state EDB as the state of the EDB after the application has been running for a while.

Auxiliary information. In addition to the encrypted database, we assume our adversary has access to auxiliary information about the system and/or the data. Access to auxiliary information is standard in any practical adversarial model since the adversary can always consult public information sources to carry out the attack. In particular, we consider the following sources of auxiliary information:

- *application details*: the application running on top of the encrypted database, possibly obtained from accessing the application (e.g., if it is a web service) or from documentation;
- *public statistics*: publicly available statistics, for example, census data or hospital statistics;
- *prior versions*: prior versions of the database, possibly obtained through a prior data breach.

We stress that our experiments will make use of a different subset of auxiliary sources and that none of the attacks need access to all of these sources.

3.1.3 Attack Accuracy

When an adversary executes an inference attack, it receives as output an assignment from the encrypted cells to the elements of the message space. Though our experiments in Section 3.6 show that there are many attributes for which the attacks are perfectly accurate, this is not always the case and for low-accuracy attributes it could be difficult for the attacker to distinguish correct assignments from incorrect ones. We note, however, that the attacks can still be damaging even for these attributes for the following reasons. First, the adversary can still learn statistics

about the attribute which in some cases, like patient died during hospitalization or major diagnostic category, can be very sensitive for hospitals because it reveals information about the quality of their care. Second, the results can still be used for phishing-style attacks where the adversary only needs a small number of successes.

3.2 Attacking DTE Columns

We describe two attacks against DTE-encrypted columns. The first is the well-known frequency analysis and the second is a family of attacks we refer to as ℓ_p-optimization attacks. The family is parameterized by the ℓ_p norms.

Here, $\mathbb{C}_k$ and $\mathbb{M}_k$ are the ciphertext and message spaces of the deterministic encryption scheme. We assume $|\mathbb{C}_k| = |\mathbb{M}_k|$ but if this is not the case we simply pad $\mathbb{M}_k$. For encryption schemes $|\mathbb{C}_k|$ is always at least $|\mathbb{M}_k|$.

3.2.1 Frequency Analysis

Frequency analysis is the most basic and well-known inference attack. It was developed in the 9th century and is used to break classical ciphers. As is well-known, frequency analysis can break deterministic encryption and, in particular, deterministically-encrypted columns. Given a DTE-encrypted column $\mathbf{c}$ over $\mathbb{C}_k$ and an auxiliary dataset $\mathbf{z}$ over $\mathbb{M}_k$, the attack works by assigning the ith most frequent element of $\mathbf{c}$ to ith most element of $\mathbf{z}$. For ease of exposition, we assume that $\mathbf{c}$ and $\mathbf{z}$ have histograms that can be strictly ordered; that is, for all $i \neq j$, $\psi_i \neq \psi_j$ and $\pi_i \neq \pi_j$, where $\boldsymbol{\psi} = \mathsf{Hist}(\mathbf{c})$ and $\boldsymbol{\pi} = \mathsf{Hist}(\mathbf{z})$. More precisely, the attack is defined as:

- **Frequency-An**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{vSort}(\mathsf{Hist}(\mathbf{c}))$;
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{vSort}(\mathsf{Hist}(\mathbf{z}))$;
  3. output $\alpha : \mathbb{C}_k \rightarrow \mathbb{M}_k$ such that

$$\alpha(c) = \begin{cases} \boldsymbol{\pi}\big[\mathsf{Rank}_{\boldsymbol{\psi}}(c)\big] & \text{if } c \in \mathbf{c}; \\ \perp & \text{if } c \notin \mathbf{c}. \end{cases}$$

If the histograms are not strictly ordered (i.e., there are $i \neq j$ such that $\psi_i = \psi_j$ or $\pi_i = \pi_j$) one can still run the attack by breaking ties in the sorting steps arbitrarily. In the worst-case, each tie will be broken erroneously and induce an error in the assignment so this will cause the attack to err on $a + b$ ciphertexts, where a and b are the number of ties in $\mathsf{Hist}(\mathbf{c})$ and $\mathsf{Hist}(\mathbf{z})$, respectively. The attack runs in $O\big(|\mathbb{C}_k| \cdot \log |\mathbb{C}_k|\}\big)$ time.

3.2.2 ℓ_p-Optimization

We now describe a family of attacks against DTE-encrypted columns we refer to as ℓ_p-optimization. The family is parameterized by the ℓ_p norms. The basic idea is find an assignment from ciphertexts to plaintexts that minimizes a given cost function, chosen here to be the ℓ_p distance between the histograms of the datasets. This has the effect of minimizing the *total* mismatch in frequencies across all plaintext/ciphertext pairs. The attack works as follows.

Given a DTE-encrypted column $\mathbf{c}$ over $\mathbb{C}_k$ and auxiliary information $\mathbf{z}$ over $\mathbb{M}_k$, the adversary first computes the histograms $\boldsymbol{\psi}$ and $\boldsymbol{\pi}$ of $\mathbf{c}$ and $\mathbf{z}$, respectively. It then finds the permutation matrix X that minimizes the ℓ_p distance between the ciphertext histogram $\boldsymbol{\psi}$ and the permuted auxiliary histogram $X \cdot \boldsymbol{\pi}$. Intuitively, the attack finds the mapping of plaintexts to ciphertexts that achieves the closest overall match of their sample frequencies. Note that this is very different than frequency analysis which ignores the amplitude of the frequencies and only takes into account their rank. More precisely, the attack is defined as follows:

- ℓ_p**-Optimization**$(\mathbf{c}, \mathbf{z})$:
    1. compute $\boldsymbol{\psi} \leftarrow \mathsf{Hist}(\mathbf{c})$;
    2. compute $\boldsymbol{\pi} \leftarrow \mathsf{Hist}(\mathbf{z})$;
    3. output $\arg\min_{X \in \mathbb{P}_n} \|\boldsymbol{\psi} - X \cdot \boldsymbol{\pi}\|_p$;

where $\mathbb{P}_n$ is the set of $n \times n$ permutation matrices. Note that in the ℓ_1-optimization attack, Step 3 can be formulated as a *linear sum assignment problem* (LSAP) [74]. The LSAP can be solved efficiently using the well-known Hungarian algorithm [75, 76] or any linear programming (LP) solver. In our experiments we use the

former which runs in time $O(n^3)$. The precise LSAP formulation is:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij}X_{ij} \\
\textbf{subject to} \quad & \sum_{i=1}^{n} X_{ij} = 1, \quad 1 \le j \le |\mathbb{C}_k| \\
& \sum_{j=1}^{n} X_{ij} = 1, \quad 1 \le i \le |\mathbb{C}_k| \\
& X_{ij} \in \{0,1\}, \quad 1 \le i,j \le |\mathbb{C}_k|.
\end{aligned}$$

where the cost matrix $C = C_{ij}$ gives the cost of matching plaintext j to ciphertext i.

For $p = 1$, the costs are simply the absolute differences in frequency, so we set $C_{ij} = |\psi_i - \pi_j|$. For $2 \le p \le \infty$, however, Step 3 of the ℓ_p-optimization attack cannot be formulated directly as a LSAP because the ℓ_p norm is not a simple linear sum. Nevertheless, we show that it can still be efficiently solved using fast LSAP solvers. To see why, let $f_1 : \mathbb{R}^+ \to \mathbb{R}$ be the function $x \mapsto \sqrt[p]{x}$ and let $f_2 : \mathbb{N}_{\ge 0}^n \to \mathbb{N}_{\ge 0}$ be the function $\mathbf{v} \mapsto \sum_{i=1}^n v_i^p$. Then we note that the ℓ_p norm of a vector can be written as

$$\|\mathbf{v}\|_p = f_1\big(f_2(\mathbf{v})\big).$$

Since f_1 is monotone increasing, the vector that minimizes $f_1 \circ f_2$ is the vector that minimizes f_2. It follows then that for any vector $\mathbf{v}$, the vector $\mathbf{w}$ with the minimum ℓ_p distance from $\mathbf{v}$ is the solution to

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{n} |v_i - w_i|^p.$$

As long as $p < \infty$, this optimization problem can be formulated as a LSAP with cost matrix C such that $C_{ij} = |v_i - w_i|^p$. The attack takes $O(|\mathbb{C}_k|^3)$ time.

Remark on ℓ_p-optimization vs. frequency analysis. In our experiments, we found that frequency analysis and ℓ_p-optimization for $p = 2, 3$ performed equally well. In fact, for a fixed encrypted column and auxiliary dataset, they decrypted same exact ciphertexts. On the other hand, frequency analysis did consistently better than ℓ_1-optimization. This raises interesting theoretical and practical questions. From a theoretical perspective it would be interesting to understand the exact

relationship between frequency analysis and ℓ_p-optimization. Our experiments tell us that ℓ_1-optimization is different from frequency analysis (since they generated different results) but they did not distinguish between frequency analysis and ℓ_2- and ℓ_3-optimization. As such, it would be interesting to either separate the attacks or prove that they are equivalent for some $p \geq 2$.

From a practical perspective, the main question is what is the motivation for ever using ℓ_p-optimization over frequency analysis? The main reason is that ℓ_p-optimization not only decrypts an encrypted column but, while doing so, also produces cost information about the different solutions it finds. Like the cumulative attack we describe in Section 3.3.2, this is due to its use of combinatorial optimization. As it turns out, this extra information can be leveraged to attack "hidden" columns (i.e., for which we do not know the attribute); something we cannot always do with frequency analysis. We discuss this in more detail in Section 3.5.

3.3 Attacking OPE Columns

In addition to the frequency information leaked by DTE, order-preserving encryption also reveals the relative ordering of the ciphertexts. Here we describe two attacks on OPE-encrypted columns that exploit this additional leakage to recover even more of the plaintext data. Note that the attacks only make use of order information so they work even against columns encrypted with ORE [77] and interactive order-preserving protocols [78, 79]. In particular, since all OPE instantiations necessarily leak more than just the order [6], stronger attacks are likely possible against OPE-encrypted columns.

Here, $\mathbb{C}_k$ and $\mathbb{M}_k$ are the ciphertext and message spaces of the OPE scheme. We assume, without loss of generality, that $|\mathbb{C}_k| = |\mathbb{M}_k|$. If this is not the case we pad $\mathbb{M}_k$ with additional symbols until it holds.

3.3.1 Sorting Attack for Dense Columns

The first attack on OPE-encrypted columns is trivial and applicable to all columns that satisfy a condition we call *density*. We call an OPE-encrypted column δ-dense, if it contains the encryptions of at least a δ fraction of its message space. If $\delta = 1$, we simply say that the column is dense.

The attack is described in detail below and works as follows. Note that it does not require any auxiliary information. Given an OPE-encrypted dense column $\mathbf{c}$ over $\mathbb{C}_k$ the adversary simply sorts $\mathbf{c}$ and $\mathbb{M}_k$ and outputs a function that maps each ciphertext $c \in \mathbf{c}$ to the element of the message space with the same rank. More precisely, the attack is defined as:

- **Sorting-Atk**$(\mathbf{c})$:
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{vSort}\big(\mathsf{Unique}\big(\mathbf{c}\big)\big)$;
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{vSort}\big(\mathbb{M}_k\big)$;
  3. output $\alpha : \mathbb{C}_k \to \mathbb{M}_k$ such that:

$$\alpha(c) = \begin{cases} \boldsymbol{\pi}\big[\mathsf{Rank}_{\boldsymbol{\psi}}(c)\big] & \text{if } c \in \mathbf{c}; \\ \perp & \text{if } c \notin \mathbf{c}. \end{cases}$$

The attack runs in $O\big(|\mathbb{C}_k| \cdot \log |\mathbb{C}_k|\big)$ time.

3.3.2 Cumulative Attack for Low-Density Columns

The main limitation of the sorting attack is that it is only applicable to dense columns. To address this, we describe a second attack for low-density OPE-encrypted columns we refer to as the *cumulative* attack. The attack requires access to auxiliary information and can recover a large fraction of column cells (see Section 3.6.2 for details).

Intuition. Given a DTE-encrypted column, the adversary learns the sample frequency of each ciphertext in the column. These sample frequencies make up the histogram for the encrypted column, and we showed in the previous section how the adversary can use them to match the DTE ciphertexts to their plaintexts by finding (c, m) pairs where c and m have similar frequencies.

Given an OPE-encrypted column, the adversary learns not only the frequencies but also the relative ordering of the encrypted values. Combining ordering with frequencies, the adversary can tell for each ciphertext c what fraction of the encrypted values are less than c. More formally, this is known as the *empirical cumulative distribution function* (ECDF, or simply CDF) of the data set.

In the cumulative attack, we leverage the CDF to improve our ability to match plaintexts to ciphertexts. Intuitively, if a given OPE ciphertext is greater than 90%

of the ciphertexts in the encrypted column $\mathbf{c}$, then we should match it to a plaintext that also is greater than about 90% of the auxiliary data $\mathbf{z}$. Although our early experiments showed that CDFs alone enable very powerful attacks on OPE, we can achieve even better results using both the CDFs and the frequencies together. Here we use an LSAP solver to find the mapping of plaintexts to ciphertexts that minimizes the total sum of the mismatch in frequencies plus the mismatch in CDFs across all plaintext/ciphertext pairs.

Overview of attack. The attack is detailed below and works as follows. Given an OPE-encrypted column $\mathbf{c}$ over $\mathbb{C}_k$ and an auxiliary dataset $\mathbf{z}$ over $\mathbb{M}_k$, the adversary computes the histograms $\boldsymbol{\psi}$ and $\boldsymbol{\pi}$ and the CDFs $\boldsymbol{\varphi}$ and $\boldsymbol{\mu}$ of $\mathbf{c}$ and $\mathbf{z}$, respectively. It then finds the permutation that simultaneously matches both the sample frequencies and the CDFs as closely as possible. More precisely, the attack is defined as:

- **Cumulative-Atk**$(\mathbf{c}, \mathbf{z})$:
  1. compute $\boldsymbol{\psi} \leftarrow \mathsf{Hist}(\mathbf{c})$ and $\boldsymbol{\varphi} \leftarrow \mathsf{CDF}(\mathbf{c})$;
  2. compute $\boldsymbol{\pi} \leftarrow \mathsf{Hist}(\mathbf{z})$ and $\boldsymbol{\mu} \leftarrow \mathsf{CDF}(\mathbf{z})$;
  3. output
$$\arg\min_{X \in \mathbb{P}} \sum_{i=1}^{|\mathbb{M}_k|} \left(\left| \psi_i - X_i \cdot \boldsymbol{\pi} \right| + \left| \varphi_i - X_i \cdot \boldsymbol{\mu} \right| \right)$$

where $\mathbb{P}$ is the set of all $|\mathbb{C}_k| \times |\mathbb{C}_k|$ permutation matrices. Note that, as in Section 3.2.2 above, Step 3 of this attack can be formulated as an LSAP which can be efficiently solved using the Hungarian algorithm. The precise LSAP formulation is:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij} \\
\textbf{subject to} \quad & \sum_{i=1}^{n} X_{ij} = 1, \quad 1 \le j \le |\mathbb{C}_k| \\
& \sum_{j=1}^{n} X_{ij} = 1, \quad 1 \le i \le |\mathbb{C}_k| \\
& X_{ij} \in \{0, 1\}, \quad 1 \le i, j \le |\mathbb{C}_k|.
\end{aligned}$$

where the cost matrix C gives the cost for mapping plaintext m_j to ciphertext c_i as the sum of the mismatch in frequencies plus the mismatch in cumulative

frequencies:

$$C_{ij} = \left|\psi_i - \pi_j\right|^2 + \left|\varphi_i - \mu_j\right|^2.$$

The attack runs in $O(|\mathbb{C}_k|^3)$ time.

3.4 Simulating a Medical EDB

To evaluate the attacks, we considered the scenario of an EMR application and its associated database. We chose this setting for several reasons.

First, medical DBs hold highly personal and sensitive information and are often covered by privacy regulations such as the Health Portability and Accountability Act (HIPAA). EMRs are vulnerable to insider and outsider threats and are increasingly targeted by professional attackers including state sponsored adversaries and organized crime. This trend is illustrated by the recent attacks on Anthem—one of the largest U.S. health insurance providers—which compromised the health records of 80 million individuals. In fact, the Ponemon Institute's recent study on *Privacy and Security of Healthcare Data* [80] reports that criminal attacks are now the number one cause of healthcare data breaches with a 125% growth in attacks reported in the last 5 years. As such, the motivation to encrypt medical DBs is very strong. In fact, medical DBs often appear as the standard motivation in the encrypted database research literature (see, e.g., [17]).

Another reason we chose this scenario is that a subset of the data stored in EMRs (e.g., demographic data) is also held in other types of sensitive DBs including human resources DBs, accounting DBs, and student DBs. Information stored in these DBs may also be covered by privacy regulations such as the Family Educational Rights and Privacy Act (FERPA). Our results against medical DBs can therefore tell us something about these other kinds of DBs.

3.4.1 Target Data

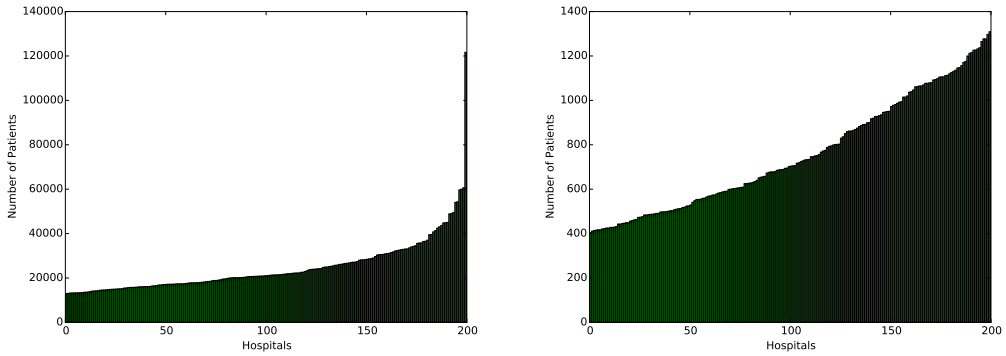
Throughout, we refer to the data we use to populate the EDB as the *target* data. In our experiments we use data from the National Inpatient Sample (NIS) database of the Healthcare Cost and Utilization Project (HCUP) [81]. HCUP makes available the largest collection of longitudinal hospital care data in the U.S. The NIS database—which includes data on inpatients (i.e., patients that stay at a hospital for

at least one night) from all the hospital in the U.S.—is available starting from 1988. The database is made available to researchers under controlled access: an online training is required and a data use limitation agreement must be signed before the data can be purchased and used. The NIS database includes attributes such as age, drugs, procedures, diagnosis, length of stay, etc. For our purposes, we only use a subset of the attributes (mostly due to space limitations) which we describe in Figure 3.2.

	Max	**Min**	**Mean**	**SD**
Large Hospitals	121,664	12,975	24,486	12,015
Small Hospitals	1,309	404	756	253

Table 3.1: Size of hospitals in number of patients

In our experiments we use the data from a subset of 1050 hospitals in the 2009 HCUP NIS database as our target data. We note that any other year would have given similar results. For all but one of our experiments we use the 200 largest hospitals but for the evaluation of the cumulative attack against low-density columns we use data from 200 small hospitals. The 200 small hospitals are the ones ranked (in decreasing order) 701 through 900 in terms of patient-size. Smaller hospitals had too few patients to attack (less than 400 and some even less than 10). The number of patients in the 200 largest and the 200 small hospitals is shown in Table 3.1.



(a) Number of patients in large hospitals **(b)** Number of patients in small hospitals

Figure 3.1: Size of hospitals in number of patients

Target attributes. We chose a subset of columns/attributes from the 2009 HCUP NIS dataset to attack. These attributes are listed in Figure 3.2. We believe these or similar attributes would be present in most real-world EMR systems. We

- **Sex.** Sex can be either male or female. The most prominent feature of the sex attribute is that most hospitals have more female patients than male patients. This is possibly due to pregnancy, births, and the fact that women live longer. Sex is universally used in all databases that store information about people.
- **Race.** Race can have the following values: white, black, Hispanic, Asian or Pacific Islander, Native American, and other. Race is stored in most databases dealing with people for a variety of reasons.
- **Age.** Age can range from 0 to 124. Age 0 is for babies less than an year old. Some databases may store birth year instead of age, e.g., as part of full date of birth. Frequency counts for age and birth year are exactly the same.
- **Admission Month.** Admission month has values that range from January to December.
- **Patient died during hospitalization.** This attribute indicates whether a patient died during hospitalization.
- **Primary Payer.** Primary payer has six values: Medicare, Medicaid, private or health maintenance organization, self-pay, no charge, and other.
- **Length of Stay.** Length of stay ranges from 0 to 364 and represents the number of days a patient spends in a hospital. It is a very sensitive attribute and reveals information about other attributes such as the nature of the patient's disease.
- **Mortality Risk.** Mortality Risk has four values showing the likelihood of dying: minor, moderate, major, and extreme. It indicates the risk of a patient dying in the hospital.
- **Disease Severity.** Disease Severity has four values showing loss of function: minor (indicates cases with no comorbidity or complications), moderate, major, and extreme. It indicates the severity of the patient's disease.
- **Major Diagnostic Category.** Major Diagnostic Category has 25 values and gives the principal diagnosis such as "Diseases and Disorders of Kidney", "Burns", "Human Immunodeficiency Virus Infection (HIV)", etc.
- **Admission Type.** Admission type has six values: emergency, urgent, elective, newborn, trauma center, and other.
- **Admission Source.** Admission source has five values: emergency room, another hospital, another facility including long-term care, court/law enforcement, and routine/birth/other. It indicates from where the patient was admitted to the hospital.

Figure 3.2: Attributes/columns used in our evaluation.

confirmed that six of them, including sex, race, age, admission month, patient died, and primary payer, are used by OpenEMR [82], which is an open source fully-functional EMR application. We stress that the form in which these attributes are stored can vary (e.g., age can be stored as an integer or computed from a date of birth) but some variant of these attributes exist in OpenEMR.

To decide whether an attribute should be DTE or OPE-encrypted we did the following. For the attributes stored by OpenEMR (in some form), we simply checked the kinds of operations OpenEMR supported on it. If it supported either range queries or sorting operations, we considered it an OPE attribute. If OpenEMR supported equality queries on the attribute we considered it a DTE attribute. For the remaining attributes, we made assumptions which we believe to be reasonable. More specifically, we assumed an EMR system would support range queries on the length of stay attribute; sorting queries on the mortality risk, disease severity, and admission type attributes (e.g., for triage); and equality queries on major diagnostic category and admission source.

3.4.2 Auxiliary Data

All but one of our attacks (sorting) require an auxiliary dataset to decrypt a PPE-encrypted column. We used the following two auxiliary datasets:

Texas PUDF data. The first auxiliary dataset we use is the Texas Inpatient Public Use Data File (PUDF), which is provided by the Texas Department of State Health Services. This dataset—unlike the HCUP NIS data—is publicly available online so there is no reason to believe an adversary would not use it to her advantage. Specifically, we use the 2008 Texas PUDF data. The Texas PUDF data until year 2008 can be downloaded from [83]. Usage of the data requires an acceptance of a data use agreement but we believe it is reasonable to assume that an adversary would not comply with such an agreement.

2004 **HCUP NIS.** Unfortunately, the Texas PUDF data has a limited number of attributes which prevents us from studying the accuracy of our attacks on several attributes of interest. We therefore also run experiments using the 2004 HCUP NIS database as auxiliary data (recall that our target data is the 2009 HCUP NIS data). Note that each year of the HCUP NIS data comes from a random sample of hospitals from a large number of U.S. hospitals and the entire data of each sampled hospital is included. This means that the 2004 HCUP NIS data is not only different

in time from the 2009 HCUP NIS data but it is also comes from a different set of hospitals. There is a small number of common hospitals between 2004 and 2009 HCUP NIS databases (less than 4%), but that does not have a noticeable impact on our experimental results.

Remark on additional datasets. Another example of a publicly-available auxiliary dataset is the Statewide Planning and Research Cooperative System (SPARCS) Inpatient data from the state of New York [84]. We do not report results using SPARCS as auxiliary data due to space limitations, but it gives similar results to those using the Texas PUDF data.

3.5 Experimental Setup

All experiments were conducted on a high-end Mac laptop with Intel Core i7 processor and 16GB memory running OS X Yosemite (v10.10.2). We used Python version 2.7.6 and Matlab version 8.4.0 (R2014b). For our experiments we developed three tools: Parser, Column Finder, and Revealer which we now describe.

Parser. Parser is written in Python and parses the target and auxiliary data to create appropriate histograms. In the case of the target data, it creates one histogram per attribute/hospital pair. More precisely, for each pair it creates a histogram that reports the number of times some value v of the attribute appears in the hospital's data. In the case of the auxiliary data, it creates a single histogram for each attribute (i.e., over all hospitals).

Column Finder. Column Finder is also written in Python. Since CryptDB-like EDB systems encrypt column names, an adversary first needs to learn which encrypted columns correspond to the attribute of interest. We do this using the following approach. First, we determine if the attribute of interest is present in the EDB by checking the database schema of the application. Then we run Column Finder which works as follows:

1. it determines the number of distinct values for the column of interest in the auxiliary data. We'll refer to this column as the auxiliary column. As an example, Column Finder would use the auxiliary data to learn that age has 125 possible values or that sex has 2 possible values.

2. it then determines the number of distinct values stored in each DTE- and OPE-

encrypted column of the EDB. This is trivial due to the properties of these encryption schemes. It then searches through these encrypted columns to find the ones that have approximately the same number of distinct values as the auxiliary column. We have to search for approximate matches since some values of an attribute may not be present in the target data. Since we know from the database schema of the application that the EDB contains an encrypted column for the attribute of interest, this step will find at least one column:

(a) If it finds only one column, then that is the encrypted column for the attribute of interest.

(b) If it finds more than one column with a close-enough number of distinct values such that it cannot determine which column belongs to the attribute of interest, then it outputs all of them.

Auxiliary Attribute	Target Attributes	Accuracy
Primary Payer	Admission Type, Primary Payer, Race	116
Race	Admission Type, Primary Payer, Race	152
Admission Type	Admission Type, Primary Payer, Race	128
Sex	Sex, Patient Died	200
Patient Died	Sex, Patient Died	200

Table 3.2: Column recovery: the accuracy column reports the number of hospitals for which the correct attribute (i.e., from the auxiliary attribute column) had the lowest ℓ_2-optimization cost among all target attributes.

Data Revealer. Revealer is written in Matlab and implements frequency analysis, ℓ_2-optimization, and the cumulative attack. The last two attacks use the Hungarian algorithm for the optimization step. We did not implement the sorting attack against dense columns since correctness and perfect accuracy is trivially true (we do run experiments to report the prevalence of dense columns in our target dataset and results are shown in Figure 3.5). Revealer takes as input the histogram of an auxiliary column from the output of Parser and the histograms for a set of target encrypted columns from the output of Column Finder. So, depending on the output of Column Finder, Revealer can receive either a single target histogram or multiple target histograms and in each case it works as follows:

- if it receives a single target histogram, Revealer simply runs the attack with its two inputs.

- if it receives multiple target histograms, Revealer runs one of the optimization-based attacks on the auxiliary histogram with *each* of the target histograms. It then outputs the result with the minimum cost.

Note that only the ℓ_p-optimization and cumulative attacks can be executed when there are multiple target histograms since frequency analysis does not have an inherent notion of cost that can be used. In our experiments, we found that when the target and auxiliary attributes are the same, the cost is significantly less than when they are different. This is reported in Table 3.2.

Time measurements. All the attacks take less than a fraction of a second per hospital. Table 3.3 reports the running times (averaged over 200 hospitals) for each attack over a different set of attributes; each with a different number of values.

Notice that attacking the length of stay column requires considerably more time than the rest. This is due to the fact that it has a large number of values (365) which especially affects the running time of ℓ_2-optimization and cumulative attacks which rely on optimization. Currently, our attacks are implemented in Matlab which is very slow compared to other languages like C so we believe that a C implementation would decrease the running time significantly.

Attributes (# of values)	**Frequency Analysis**	**ℓ_2-optimization**	**Cumulative**
Sex (2)	0.11ms	0.11ms	0.31ms
Mortality Risk (4)	0.12ms	0.12ms	0.49ms
Admission Source (5)	0.12ms	0.13ms	0.60ms
Major Diagnostic Category (25)	0.19ms	0.20ms	3.5ms
Age (125)	0.63ms	3.03ms	311.6ms
Length of stay (365)	1.73ms	68.7ms	35,910ms

Table 3.3: Time (in milliseconds) of attacks per hospital.

3.6 Experimental Results

For each hospital and each column in the EDB, we compute the accuracy of our attack as the number of encrypted cells for which the recovered plaintext matches the ground truth, divided by the total number of column cells.

We present the results of these experiments in Figures 3.4, 3.3 and 3.6. Each plot shows the *empirical* CCDF (Complementary Cumulative Distribution Function) of our record-level accuracy across all the hospitals in our target data. For example,

a point at location (x, y) indicates that we correctly recovered at least x fraction of the records for y fraction of the hospitals in the target data. The results show that our attacks recover a substantial fraction of the encrypted DBs and perform significantly better than random guessing.

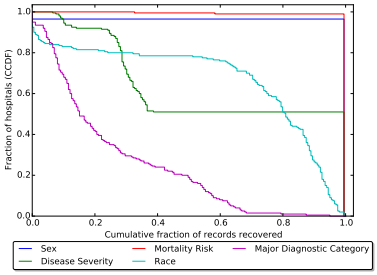
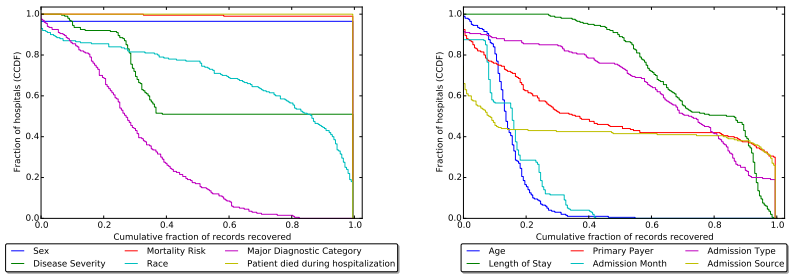
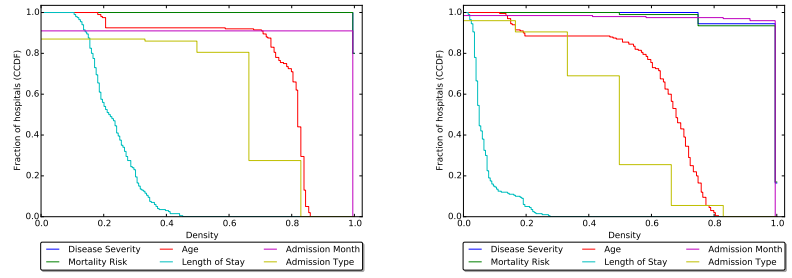


Figure 3.3: Results of ℓ_2-optimization on DTE-encrypted columns on 200 largest hospitals with 2009 HCUP NIS as target data and Texas PUDF as auxiliary data



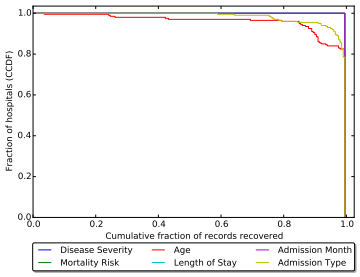
(a) First set of attributes **(b)** Second set of attributes

Figure 3.4: Results of ℓ_2-optimization on DTE-encrypted columns on 200 largest hospitals with 2009 HCUP NIS as target data and 2004 HCUP NIS as auxiliary data

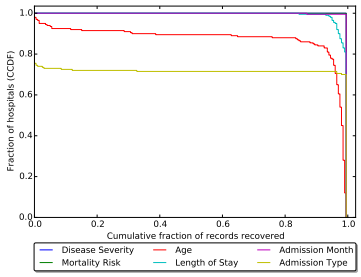


(a) Density for large hospitals **(b)** Density for small hospitals

Figure 3.5: Density – Ratio of the number of values of an attribute present in a column to the total number of values of the attribute



(a) Results for the 200 largest hospitals with 2009 HCUP NIS as target data and 2004 HCUP NIS as auxiliary data.



(b) Results for the 200 small hospitals with 2009 HCUP NIS as target data and 2004 HCUP NIS as auxiliary data.

Figure 3.6: Results of Cumulative attack on OPE-encrypted columns

3.6.1 Attacks on DTE-Encrypted Columns

Figure 3.4 shows the results of our ℓ_2-optimization attack against DTE-encrypted columns using the 2004 HCUP NIS dataset as auxiliary. Figure 3.3 shows results of the same attack using Texas PUDF dataset as auxiliary.

Using the 2004 HCUP NIS as auxiliary data (Figure 3.4), ℓ_2-optimization recovers cells for a significant number of patients, even for attributes with a large number of distinct values such as Age and Length of Stay. It recovered Mortality Risk and whether the patient died for 100% of the patients for 99% and 100% of the hospitals respectively. It also recovered the Disease Severity for 100% of the patients for 51% of the hospitals. The attack recovered Race for at least 60% of the patients for at least 69.5% of the hospitals; Major Diagnostic Category for at least 40% of the patients for 27.5% of the hospitals; Primary Payer for at least 90% of the patients for 37.5% of the hospitals; Admission Source for at least 90% of the patients for 38% of the hospitals; Admission Type for at least 60% of the patients for 65% of the hospitals.

Perhaps surprisingly, ℓ_2-optimization also recovered a relatively small but significant fraction of cells for the Age attribute. Recovering DTE-encrypted Ages is very difficult because Age takes on a large range of values, and multiple values have very similar frequencies. Nonetheless, it recovered Age for at least 10% of patients for 84.5% of the hospitals. The attack also works surprisingly well for Length of Stay despite its large range of 365 possible values: specifically, it recovers this attribute for at least 83% of the patients for 50% of the hospitals. The reason for this unexpected accuracy is that most patients stay in the hospital for only a few

days. Therefore, by decrypting the plaintexts for a few very common lengths of stay (e.g., $1, 2, 3, \ldots$), we recover a large fraction of the database.

Using the Texas PUDF data as auxiliary (Figure 3.3), the attack performs similarly well. There is a small decrease in accuracy for Race and Major Diagnostic Category. We believe this is due to regional differences in demographics across the U.S.

3.6.2 Attacks on OPE-Encrypted Columns

The sorting attack succeeds only if a column has density 1, meaning that all possible values of an attribute are present in both the target and the auxiliary data. If this condition holds, the sorting attack can recover *all* the OPE-encrypted cells in a column; otherwise it fails. Figure 3.5 shows the density for six selected attributes for the large and small 200 hospitals, respectively, of the 2009 HCUP NIS dataset. For large hospitals, the density is 1 for 100% of the hospitals for Disease Severity and Mortality Risk and 90% of the hospitals for Admission Month. For small hospitals, the density is 1 for 95% of the hospitals for Disease Severity, Mortality Risk, and Admission Month. It follows that the sorting attack would recover 100% of the cells for these columns for these hospitals.

To evaluate the cumulative attack, we executed it over both large and small hospitals since the latter tend to have lower density on many attributes. Figure 3.6 shows the results. For large hospitals (Figure 3.6a) the attack performed extremely well, even for low-density attributes. It recovered at least 80% of the patient records for 95% of the hospitals for all the attributes shown in Figure 3.6a. The attack recovered Admission Month, Disease Severity, and Mortality Risk for 100% of the patients for 100% of the hospitals; Length of Stay for at least 99.77% of the patients for 100% of the hospitals; Age for at least 99% of the patients for 82.5% of the hospitals ; and Admission Type for 100% of the patients for 78.5% of the hospitals.

For small hospitals (Figure 3.6b), despite the attributes' low densities, the attack still performed surprisingly well. It recovered Disease Severity and Mortality Risk for 100% of the patients for 100% of the hospitals; Admission Month for 100% of the patients for 99.5% of the hospitals; Length of Stay for at least 95% of the patients for 98% of the hospitals; Age for at least 95% of the patients for 78% of the hospitals; and Admission Type for 100% of the patients for 69.5% of the

hospitals.

Chapter 4

A Practical Model for Searching on Encrypted Data

In the last chapter, we showed that property-preserving encryption is completely insecure, even against the weakest adversary which has access to just the encrypted data. In this chapter, we develop a new primitive called Blind Storage and use it to develop a symmetric searchable encryption scheme, which is provably secure against the adversary with access to just the encrypted data. Symmetric searchable encryption still leaks significant information and this leakage can be exploited in some settings, however, symmetric searchable encryption is much more secure than property-preserving encryption.

In recent years, searchable symmetric encryption (SSE) has emerged as an important problem at the intersection of cryptography, cloud storage, and cloud computing. SSE allows a client to store a large collection of encrypted documents with a server, and later quickly carry out keyword searches on these encrypted documents. The server is required to not learn any more information from this interaction, beyond certain patterns (if two searches involve the same keyword, and if the same document appears in the result of multiple searches, but not the actual keywords or the contents of the documents).

A long line of recent work has investigated SSE with improved security, more flexible functionality and better efficiency [50, 53, 57, 58, 7]. The techniques in all these works build on the early work of [53, 85]. In this work we present a radically different approach that achieves stronger security guarantees and flexibility, with significant performance improvements. In particular, our construction enjoys the following features:

- Dynamic SSE, which supports adding and removing documents at any point during the life-time of the system.
- The server is "computation free". Indeed, the only operations that need to be supported by the server are uploading and downloading blocks of data, if possible, parallelly. This makes our system highly scalable, and any

optimizations in these operations (e.g., using a content delivery network) will be directly reflected in the performance of the system.

- The information revealed to the server ("leakage functions") is strictly lesser than in all prior Dynamic SSE schemes except [59]. Scheme of [59] reveals less information to the server at the expense of poly-logarithmic overhead on top of Dynamic SSE overhead of other schemes (including ours).

- Satisfies a fully adaptive security definition, allowing for the possibility that the search queries can be adversarially influenced based on the information revealed to the server by prior searches.

- Security is in the standard model, rather than the heuristic Random Oracle Model; relies only on the security of block ciphers and collision resistant hash functions.

- Optional *document-set privacy*. The number of documents in the system and their lengths can be kept secret, revealing the existence of a document only when it is accessed by the client (typically after learning that a keyword appears in that document). This allows one, for instance, to archive e-mail with support for keyword searching, while keeping the number and lengths of e-mails hidden from the server (until each one is retrieved).

A simple prototype has been implemented to demonstrate the efficiency of the system.

Blind Storage. An important contribution of this work is to identify a more basic primitive that we call *Blind Storage*, on which our Dynamic SSE scheme is based. A Blind Storage scheme allows a client to store a set of files on a remote server in such a way that the server does not learn how many files are stored, or the lengths of the individual files; as each file is retrieved, the server learns about its existence (and can notice the same file being downloaded subsequently), but the file's name and contents are not revealed. Our Blind Storage scheme also supports adding new files and updating or deleting existing files. Further, though not needed for the Dynamic SSE construction, our Blind Storage scheme can be used so that the actual operation — whether it is reading, writing, deleting or updating — is hidden from the server.

Though not the focus of this work, we remark that a Blind Storage system would have direct applications in itself, rather than as a tool in constructing flexible and

efficient Dynamic SSE schemes. As our Blind Storage scheme does not make requirements on the server other than storage, it can be used with commodity storage systems such as Dropbox. This enables a wide range of simple applications that can take advantage of modular privacy protections to operate at a large scale and low expense but with strong privacy guarantees. Applications can range from backing up a laptop to archiving patient records at a hospital. Further, in our dynamic SSE scheme, document set privacy with relatively low overhead is made possible because we can simply store all the documents in the same Blind Storage system that is used to implement the SSE scheme.

4.1 Overview

In this section, we briefly discuss our techniques and the advantages of our scheme compared to prior SSE constructions. Most of the advantages follow from the simplicity of our scheme, and in particular, as depicted in Figure 4.1, from the fact that our server is computation free.

Techniques. Our main construction is that of a versatile tool called Blind Storage, which is then used to build a full-fledged SSE scheme. A Blind Storage scheme lets the client keep all information — including the number and size — about files secret from the server storing them, until they are accessed. In building the SSE scheme, the *search index* entries for all the keywords are stored as individual files in the Blind Storage scheme (with care taken to facilitate updates).

Our Blind Storage scheme, called SCATTERSTORE, is constructed using a simple, yet powerful technique: each file is stored as a collection of blocks that are kept in pseudorandom locations; the server sees only a *super-set* of the locations where the file's blocks are kept, and not the exact set of locations.[1] The key security property this yields us is that, from the point of view of the server, each file is associated with a set of locations *independent of the other files* in the system. (Indeed, the sets of locations for two files can overlap.)

A rigorous probabilistic analysis shows that for appropriate choice of parameters, the probability that any information about files not yet accessed is leaked to the server can be made negligible (say, 2^{-40} or 2^{-80}), with a modest blow-up in the storage and communication costs (e.g., by a factor of 4) over unprotected storage.

[1] To the extent that extra blocks are read, our scheme is similar to existing Oblivious RAM constructions. However, in our case, the overhead of extra blocks is bounded by a constant factor.

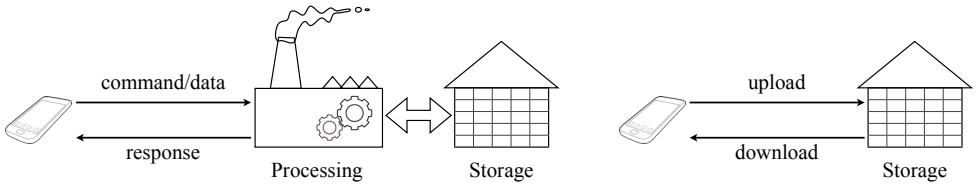


Figure 4.1: Contrasting the architecture of existing SSE Schemes (on the left) with that of the proposed scheme.

The only cryptographic tools used in our scheme are block ciphers (used for standard symmetric key encryption as well as for generating pseudorandom locations where the data blocks are kept) and collision resistant hash functions. The security parameters for these tools are chosen independently of the other parameters in the scheme.

Architecture. Most of the previous SSE schemes were presented as using a dedicated server, that performed both storage and computation. (See Figure 4.1.) The computation typically involved an (unparallelizable) sequence of decryptions. To deploy such a scheme using commodity services, one would need to rely not only on cloud storage services, but also cloud computation services. This presents several limitations. Firstly, this limits the choice of service providers available to a user: one could use Amazon EC2 for computation, combined with Amazon S3 for storage; however, it is not viable to use Dropbox for persistent storage and Amazon EC2 for computation, as this would incur high costs for communication between these two services. Storage and compute clusters are physically separated in modern data centers. This would add additional latency in all dynamic SSE schemes except ours, as data needs to be transmitted from storage nodes to compute nodes over the data center network. In contrast, our system can be easily implemented using Dropbox or other similar services which provide only storage. Secondly, relying on cloud computation makes the deployment less flexible, as it is harder to change choices like that of the operating system (due to pricing changes or technical support, for instance).

Another important issue in existing schemes is that one relies on availability and trust assumptions (e.g., honest-but-curious) for both computation and storage. Clearly, it is desirable to trust storage alone, as is the case in our scheme. Further, in ongoing work, we consider obtaining security against actively corrupt (rather than honest-but-curious) servers; this is easier and more efficient to achieve starting from our scheme, since we need to enforce honest behavior on part of a server that

provides storage alone.

Finally, it is significantly cheaper to rely on a cloud-storage service alone than on cloud computation (plus persistent storage).

Security definition. An important feature of our schemes is the stronger and easier to understand security guarantees. All the information leaked to the server is fully captured in relatively simple functionalities. For the Blind Storage scheme, as shown in Figure 4.2, each time a file is accessed, the functionality $\mathcal{F}_{\text{STORE}}$ reveals just a triple $(\mathsf{op}, j, \mathsf{size})$ to the server, where op specifies what the access operation is (read, write, update or delete), j specifies the last time, if any, the same file was accessed, and size specifies the size of the file.

The functionality $\mathcal{F}_{\text{SSE}}$ (shown in Figure 4.7) specifies all the information revealed by our SSE scheme. It is slightly more complex, partly because it allows the client to reuse document IDs. Further, it offers a higher level of secrecy for documents that are originally in the system, compared to those added later during the operation of the system.

Fully Adaptive Security. We achieve fully adaptive security, without relying on heuristics like the random oracle model. Technically, this is a consequence of the fact that the server does not carry out any decryptions. We point out that achieving adaptive security by making the client do decryptions for the server would not be viable in existing SSE schemes because a long sequence of decryptions (that cannot be parallelized) need to be carried out; several rounds of communication (with attendant network delays) would be necessary if the client carries out these decryptions for the server. Nevertheless, a similar approach was mentioned in [7] as a theoretical solution to avoid the Random Oracle Model and retain adaptive security.

The price we pay for the improved security, greater computational efficiency, parallelizability and simpler architecture is that the server storage and communication costs are possibly higher than that of some of the existing schemes (e.g., a factor of 2 to 4 over unprotected storage, which is in fact, comparable to overheads incurred in some other schemes like that of [7]). Also our SSE scheme could, in principle, involve up to three rounds of communication for retrieving the documents (this happens if the keyword has a large number of matching documents). In contrast, many existing schemes involve only two rounds (one to retrieve encrypted list of documents, and one to retrieve the documents themselves).

Comparative Performance. The most natural prior work for us to compare

against is [57] (though, unlike this work, it uses the Random Oracle Model). We remark that the more recent work of [7] augments the functionality of [57] (but without support for dynamic updates), and provides a highly streamlined implementation over very large scale data; however, *for the task of simple keyword searches,* its algorithm remains comparable to [57]. Since [57] reports performance of a prototype implemented in a comparable environment as ours (conservative comparison: we use a laptop and they used a server), we compare with it. Asymptotically, the client-side storage and computation in our system is same as [57], but the constants for our scheme are much better, and is reflected in the performance measured. Our scheme completely avoids server-side computation (which is quite significant in [57]).

4.2 Blind Storage

An important contribution of this thesis is to identify a versatile primitive called Blind Storage. It allows a client to store a set of files with a remote server, revealing to the server neither the number nor the sizes of the files. The server would learn about the existence of a file (and its size, but not the name used by the client to refer to the file, or its contents) only when the client retrieves it later. We also allow the client to add new files, and to update or delete existing files. The client's local storage should be independent of the total amount of data stored in the system.

In this section, first we present the definition of a Blind Storage system, followed by an efficient construction ScatterStore, and a proof of security. Later, in Section 4.3.2, we show how to build a Dynamic SSE scheme using a Blind Storage system.

4.2.1 Definition

Below, first we define the syntax of a Blind Storage system (and the infrastructure it needs), followed by the security requirements on it.

The Syntax. A blind storage system consists of a client and a "dumb" storage server. The server is expected to provide only two operations, download and upload. The data is represented as an array of *blocks*; the download operation is allowed to specify a list of indices of blocks to be downloaded; similarly, the upload operation

- On receiving the command $\mathcal{F}_{\text{STORE}}.\mathsf{Build}$ from the client:
  - $\mathcal{F}_{\text{STORE}}$ accepts input $(d_0, \{\mathsf{id}_i, \mathsf{data}_i\}_{i=1}^{t})$ from the client (where d_0 is an upperbound on the total number of data blocks to be stored in the system at any time, and the rest specify files to be stored in the system initially); it internally stores the specified files.
  - **Build Leakage:** In addition, $\mathcal{F}_{\text{STORE}}$ sends d_0 to the server.
- On receiving the command $\mathcal{F}_{\text{STORE}}.\mathsf{Access}(\mathsf{id}, \mathsf{op})$ from the client:
  - If no file matching the identifier id exists, and the operation $\mathsf{op} \in \{\mathsf{read}, \mathsf{delete}\}$, $\mathcal{F}_{\text{STORE}}$ returns a status message to the client indicating so. Else, if $\mathsf{op} = \mathsf{read}$, $\mathcal{F}_{\text{STORE}}$ returns the file with identifier id; if $\mathsf{op} = \mathsf{delete}$, it is removed. If $\mathsf{op} = \mathsf{write}$, the content data for the file is also accepted from the client, and the file is created or its content replaced with data. If $\mathsf{op} = \mathsf{update}$, $\mathcal{F}_{\text{STORE}}$ interacts with the client as follows:
    - $\mathcal{F}_{\text{STORE}}$ returns the current size of the file (in blocks – possibly 0, if the file does not exist) to the client.
    - $\mathcal{F}_{\text{STORE}}$ accepts the size of the updated file from the client.
    - $\mathcal{F}_{\text{STORE}}$ returns the current contents of the file to the client.
    - $\mathcal{F}_{\text{STORE}}$ accepts the updated contents of the file from the client. The file stored internally is updated with this.
  - **Access Leakage:** In addition, $\mathcal{F}_{\text{STORE}}$ sends the tuple $(\mathsf{op}, j, \mathsf{size})$ to the server where:
    - op specifies what the current access operation is,[a]
    - j is the last instance when the same file was accessed ($j = 0$ means that this file was not accessed before)
    - size is the size (in number of blocks) of the file being accessed. For the update operation, size is the larger of the sizes before and after the update.

[a] A refined version of Blind-Storage would require the operation to be not revealed. See Section 4.2.2.

Figure 4.2: The $\mathcal{F}_{\text{STORE}}$ functionality: all the information leaked to the server in our Blind Storage scheme is specified here.

is allowed to specify a list of data blocks and indices for those blocks.

A blind storage system is defined by three polynomial-time algorithms on the client-side: KeyGen, BSTORE.Build and BSTORE.Access. Of these, BSTORE.Access is an interactive protocol.

- KeyGen takes security parameter as an input and outputs a key $\mathsf{K}_{\text{BSTORE}}$ (typically a collection of keys for the various cryptographic primitives used). Note that $\mathsf{K}_{\text{BSTORE}}$, which the client is required to retain throughout the lifetime of the system, is required to be independent of the data to be stored.

- BSTORE.Build takes as input $(\mathsf{K}_{\text{BSTORE}}, d_0, \{\mathsf{id}_i, \mathsf{data}_i\}_{i=1}^{t}$, where $\mathsf{K}_{\text{BSTORE}}$ is a key, d_0 is an upperbound on the total number of data blocks to be stored in the system, $(\mathsf{id}_i, \mathsf{data}_i)$ are the id and data of the files that the system to be initialized with; it outputs an array of blocks D to be uploaded to the server.

- BSTORE.Access takes as input a key $\mathsf{K}_{\text{BSTORE}}$, a file id id, an operation specifier $\mathsf{op} \in \{\mathsf{read}, \mathsf{write}, \mathsf{update}, \mathsf{delete}\}$, and optionally data data (if op is write or update). Then it interacts with the server (through the upload/ download interface) and returns a status message and optionally file data (for the read and update operations). For the update operation, BSTORE.Access allows more flexibility:[2] first it requires only id as input, and outputs the current size of the file with that ID; then it accepts as input (an upperbound on) what the size of the file will be after update; then it outputs the current file data, and only then requires the new data with which the file will be updated.

Security Requirement. We specify the security requirement of a blind-storage system following the "real/ideal" paradigm that is standard for secure multi-party computation (as opposed to using specific game-based security definitions used in some of the earlier literature on SSE). This includes specifying an adversary model and an "ideal functionality," as detailed below. The formal security requirement we shall require is that of Universally Composable security [86] (but restricted to our adversary model).[3]

[2] One can always use a read followed by a write to get the effect of an update, but this is less efficient and potentially reveals more information.

[3] We remark that for our setting of passive adversaries, UC security is a conceptually simpler notion than for the setting of active adversaries. Nevertheless, for the sake of concreteness, we use the UC security model, which automatically ensures security even when the inputs to the client are adaptively chosen under adversarial influence.

In the adversary model we consider, the adversary is allowed to corrupt only the server *passively* — i.e., as an honest-but-curious adversary. (If the client is corrupt, we need not provide any security guarantees.)

The ideal functionality is specified as a virtual trusted third party $\mathcal{F}_{\text{STORE}}$ that mediates between the client and the server (modeling the information leaked to the server). $\mathcal{F}_{\text{STORE}}$ accepts two commands from the client: $\mathcal{F}_{\text{STORE}}$.Build and $\mathcal{F}_{\text{STORE}}$.Access, along with inputs to these commands (which are identical to the inputs to BSTORE.Build and BSTORE.Access as described above, except for the key $\mathsf{K}_{\text{BSTORE}}$). In this ideal model, it is $\mathcal{F}_{\text{STORE}}$ which maintains the collection of files, and performs all the operations specified by the $\mathcal{F}_{\text{STORE}}$.Build and $\mathcal{F}_{\text{STORE}}$.Access commands. In addition, it reveals limited information to the server as specified in Figure 4.2.

We stress that *all the information revealed to the server by our blind-storage scheme is captured by the $\mathcal{F}_{\text{STORE}}$ functionality.* Note that the information leaked (during $\mathcal{F}_{\text{STORE}}$.Build and $\mathcal{F}_{\text{STORE}}$.Access) is limited and simple to specify. This simplicity is one of the important contributions of this work.

Remark. Even when using the ideal $\mathcal{F}_{\text{STORE}}$ functionality, an adversary can learn some statistics about the files and accesses by analyzing the patterns in the information revealed to it. Such information could indeed be sensitive, and it is up to the higher-level application that uses a blind-storage system to ensure that this is not the case. The cryptographic construction seeks to only match the guarantees given by $\mathcal{F}_{\text{STORE}}$.

4.2.2 Our Construction

Our Blind Storage construction is called SCATTERSTORE. First, we shall present a simplified version, called SCATTERSTORE-LITE, which already involves most of the critical components in the full construction. The only drawback of the simplified construction is that the client is required to maintain a data structure to map each file-name to a small piece of information. This solution is well-suited for a scenario when the system consists of a moderate number of large files. In our final construction, we show how to avoid this local data structure, so that the client's storage is of constant size, independent of the number of files in the system.

The construction relies on the following primitives:

- a full-domain collision resistant hash function (CRHF), H,
- a pseudorandom function (PRF), Φ,
- a *full-domain* pseudorandom function (FD-PRF), Ψ (implemented by applying Φ to the output of H),
- a pseudorandom generator (PRG), Γ.

(In our prototype, as described in Section 4.4, the implementation of Φ, Ψ and Γ all rely on the AES block-cipher; H is implemented using SHA-256.) The security parameter k is an implicit input to all the cryptographic primitives used in the construction. The other parameters in the construction are the size parameters n_{D}, m_{D}, an expansion parameter $\alpha > 1$, and the minimum number of blocks communicated in each transaction, κ.

- KeyGen: A key K_{Φ} for the PRF Φ, and a key K_{ID} for the FD-PRF Ψ are generated; $\mathsf{K}_{\textsc{bstore}}$ is set to be the pair $(\mathsf{K}_{\Phi}, \mathsf{K}_{\mathrm{ID}})$.
- $\textsc{bstore}.\mathsf{Build}(\mathbf{F}, \mathsf{K}_{\textsc{bstore}})$: $\mathbf{F}$ is a list of files $\mathbf{f} = (\mathsf{id}_{\mathbf{f}}, \mathsf{data}_{\mathbf{f}})$. Below $\mathsf{size}_{\mathbf{f}}$ denotes the number of blocks in an encoding of $\mathsf{data}_{\mathbf{f}}$; each block has two short header fields containing a *version number* initialized to 0, and $H(\mathsf{id}_{\mathbf{f}})$; the latter is not allowed to be all 0s, which is reserved to indicate a free block. In addition, the first block has a header field that records $\mathsf{size}_{\mathbf{f}}$. (It will be convenient to keep the version number field at an extreme end of the block, as it needs to kept unencrypted, whereas the rest of the block will be encrypted at the end of this phase.)
  - Let D be an array of n_{D} blocks of m_{D} bits each.
  - Initialize every block in D with all 0s (to be encrypted later).
  - For each file $\mathbf{f}$ in $\mathbf{F}$,
    1. Generate a pseudorandom subset $S_{\mathbf{f}} \subseteq [n_{\mathsf{D}}]$, of size $|S_{\mathbf{f}}| = \max(\lceil \alpha \cdot \mathsf{size}_{\mathbf{f}} \rceil, \kappa)$ as follows.
       (a) Generate a seed $\sigma_{\mathbf{f}} = \Psi_{\mathsf{K}_{\mathrm{ID}}}(\mathsf{id}_{\mathbf{f}})$ for the PRG Γ.
       (b) Let $S_{\mathbf{f}}$ be the set of integers in the sequence $\Lambda[\sigma_{\mathbf{f}}, |S_{\mathbf{f}}|]$. Here $\Lambda[\sigma, \ell]$ denotes a sequence of ℓ integers obtained as follows.
          * Generate a (sufficiently long) output from the PRG Γ, with seed σ, and parse it as a sequence of integers in the range $[n_{\mathsf{D}}]$.
          * $\Lambda[\sigma, \ell]$ is the first ℓ distinct integers in this sequence.
    2. Check if the following two conditions hold:
       * at least $\mathsf{size}_{\mathbf{f}}$ blocks in D that are indexed by the numbers in $S_{\mathbf{f}}$ are free;
       * at least one block in D that is indexed by the numbers in $S^0_{\mathbf{f}}$ is free.

       If either condition does not hold, *abort*. By the choice of our parameters, this will happen only with negligible probability.
    3. Pick a pseudorandom subset $\widehat{S}_{\mathbf{f}} \subseteq S_{\mathbf{f}}$ of size $|\widehat{S}_{\mathbf{f}}| = \mathsf{size}_{\mathbf{f}}$, such that the blocks in D that are indexed by the numbers in $\widehat{S}_{\mathbf{f}}$ are all free. For convenience, we shall rely on the fact that the numbers in the sequence used to generate $S_{\mathbf{f}}$ are in a pseudorandom order; we pick the shortest prefix of this sequence that contains $\mathsf{size}_{\mathbf{f}}$ numbers indexing free blocks, and let $\widehat{S}_{\mathbf{f}}$ be the set of these $\mathsf{size}_{\mathbf{f}}$ numbers.
    4. Write the $\mathsf{size}_{\mathbf{f}}$ blocks of $\mathsf{data}_{\mathbf{f}}$ onto the blocks in D that are indexed by the numbers in $\widehat{S}_{\mathbf{f}}$ (in increasing order). These blocks get marked as not free.
  - Encrypt each block of D using the PRF Φ and the key K_{Φ}. The version number field is left unencrypted, while the rest is encrypted using the version number (initialized to 0) and the index number of the block as IV. More precisely, for the i^{th} block $\mathsf{D}[i]$, we split it as $v_i || \mathsf{B}[i]$ (v_i being the version number), and then update $\mathsf{B}[i]$ to $\mathsf{B}[i] \oplus \Phi_{\mathsf{K}_{\Phi}}(v_i || i)$.

    (If the block-size of the PRF is less than the size of the block $\mathsf{B}[i]$, then a few lower-order bits of the IV are reserved for use as a counter, to obtain multiple blocks from the PRF for a single block in D.)

Figure 4.3: SCATTERSTORE: A Blind-Storage Scheme (continued on next page)

- BSTORE.Access($\mathsf{id_f}, \mathsf{op}, \mathsf{K}_{\text{BSTORE}}$): We describe the case when $\mathsf{op} = \mathsf{update}$, and mention how the other operations differ from it.

    1. First, compute $\sigma_{\mathbf{f}} = \Psi_{\mathsf{K}_{\text{ID}}}(\mathsf{id_f})$, and define the set $S^0_{\mathbf{f}}$ of size κ, to consist of the numbers in the sequence $\Lambda[\sigma_{\mathbf{f}}, \kappa]$. Retrieve the blocks indexed by $S^0_{\mathbf{f}}$ from D.
    2. Decrypt the blocks of $\mathsf{D}[S^0_{\mathbf{f}}]$ (where $\mathsf{D}[i] = (v_i || \mathsf{B}[i])$ is decrypted as $\mathsf{B}[i] \oplus \Phi_{\mathsf{K}_\Phi}(v_i||i)$), in the order in which they appear in $\Lambda[\sigma_{\mathbf{f}}, \kappa]$, until a block which is marked as belonging to $\mathsf{id_f}$ is encountered. If no such block is encountered the file is not present in the system. In this case, set $\mathsf{size_f} = 0$.
    3. Otherwise (if a block marked as belonging to $\mathsf{id_f}$ is found), this is the first block of the file with identifier $\mathsf{id_f}$: recover the size of the file $\mathsf{size_f}$ from the header of this block.
    4. Output $\mathsf{size_f}$ to the client and accept as input $\mathsf{size'_f}$, the size of the file after update.
    5. Let $\ell = \lceil \alpha \cdot \max(\mathsf{size_f}, \mathsf{size'_f}) \rceil$. If $\ell \leq \kappa$, let $S_{\mathbf{f}} = S^0_{\mathbf{f}}$. Else, let $S_{\mathbf{f}}$ be the set of numbers in $\Lambda[\sigma_{\mathbf{f}}, \ell]$. In this case, retrieve the blocks indexed by $S_{\mathbf{f}} \setminus S^0_{\mathbf{f}}$ from the server.
    6. Some of the blocks indexed by $S_{\mathbf{f}}$ would have already been decrypted in Step 2 above. Decrypt the remaining blocks indexed by $S_{\mathbf{f}}$, as well.
    7. Identify $\widehat{S}_{\mathbf{f}}$ as the set of indices of blocks belonging to the file being accessed (by checking if their headers match $\mathsf{id_f}$). If $\widehat{S}_{\mathbf{f}}$ is not empty, combine these blocks together (in increasing order of their indices) to recover the entire contents of the file, and output it.
    8. Accept as input new contents data' encoded as $\mathsf{size'_f}$ blocks.
    9. Identify a subset $\widehat{S}'_{\mathbf{f}} \subseteq S_{\mathbf{f}}$ of size $\mathsf{size'_f}$ as follows. Find the shortest prefix of the sequence $\Lambda[\sigma_{\mathbf{f}}, \ell]$ which contains $\mathsf{size'_f}$ blocks that are either marked as belonging to $\mathsf{id_f}$ (i.e., in $\widehat{S}_{\mathbf{f}}$) or are free.

       If no such prefix exists, or if the first of the $\mathsf{size'_f}$ blocks identified is not within $\Lambda[\sigma_{\mathbf{f}}, \kappa]$ (this can happen only when $\mathsf{size_f} = 0$), then *abort*; again, by the choice of our parameters, this will happen only with negligible probability.

       Note that if $\mathsf{size'_f} < \mathsf{size_f}$, then $\widehat{S}'_{\mathbf{f}} \subseteq \widehat{S}_{\mathbf{f}}$; else, $\widehat{S}_{\mathbf{f}} \subseteq \widehat{S}'_{\mathbf{f}} \subseteq S_{\mathbf{f}}$.
    10. Then update the blocks indexed by $\widehat{S}'_{\mathbf{f}}$ with the blocks of data'. If $\mathsf{size'_f} < \mathsf{size_f}$ mark as free the blocks indexed by $\widehat{S}_{\mathbf{f}} \setminus \widehat{S}'_{\mathbf{f}}$.
    11. Encrypt all the blocks indexed by $S_{\mathbf{f}}$ using the IV $v_i||i$ as described in the BSTORE.Build step, but after incrementing v_i for each block.
    12. Upload the newly reencrypted blocks back to the server. Note that all the blocks that were downloaded, i.e., $\mathsf{D}[S_{\mathbf{f}}]$, are uploaded back, with their version numbers incremented by 1, and reencrypted.

- When $\mathsf{op} = \mathsf{read}$, the Steps 1 through 7 from above are carried out, but setting $\mathsf{size'_f} = 0$.
- When $\mathsf{op} = \mathsf{write}$, the behavior is the same as when $\mathsf{op} = \mathsf{update}$, except that the new file data is taken as input upfront, and no data is returned.
- When $\mathsf{op} = \mathsf{delete}$, the behavior is the same as when $\mathsf{op} = \mathsf{write}$, except that it takes $\mathsf{size'_f} = 0$.

Figure 4.4: SCATTERSTORE: A Blind-Storage Scheme (continued from Figure 4.3)

Simplified Construction: SCATTERSTORE-LITE

In this section, we present a sketch of SCATTERSTORE-LITE, our simplified Blind-Storage construction. We defer a formal description to the next section where we present the full construction.

In our construction, each file in the blind storage system is kept in a large array D of encrypted blocks, at positions indexed by a *pseudorandom set*. This set is defined by a short seed and the size of the set: the seed can be used to generate a

(virtually infinite) pseudorandom sequence, and the size specifies the length of the prefix of this sequence that defines $S_{\mathbf{f}}$. In our simplified construction, the client stores this information in a data-structure that maps the file-names to the descriptor of the pseudorandom set.[4]

The main security property that we need to ensure is that the location of the blocks of one file does not reveal any information about the blocks of the other files, or even the proportion of occupied and free blocks in D.[5] However, clearly, we cannot choose the positions to store blocks of one file independent of the blocks of the other files, since two files must not occupy the same block. A naïve solution would be to use a large D, to reduce the probability that the blocks chosen for one file do not overlap with that for any other file. But this is problematic, because to reduce the probability of such a collision to a small quantity (say, negligible in the security parameter), size of D needs to be enormously larger (i.e., a super-polynomial factor larger) than the actual amount of data stored.

We overcome this inherent tension between collision probability and wasted space as follows. To store a file $\mathbf{f}$of n blocks, we choose a pseudorandom subset $S_{\mathbf{f}}$ of not n blocks, but say (for a typical setting of parameters), $2n$ blocks. This subset of $2n$ blocks will be chosen independent of the other files in the system (and it is this subset that the server sees when the client accesses this file). Within this set we choose a subset $\widehat{S}_{\mathbf{f}} \subseteq S_{\mathbf{f}}$ of n blocks, where the actual data is stored. The set $\widehat{S}_{\mathbf{f}}$ is of course, selected depending on the other blocks used by other files, to avoid collisions. However, since the contents of the blocks are kept encrypted, the server does not learn anything about $\widehat{S}_{\mathbf{f}}$ (except its size).

This, it turns out, allows D to be only a small constant factor larger than the total data to be stored in the system. A typical parameter setting would be to let D have 4 times as many blocks as total data blocks to be stored. Then, we can drive the information leaked to the server to a negligible quantity with only small constant factor overheads in the storage and communication.[6]

[4]Only the *size* of the pseudorandom set needs to be stored. The seed for the set can be derived by applying a (full-domain) pseudorandom function to the file-name. See next section.

[5]This property manifests itself in the simulation based proof of security, since the simulator will pick the locations of blocks of a file being accessed independent of the number and size of files that are not yet accessed.

[6]We note that the pseudorandom set $S_{\mathbf{f}}$ would have to be at least a minimum size κ (say, $\kappa = 80$ blocks); when accessing small files which are just a few blocks long (i.e., n is small), $2n$ will be less than this minimum. For such files, the *communication* involves an additive overhead equal to this minimum. However, the number of blocks occupied in D, i.e., size of $\widehat{S}_{\mathbf{f}}$, is always n, irrespective of n being small or large.

An important feature of our pseudorandom set construction, compared to linked-list based construction of related work in the literature, is that the server need not carry out any decryptions. In linked-list based constructions, each node in the list is progressively revealed; even if the server were to take the help of the client in decrypting each node, several rounds of communication will be required.[7] In contrast, our construction allows the server to be "crypto-free" and still have only constant number of rounds of interaction.

This simplicity leads to another advantage: our construction meets a fully adaptive security definition for blind storage (and for searchable encryption) against honest-but-curious servers. Here, adaptive security refers to the fact that the choice of which files the client needs to access can be adversarially influenced, after the system has been deployed. Prior work required less efficient and more complicated schemes to achieve adaptive security, and often employed the Random Oracle heuristics [57, 53]. We use only standard primitives (PRFs and collision-resistant hash functions) and obtain security in the standard model (i.e., not in the Random Oracle model).

Finally, our pseudorandom set construction easily supports a *dynamic* blind-storage scheme. We sketch the update operation (creating and deleting a file are essentially special cases of the update operation). To update a file, the client retrieves the encrypted blocks corresponding to the file's pseudorandom set $S_{\mathbf{f}}$, decrypts them, updates the subset of blocks $\widehat{S}_{\mathbf{f}}$ where the file's blocks are present, reencrypts all of the downloaded blocks (i.e., all of $S_{\mathbf{f}}$), and uploads them back to the server. There are two details worth highlighting:

- Encryption of each block is carried out by XOR-ing the contents of the block with the output of a PRF, which keyed using a fixed secret key, but whose inputs depend on the block: this input consists of the block's index in D and its current *version number*. (The version number is specific to each block, and it is kept unencrypted in the block.) Initially, all blocks have version number 0, and when reencrypting a block, its version number is incremented.

- In the above process, the updated file may need fewer or more blocks than the original file. We let the size of the set $S_{\mathbf{f}}$ that is retrieved to correspond to the longer of the two versions of the file. If the updated version needs fewer

[7] In our full construction, the server does take the help of the client to carry out a decryption and to recover the description of the pseudorandom set; but this involves only one round of communication.

blocks than the original file (in which case $S_{\mathbf{f}}$ corresponds to the original file), the extra blocks are marked as free. If the updated file needs more blocks, then the subset $S_{\mathbf{f}}$ that is retrieved corresponds to the size of the file after the update; then, additional empty blocks are located in $S_{\mathbf{f}}$ to extend $\widehat{S}_{\mathbf{f}}$ to be large enough for the updated file. In either case, the server sees the size of the larger of the two versions.

Full Construction: SCATTERSTORE

We present the details of our final Blind Storage scheme in Figure 4.3 and Figure 4.4. Here we give a brief sketch of the main ideas.

In the simpler scheme above, we allowed the client to maintain a data-structure mapping a file-identifier $\mathsf{id}_{\mathbf{f}}$ to a descriptor of the pseudorandom set $S_{\mathbf{f}}$. This is not desirable if the system would store a large number of small files; then the size of this data structure is comparable to that of the entire collection of files. We would like our client to store only a constant number of cryptographic keys, so that its storage requirement does not grow with the size of the entire set of files stored in the system.

For this, recall that the two pieces of information needed to define a pseudorandom set $S_{\mathbf{f}}$ are a seed and the size of the set. The seed itself can be obtained by applying a full-domain PRF to the file-identifier. (A full-domain PRF can be implemented using a full-domain collision resistant function (CRHF) and a normal PRF: an arbitrary-length file-identifier is first hashed to a fixed-length input for the PRF using the CRHF.) If the client knew the size of $S_{\mathbf{f}}$ as well, there will be no need to store this map at all. We exploit this to use a two-level access to a file, as follows.

For each file, the first block consists of a header that stores the size (number of blocks) of the file. To retrieve a file $\mathsf{id}_{\mathbf{f}}$, the client assumes that the file is "small" and retrieves a pseudorandom set $S^0_{\mathbf{f}}$ with the smallest possible number of blocks, i.e., κ. After recovering the first block of the file from the blocks in $S^0_{\mathbf{f}}$, the client computes the actual size of $S_{\mathbf{f}}$ and if it is larger than κ, then retrieves the rest of $S_{\mathbf{f}}$ from the server. (Note that $S_{\mathbf{f}}$ is simply a superset of $S^0_{\mathbf{f}}$, obtained from a longer pseudorandom sequence.) We remark that it is important for security that when $|S_{\mathbf{f}}| > \kappa$ the client performs this second access, even if the entire file happened to fit within the blocks in $S^0_{\mathbf{f}}$.

The update functionality as we have defined, fits well into this two-level access. To update a file id_f, first the client is allowed to learn the current size of the file before providing any information about the update; this size information is retrieved after the first level of access and returned to the client. (Note that we could have in fact provided the client with the first few blocks of the current file too, but for simplicity we omit this from the specification of the functionality.) Next, before the second level of access, the larger of the current file size and updated file size needs to be known. So at this point, we require the client to submit the size of the updated file. Then the size of the set S_f to be retrieved is defined by the larger of the current and updated sizes. If this set has more than κ blocks, the second level of access retrieves the remaining blocks; then, as in the simpler construction, all the retrieved blocks will be reencrypted (with a subset of them having updated contents) and uploaded back on the server.

Variations and Enhancements

There are several optimizations and variations to this construction that would be of interest. We mention a few.

- The time taken for the read operation can be significantly improved as follows. As presented above, in reading file, the client retrieves a pseudorandom subset of blocks from the server, and *decrypts all of them*. Of these, the blocks that actually contain data from this file are identified from each block's header. Since decryption is the most computationally intensive operation, if we can avoid decrypting the blocks not belonging to the file being read, we can speed up the operation by a constant factor (namely, α, a parameter discussed later). This is indeed possible by storing the relevant information in the first block of the file. Note that we still need to sequentially decrypt a few blocks (for our choice of parameters, up to four blocks, in expectation) before the first block of the file is encountered.

- Almost all our operations — especially the computationally intensive parts involving encryption and decryption — are "embarrassingly parallel." For instance, a set of blocks received from the server can be decrypted in parallel and assembled together using an array pre-allocated to hold all the blocks in the file.

- Our construction can be easily extended to meet a stronger security requirement, that the server does not learn the *kind of operation* (read, write or update) performed by the client (beyond what it can infer from the access pattern). For this, we shall use the update operation in place of every operation, since it offers the facility of reading and writing. (If this is used for actual updates — which allow read and write in the same operation — and if the data being written depends on the date being read, then care should be taken to avoid observable delays that can lead to a timing attack.)

4.2.3 Security Analysis

We sketch a proof of security that our construction is a secure realization of the deal blind storage functionality $\mathcal{F}_{\text{STORE}}$, for the adversary model in which the server is corrupted only passively. The proof follows the standard real/ideal paradigm in cryptography (see [87], for instance), and uses some of the standard conventions and terminology.

Roughly, the proof involves demonstrating a simulator Sim which interacts with a client only via the ideal functionality $\mathcal{F}_{\text{STORE}}$ (the ideal experiment), yet can simulate the view of the server in an actual interaction with the client in an instance of our scheme (the real experiment). The simulated view would be indistinguishable from the real view of the server, even when combined with the inputs to the client. Further — and this is the adaptive nature of our security guarantee — the inputs to the client at any point in either experiment can be arbitrarily influenced by the view of the server till then.

Before describing our simulator, we describe the main reason for security. Suppose the client makes a read access to a file $\mathbf{f}$for the first time. In the *ideal experiment*, the server learns this file's size from $\mathcal{F}_{\text{STORE}}$, and nothing about the other files. In the real experiment, the server sees one or two downloads from D— a set of κ blocks $S^0_{\mathbf{f}}$ and a set of blocks $S_{\mathbf{f}} \setminus S^0_{\mathbf{f}}$ (with the possibility that $S_{\mathbf{f}} = S^0_{\mathbf{f}}$, in which case there is only one download). Thanks to the encryption, it is easy to enforce that the *contents* of these downloaded blocks give virtually no information to the server (beyond the size of $\mathbf{f}$). But we need to ensure that the *location* of these blocks also do not reveal anything more. For instance, it should not reveal how many other files are present in the system. In our construction, this is ensured by the fact that the pseudorandom subsets $S^0_{\mathbf{f}}$ and $S_{\mathbf{f}}$ are *determined by a process*

The simulator Sim interacts with the functionality $\mathcal{F}_{\text{STORE}}$ on the one hand, and interacts with the server on the other, translating each message it receives from $\mathcal{F}_{\text{STORE}}$ into a set of simulated messages in the interaction between the client and the server in our scheme.

1. When it receives the initial message from $\mathcal{F}_{\text{STORE}}$ with the system parameters, Sim can calculate the size of D; it simulates the contents of the blocks in D by picking uniformly random bit strings, with the version number in each block set to 0.

2. Sim initializes a map with entries of the form $(j; \Lambda_j, \mathsf{size}_j)$, which maps an integer j (indicating the sequence number of accesses) to a sequence of blocks in D and the size of the file accessed (in blocks).

   The maps are initialized to be empty, and is filled up as $\mathcal{F}_{\text{STORE}}$ reports file accesses to Sim.

3. For access number j^*, first the table entry $(j^*; \Lambda_{j^*}, \mathsf{size}_{j^*})$ is created as described below.

   Let the triple reported by $\mathcal{F}_{\text{STORE}}$ to Sim for access number j^* be $(\mathsf{op}, j, \mathsf{size})$. Recall that if $j > 0$, then the file being accessed has already been accessed (as the j^{th} access).

   (a) If $\mathsf{op} = \mathsf{delete}$, then let $\mathsf{size}_{j^*} = 0$. Else, set $\mathsf{size}_{j^*} = \mathsf{size}$. Let $\ell = \max(\lceil \alpha \cdot \mathsf{size}_{j^*} \rceil, \kappa)$.

   (b) If $j = 0$, then Sim samples a random sequence of ℓ distinct integers in the range $[n_{\mathsf{D}}]$, uniformly randomly, and sets Λ_{j^*} to be this sequence.

   (c) Else ($j > 0$), if $|\Lambda_j| \geq \ell$, set $\Lambda_{j^*} = \Lambda_j$; else ($j > 0$, and $|\Lambda_j| < \ell$), extend Λ_j to a sequence of length ℓ uniformly at random (without duplicates). Set Λ_{j^*} to be this extended sequence.

4. Next, Sim creates the simulated view in which first the server gets a request to download κ blocks indexed by the first κ entries of Λ_{j^*}; if $\ell > \kappa$, this is followed by a request to download blocks indexed by the next $\ell - \kappa$ entries of Λ_{j^*}. For operations other than read, this is followed by an upload consisting of new versions (with the blocks' version numbers incremented, and with fresh random strings as contents) of the blocks indexed by the first ℓ entries of Λ_{j^*}.

Figure 4.5: Description of the simulator Sim used in the proof of Theorem 4.2.3.1.

that is independent of the other files in the system – they are chosen randomly (or rather, pseudorandomly) for each file independently. The other files in the system influence the subset $\widehat{S}_{\mathbf{f}} \subseteq S_{\mathbf{f}}$ of blocks that actually carries the data (because these blocks must not be shared with the data-carrying blocks of any other file). However, due to the encryption, the server does not learn anything about $\widehat{S}_{\mathbf{f}}$ (beyond the fact that it must be a subset of $S_{\mathbf{f}}$).

Formally, a simulator can simulate the view of the adversary *randomly*, based only on the size of the file $\mathbf{f}$being accessed. The only difference between this simulation and the real execution (beyond what is hidden by the encryption and the security of pseudorandomness) is the following: in the real execution, there is a small probability that an update could fail, if there are not enough free blocks within the pseudorandom subset $S^0_{\mathbf{f}}$ or $S_{\mathbf{f}}$. In the simulation, no failure occurs. Thus the crucial argument in proving security is to show that it is only with negligible probability that the client would be left without adequate number of free blocks in such a pseudorandom set, forcing it to abort the protocol. We will give a standard probabilistic argument to prove that this is indeed the case.

In the proof below we describe our simulator Sim more formally, and then discuss the main combinatorial argument used to show that the simulation is indistinguishable from the real execution. For the sake of clarity, we leave out some of the routine details of this proof, and focus on aspects specific to our construction.

The following theorem statement is in terms of the "*storage slack ratio*" in a Blind Storage system, which is the ratio of the number of blocks n_{D} in the system to the number of blocks of (formatted) data in the files stored in the system. Note that the storage slack ratio decreases as files are added (or updated to become longer) and increases as files are deleted (or updated to become shorter). The security guarantee below uses the standard security definition in cryptography literature (see, for instance, [87]), which assures that the security "error" (statistical distance between the simulated execution and the real execution) is *negligible*,[8] as a function of the security parameter. Later, we discuss the choice of concrete parameters.

Theorem 4.2.3.1. *Protocol* SCATTERSTORE *securely realizes the functionality* $\mathcal{F}_{\text{STORE}}$ *against honest-but-curious adversaries, provided the storage slack ratio at*

[8] A function $\nu : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if, for every $c > 0$, there exists a sufficiently large $k_0 \in \mathbb{N}$ such that for all $k \geq k_0$, $\nu(k) < \frac{1}{k^c}$. That is, $\nu(k)$ becomes smaller than $1/\text{poly}(k)$ eventually, for any polynomial poly.

all times is at least $\frac{2}{1-1/\alpha}$ *and* $n_{\mathsf{D}} \geq \kappa = \omega(\log k)$.

Proof. The non-trivial case is when the server is corrupt (honest-but-curious) and the client is honest. We describe a simulator for this setting in Figure 4.5. The simulator essentially maintains the indices of the sets of blocks seen by the server. It need not maintain the subsets within these sets that carry actual data for the file being accessed. The maps are used to maintain consistency in terms of the pattern (same subsets are used if the same file is accessed) and the size of the files.

There are two differences between this simulation and the real execution. Firstly, the simulated execution uses truly random strings instead of the outputs from Φ, Φ and Γ. To handle this we can consider a "hybrid experiment" in which the real execution is modified so that instead of Φ, Ψ and Γ, truly random functions are used. By the security guarantees of the PRF, the FD-PRF and the PRG (applied one after the other), this causes only an indistinguishable difference.

The second difference is in aborting: in the real protocol, the client aborts when it cannot find enough free blocks in a pseudorandom subset, whereas the simulation never aborts. Conditioned on the protocol never aborting in the hybrid execution, the server's view in that execution is identical to that in the simulated execution.

To complete our proof, therefore it remains to show that the probability of the client aborting in the hybrid (or real) protocol is negligible. We denote this probability by p_{err}. Before proceeding, we remark that our goal here is to give an asymptotic proof of security (showing that p_{err} goes down as a negligible function of the security parameter). The concrete parameters from this analysis are overly pessimistic and an actual implementation can use less conservative parameters. The key message is that p_{err} provides a bound on the extent of insecurity, and this probability can be quickly driven down by modestly large parameters that scale linearly with the size of the data stored.

To analyze p_{err}, recall that we are analyzing a modified execution in which the output of the PRG Γ on pseudorandom seeds (used to define the pseudorandom subsets) have been replaced with truly random strings. Suppose there has been no abort so far, and a new file $\mathbf{f}$ of $\mathsf{size}_{\mathbf{f}}$ blocks is to be inserted into the system (either during the BSTORE.Build stage of during an update or write operation). Let d out of the n_{D} blocks in D be filled. These blocks were filled by picking random subsets, and then within these subsets, choosing random subsets with free blocks. The net effect is of choosing a random subset of d blocks out of the n_{D} blocks. Now, when $\mathbf{f}$ is being inserted, we pick a random subset $S^0_{\mathbf{f}}$ of size κ and a random

set $S_{\mathbf{f}} \supseteq S^0_{\mathbf{f}}$ of size $|S_{\mathbf{f}}| = \max(\lceil \alpha \cdot \mathsf{size}_{\mathbf{f}} \rceil, \kappa)$. The *expected number* of occupied blocks within this set is $\frac{d}{n_{\mathsf{D}}} \cdot |S_{\mathbf{f}}|$. By a standard application of Chernoff bound,[9] the probability that more than $2\frac{d}{n_{\mathsf{D}}}|S_{\mathbf{f}}|$ blocks are occupied is $2^{-\Omega(|S_{\mathbf{f}}|)}$, provided $\frac{d}{n_{\mathsf{D}}}$ is upperbounded by a constant less than 1. Since $|S_{\mathbf{f}}| \geq \kappa$, this probability is $2^{-\Omega(\kappa)}$, and since κ is super-logarithmic in k (for e.g., $\log^2 k$), this probability is $2^{-\omega(\log k)}$ which is negligible in k. Thus except with negligible probability, of the $|S_{\mathbf{f}}|$ blocks chosen, at least $|S_{\mathbf{f}}|(1 - 2\frac{d}{n_{\mathsf{D}}}) \geq \alpha \cdot \mathsf{size}_{\mathbf{f}} \cdot (1 - 2\frac{d}{n_{\mathsf{D}}})$ are free.

By the hypothesis in the theorem statement, the storage slack ratio $\frac{n_{\mathsf{D}}}{d} \geq \frac{2}{1-1/\alpha}$, or equivalently, $1 - 2\frac{d}{n_{\mathsf{D}}} \geq \frac{1}{\alpha}$. Thus, except with negligible probability, of the $|S_{\mathbf{f}}|$ blocks chosen $\alpha \cdot \mathsf{size}_{\mathbf{f}} \cdot (1 - 2\frac{d}{n_{\mathsf{D}}}) \geq \mathsf{size}_{\mathbf{f}}$ blocks are free. The same analysis shows that $S^0_{\mathbf{f}}$ will have at least one free block (in fact, at least $\lfloor \kappa/\alpha \rfloor$ free blocks), except with negligible probability. If both these conditions hold, the client will not abort when adding this file. By a union bound, the probability that it aborts remains negligible as long as it adds only polynomially many files. □

On the choice of parameters. There are a few parameters that one can set in an implementation of our blind storage scheme to optimize security levels and performance. For simplicity we treat p_{err} (which measures the probability that any *illegitimate* information is revealed to the server) as fixed at either 2^{-40} or 2^{-80}. The other important parameters are the following:

- γ, an upperbound on the storage slack ratio — i.e., $\frac{n_{\mathsf{D}}}{d_0}$, where d_0 is an upperbound on the total number of blocks of all the files (formatted correctly);
- α, the ratio between the number of blocks in a (large enough) file and the number of blocks in the pseudorandom subset which is downloaded/uploaded when that file is accessed; and
- κ, the minimum number of blocks in a pseudorandom subset.

The higher these parameters, the better the security level would be. However, they also reflect higher storage and communication costs. One can find different combinations of (γ, α, κ) to meet a security level (probability of "error" in simulation)

[9] In choosing a random subset of blocks, the blocks are not chosen independent of each other. So in order to apply Chernoff bound, we first consider the experiment in which the blocks are selected independent of each other with the same fixed probability, so that the expected number of blocks chosen is, say $3/2d$. Then, by an application of Chernoff bound, except with $2^{-\Omega(n_{\mathsf{D}})}$ probability, at least d blocks are occupied. Now, in this experiment, we bound the probability that more than $2\frac{d}{n_{\mathsf{D}}}|S_{\mathbf{f}}|$ blocks in $S_{\mathbf{f}}$, again using Chernoff bound. This probability is an upperbound on the corresponding probability in the original experiment.

using the following explicit upperbound, which is tighter than the Chernoff bound used for asymptotic analysis above.[10]

$$p_{\text{err}}(\gamma, \alpha, \kappa) \leq \max_{n \geq \frac{\kappa}{\alpha}} \sum_{i=0}^{n-1} \binom{\lceil \alpha n \rceil}{i} \left(\frac{\gamma - 1}{\gamma}\right)^{i} \left(\frac{1}{\gamma}\right)^{\lceil \alpha n \rceil - i}$$

Figure 4.6 plots various possible combinations of α and κ for various choices of p_{err} and γ. A few suggested choices of (γ, α, κ) which achieve $p_{\text{err}} \leq 2^{-40}$ are $(4, 4, 45)$, $(2, 8, 60)$ and $(4, 8, 25)$. Thus, for instance, one could use the parameter setting of $(\gamma, \alpha, \kappa) = (4, 4, 45)$ which means that the amortized storage requirement for each file and the communication requirement for reading *large files* is roughly 4 times the size of the file; however, for small files – any file with at most 11 blocks, including empty or non-existent files – 45 blocks would be downloaded, decrypted, reencrypted and uploaded back.

While a very large value of κ would require a large amount of communication and extra computation on part of the client (for updates), we recommend moderately large values for κ. This is because, firstly, *increasing κ does not have any effect on the storage needed* (because only as many blocks are occupied in D as the actual data consists of), and secondly it actually provides a higher security guarantee and may slightly increase the overall efficiency too! Apart from lowering p_{err}, another reason for a higher security guarantee (not captured in $\mathcal{F}_{\text{STORE}}$, for simplicity) is that the server does not learn the exact number of blocks in every file that is accessed; for "small" files, it learns only that the file is small (at most 11 blocks, in the above example). The higher the value of κ, the less the information that the server learns. The potential (slightly) higher efficiency is due to the fact that when a "small" file is retrieved, a single round of interaction suffices, and again, the higher the value of κ, the more the files that fall into the "small" category. This does not increase the computational cost during read operations.

We point out that while α, κ (and n_{D}) are parameters built into the system specification, it is not necessary to have a hard bound d_0 on the number of blocks of D that can be filled. In other words, γ and p_{err} exhibit graceful degradation: as the array D fills up and γ decreases, p_{err} increases.

Another parameter that affects the choice of these parameters is the size of the

[10]The error probability when adding a file of $n \geq \frac{\kappa}{\alpha}$ blocks is upperbounded by the probability that when $\lceil \alpha n \rceil$ blocks are picked (with replacements) from a set of n_{D} blocks of which at most d_0 would be occupied, $i < n$ distinct blocks that are picked are free. The actual experiment involves picking blocks without replacement, but for our range of parameters, this gives a valid upperbound.

blocks in D. As the block size decreases, on the one hand, the number of blocks in files grows and the effect of the communication overhead due to the minimum number of blocks used for small files (the parameter κ) decreases; on the other hand, the overhead due to the header size in each block increases.

An implementation can choose a default standard setting of the above parameters, or seek to optimize performance by tuning them to suit the profile of the files to be stored in the system. For instance, the set of parameters appropriate for an application like our SSE construction in the sequel (in which the keyword index files are stored in a Blind Storage system) may be different from those appropriate for an application storing a relatively small number of large files. But it is important that any such optimization is based on a *public* profile of the set of files to be stored in the system. This is because, conservatively, we should assume that the server would know all the system parameters (and exact sizes of the files accessed). It is true that, heuristically, slightly better guarantees may be available, since the server learns only $\max(\lceil \alpha \cdot \mathsf{size} \rceil, \kappa)$, and need not exactly know α and κ (except as revealed by the former, combined with any auxiliary information it may have about the sizes of the files being accessed). Further heuristics could be employed to make this information noisy, so that it remains hard to decipher the parameters even from a large number of correlated accesses. Nevertheless, we recommend that the system parameters are optimized only using information that can be made known to the server.

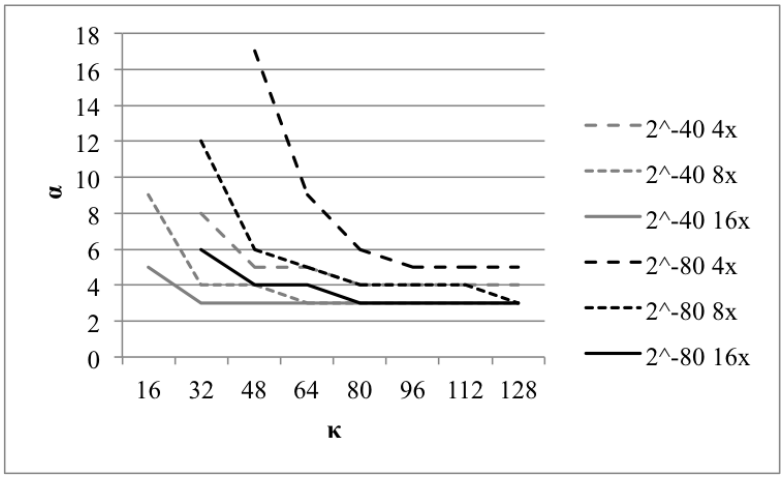


Figure 4.6: Finding the right parameters. Each line on the graph corresponds to trade-offs between α and κ for a choice of $p_{\text{err}} \in \{2^{-40}, 2^{-80}\}$ and $\gamma \in \{4, 8, 16\}$.

4.3 Searchable Symmetric Encryption

In this section we formally define the syntax and security requirements of a dynamic SSE scheme, and also present an efficient construction. As we shall see, our syntax for a dynamic SSE scheme is simpler than in [57], since all non-trivial operations are carried out by the client, and hence, there are no server side algorithms to be specified. Our construction

4.3.1 Definitions

A dynamic searchable symmetric encryption scheme (or simply, SSE) consists of five probabilistic polynomial time procedures (run by the client), SSE.keygen, SSE.indexgen, SSE.search, SSE.add and SSE.remove. These procedures interact with a "dumb" server which provides download and upload facilities to access blocks in an array (see Section 4.2.1), and also a simple file-system to lookup documents by identifiers. Looking ahead, in our implementation, the upload and download facilities are used to implement a blind-storage scheme which is used to store the keyword indices, and the file lookup facility is used to store the actual (encrypted) documents.[11]

- SSE.keygen: Takes the security parameter as input, and outputs a key K_{SSE}. All of the following procedures take K_{SSE} as an input.
- SSE.indexgen: Takes as input the collection of all the documents (labeled using document IDs), a dictionary of all the keywords, and for each keyword, an *index file* listing the document IDs in which that keyword is present.[12] It interacts with the server to create a representation of this data on the server side.[13]
- SSE.search: Takes as input a keyword w, interacts with the server, and returns all the documents containing w.

[11] In our scheme, if we opt to have document-set security, then the file lookup facility is not used, as the documents will also be stored in the blind-storage.

[12] The index files can be created by SSE.indexgen, if it is not given as input.

[13] Typically, this would consist of a collection of (encrypted) documents, labeled by document indices (different from document IDs), and a representation of the index, which in our constructions will be stored using a blind-storage system.

- *Initialization.* On receiving the command $\mathcal{F}_{\text{SSE}}.\mathsf{indexgen}$ from the client, $\mathcal{F}_{\text{SSE}}$ accepts a set of ∂_0 documents — refered to as the *original documents* (as opposed to newly added documents) — and stores them internally in an array Δ. For $1 \leq \partial \leq \partial_0$, the array stores $\Delta[\partial] = (\mathsf{id}_\partial, \mathsf{contents}_\partial, \mathsf{W}_\partial)$ — a unique document ID, the document contents and a set of keywords in the document. It also accepts from the client a number N, which is a (possibly liberal) upperbound on the total number of (keyword, document) pairs that will be present in the system at any one time. $\mathcal{F}_{\text{SSE}}$ also maintains a set called $\mathsf{Removed}$ initialized to $\emptyset$.

  - **Initialization Leakage.** $\mathcal{F}_{\text{SSE}}$ reveals to the server the $(\mathsf{N}, s_1, \cdots, s_{\partial_0})$ where $s_\partial = |\mathsf{contents}_\partial|$ (in number of bits).

- *Addition.* On receiving the command $\mathcal{F}_{\text{SSE}}.\mathsf{add}$ to add a document $(\mathsf{id}, \mathsf{contents}, \mathsf{W})$ (with a new or existing document ID), $\mathcal{F}_{\text{SSE}}$ appends it to the array Δ: i.e., if there are $\partial - 1$ entries currently, let $\Delta[\partial] = (\mathsf{id}, \mathsf{contents}, \mathsf{W})$. If there exists $\partial' < \partial$ with $\mathsf{id}_{\partial'} = \mathsf{id}$ and $\partial' \notin \mathsf{Removed}$, then add ∂' to $\mathsf{Removed}$.

  - **Addition Leakage.** $\mathcal{F}_{\text{SSE}}$ reveals to the server the updated set $\mathsf{Removed}$ and $\{M_w^{\mathsf{new}} | w \in \mathsf{W}\}$, where $M_w^{\mathsf{new}} = \{\partial' | \partial' > \partial_0 \text{ and } w \in \mathsf{W}_{\partial'}\}$. i.e., M_w^{new} is the set of *newly added documents* (i.e., not the original documents) that have the keyword w. Note that only the sets M_w^{new}, and not their labels w, are shared with the server.

- *Removal.* On receiving the command $\mathcal{F}_{\text{SSE}}.\mathsf{remove}$, $\mathcal{F}_{\text{SSE}}$ accepts a document ID id and identifies ∂ (if any) such that $\mathsf{id}_\partial = \mathsf{id}$ and $\partial \notin \mathsf{Removed}$. If such an index ∂ exists, it adds ∂ to $\mathsf{Removed}$.

  - **Removal Leakage.** $\mathcal{F}_{\text{SSE}}$ reveals to the server the updated set $\mathsf{Removed}$.

- *Search.* On receiving the command $\mathcal{F}_{\text{SSE}}.\mathsf{search}$, it accepts a keyword w from the client and returns $\{(\mathsf{id}_\partial, \mathsf{contents}_\partial) | w \in \mathsf{W}_\partial \text{ and } \partial \notin \mathsf{Removed}\}$ to the client.

  - **Search Leakage.** $\mathcal{F}_{\text{SSE}}$ reveals to the server the last instance the same keyword was searched on (or that it is being searched for the first time) and also $M_w = \{\partial | w \in \mathsf{W}_\partial\}$.

Figure 4.7: The $\mathcal{F}_{\text{SSE}}$ functionality: all the information leaked to the server in our SSE scheme is specified here.

- SSE.add: Takes as input a new document (labeled by a document ID that is currently not in the document collection), interacts with the server, and incorporates it into the document collection.

- SSE.remove: Takes as input a document ID, interacts with the server, and if a document with that ID is present in the server, removes it from the document collection.

Security Requirement. As in the case of blind-storage, we specify an ideal functionality, $\mathcal{F}_{\text{SSE}}$ (Figure 4.7) to capture the security requirements of a dynamic SSE scheme. We note that the standard simulation-based security (with an environment) applied to the functionality $\mathcal{F}_{\text{SSE}}$ automatically ensures what has been called security against adaptive chosen keyword attacks (CKA2-security) for searchable encryption.

The functionality $\mathcal{F}_{\text{SSE}}$ is described in detail in Figure 4.7. If document-set privacy is required, then the functionality behaves slightly differently: the original set of documents are not added to the list of documents Δ upfront, but each one is added only at the first instance when it is accessed using $\mathcal{F}_{\text{SSE}}$.search or $\mathcal{F}_{\text{SSE}}$.remove.

We highlight a few aspects of our security definition, compared to that in [57] and prior work. In all forms of SSE, *keyword access pattern* and *document access pattern* are revealed: i.e., if the same keyword is searched for multiple times or if the same document appears in multiple keyword searches, the server learns about that; techniques for hiding this information incur significant costs. The goal of an SSE scheme is to reveal as little information as possible, beyond this information. In our scheme we reveal very little information beyond this, for the *original set of documents*. For newly added documents, a little more information is revealed, as they are added (see *Addition leakage* in Figure 4.7). This has the effect that for every subset of newly added documents, the server learns only the *number of keywords that are common to all the documents in that subset*.

Existing schemes reveal significantly more information. For example, in [57], when a document is removed, the scheme reveals *the number of keywords in the document* and further, for each keyword in it, *up to two other documents that share the same keyword*. This is the case even if that keyword is never searched on. In contrast, by our security requirement, if an original document is removed, only the number of *keywords in it that are searched* can be revealed. Further, it is not

revealed that a removed document shared a keyword with another document, unless such a keyword is explicitly searched for.

We remark that our functionality reveals "removed" versions of the documents in search results, but this information was revealed (implicitly) by the leakage functions in [57] as well, as the identifiers for each keyword in a removed document is revealed and this information links the removed documents to future searches on the same keyword (when the same identifier for the keyword is revealed).

Finally, our scheme allows the client to refer to a document using an arbitrary document ID rather than a serial number (which is useful when removing documents from the collection). We also allow the client to reuse document IDs. The server does learn when a document ID is reused (though not the actual identifier of the document ID itself); further, in the pattern information revealed to the server, the different versions that use the same document ID are differentiated. Other dynamic SSE schemes often avoid this aspect simply by not using document IDs. This suffices if the only time a document is removed is immediately after retrieving it from a search (or if the client is willing to maintain a map from document IDs to serial numbers); however, realistically, in many applications of a dynamic SSE scheme, it will be important to efficiently remove documents referenced by their document IDs.

4.3.2 Searchable Encryption from Blind Storage

In this section, we describe an efficient dynamic searchable encryption scheme, BSTORE-SSE, built on top of a blind-storage scheme. The full details are given in Figure 4.8. Here we sketch the main ideas.

First, note that we can implement a *static* searchable encryption scheme simply by storing the index file for each keyword (which lists all the documents containing that keyword) in a blind storage system. The guarantees of blind storage readily translate to the security guarantees of searchable encryption: the server learns only the pattern of index files (i.e., keywords) accessed by the client.

In a dynamic searchable encryption scheme, we need to support adding and removing documents, which in turn results in changing the index files. We seek to do this without revealing much information about the keywords in a document being added or removed, if those keywords have not been searched on before. To support dynamic searchable encryption (with much better security guarantees

The construction uses a blind-storage system BSTORE, and a pseudorandom permutation Ψ' for mapping document IDs (with versioning) to pseudorandom document indices. It also uses a clear-storage system CLEARSTORE (see text).

- SSE.keygen: Let $\mathsf{K}_{\text{SSE}} = (\mathsf{K}_{\text{BSTORE}}, \mathsf{K}_{\partial\mathsf{ID}})$ where $\mathsf{K}_{\text{BSTORE}}$ is generated by KeyGen and $\mathsf{K}_{\partial\mathsf{ID}}$ is a key for the PRP Ψ'.

- SSE.indexgen:

  1. Firstly, for each document ∂, assign a pseudorandom ID $\eta^0_\partial = \Psi'_{\mathsf{K}_{\partial\mathsf{ID}}}(\mathsf{id}_\partial)$, where id_∂ is the document ID.
  2. For each keyword w that appears in at least one document, construct an *index file* with file-ID index^0_w that contains η^0_∂ for each document ∂ that contains the keyword w. No specific format is required for the data in this file; in particular, it could contain a "thumbnail" (of fixed size) about each document in the list.
  3. Next, initialize a blind-storage system with the collection of all these index files (using BSTORE.Build).
  4. Also, (outside of the blind-storage system) upload encryptions of all the documents labeled with their pseudorandom document index η^0_∂.

- SSE.remove: To minimize the amount of information leaked, and for efficiency purposes, we rely on a lazy delete strategy.

  1. Given a document ID id_∂, check if a document with index $\eta^0_\partial = \Psi'_{\mathsf{K}_{\partial\mathsf{ID}}}(\mathsf{id}_\partial)$ exists, and if so remove it, using the file system interface of the server. The index files (in the blind storage or the clear storage) are not updated for the keywords in this document right away, but only during a subsequent search operation (see below).

- SSE.add: To add a document ∂ to the document collection, first call SSE.remove to remove any earlier copy of a document with the same document ID. Then proceed as follows:

  1. Compute a pseudorandom document index $\eta^1_\partial = \Psi'_{\mathsf{K}_{\partial\mathsf{ID}}}(\mathsf{id}_\partial)$.
  2. Generate a random tag tag and add it to the document (say, as a prefix, before or after encrypting the document). Encrypt the document and upload it, as in the SSE.indexgen phase, using the label η^1_∂.
  3. Then, for each keyword w that appears in this document, use the append facility of the clear-storage scheme to append a record consisting of $(\eta^1_\partial, \mathsf{tag})$ to the file with file-ID index^1_w to include η^1_∂. Note that the append operation will create a file in the clear storage system, if it does not already exist.

- SSE.search: Given a keyword w, retrieve and update the index files with file-ID index^0_w and index^1_w as follows:

  1. Retrieve the index file index^0_w from the blind storage system using the first stage of update operation of the blind storage scheme. Also, retrieve the index file index^1_w from the clear storage system, using the first stage of its update operation. All the documents containing the keyword w have their document indices listed in these two index files. Attempt to retrieve all these documents listed from the server.
  2. Some of the documents listed in the index file index^0_w could have been removed. Complete the blind storage update operation on the file index^0_w to erase the removed files from its list, without changing the size of the file.
  3. Some of the documents listed in the index file index^1_w may have been removed or replaced with newer versions. Complete the clear storage update operation on the file index^1_w to remove from its list any document that could not be retrieved, or for which the listed tag did not match the one in the retrieved document. (Both the update operations are completed in the same round).

  One could add an extra round to first check just the tags of the documents before retrieving the documents themselves.

Figure 4.8: Searchable Encryption Scheme BSTORE-SSE

than previous constructions), we rely on the following observation. The access pattern that server would be allowed to learn tells the server if two newly added documents share a keyword or not, *as soon as they are added and before such a keyword is searched for* (but not whether they share keywords with the original set of documents that were added when initializing the system). This means we can treat the set of newly added documents virtually as a different system, with significantly weaker security requirements.

Thus, for each keyword, we use two index files: one listing the original documents that include that keyword, and another listing the newly added documents that include it. The first index file is stored with the server using a blind storage scheme, where as the second can be stored in a "clear storage" system (see below). Searching for keywords now involves retrieving both these index files. Adding documents involves updating only the second kind of index files (using an append operation of the clear storage). Also, removing a newly added document involves updating only the second kind of index files, which is straightforward (except for efficiency concerns, addressed below). But in removing an original document, we need to ensure that the information on keywords in it that are not searched for (for e.g., the number of such keywords) remains secret. This is achieved by a *lazy deletion* strategy. The index file of a keyword (for the original set of documents) is not updated until that keyword is searched for. At that point, if the client learns that a document listed in that index has been deleted, the index is updated accordingly. This update can be carried out in a single update operation of the blind storage scheme, with little overhead.

In fact, for removing newly added documents too, we follow a similar lazy delete strategy, for efficiency purposes. (Otherwise, during a delete operation, the client will need to fetch the index files for all the keywords in the deleted document in order to update them, unless the server is willing to carry out a small amount of computation.) However, we need to account for the possibility that a document ID could be reused and that a later version may not have a keyword present in an earlier version. We associate a random tag with a document to check if the version listed in an index is the same as the current version.

Properly instantiated, this simple idea yields strictly better security than prior dynamic searchable encryption schemes [57, 53] which revealed more information about keywords not searched for, especially when removing documents.

Clear Storage. To store the index files for newly added documents, our SSE

scheme uses a "clear storage" scheme CLEARSTORE that supports the following operations:

- Files labeled with file-IDs can be stored (in the clear, without any encryption). A two-stage update operation can be used to read this file and then write back an updated version (which could be shorter).
- In addition, there is an efficient append operation, that allows appending a record (of fixed size) to the file in constant time (without having to retrieve the entire file and update it).

Note that a standard file-system interface provided by the server can support all these operations. But the append operation may not be supported by a cloud storage provider. In this case, it can be implemented by the client, as we consider in our evaluation.

We consider a simplified version of the SCATTERSTORE scheme to implement CLEARSTORE with efficient append. In this implementation, to store a file $\mathbf{f} = (\mathsf{id}_\mathbf{f}, \mathsf{data}_\mathbf{f})$, the file data $\mathsf{data}_\mathbf{f}$ is stored (unencrypted) in a subset of blocks of a pseudorandom set $\widehat{S}_\mathbf{f} \subseteq S_\mathbf{f}$. We use a separate file-system interface (without append) to store fixed-size header files labeled with the file-name $\mathsf{id}_\mathbf{f}$; This header file stores an index indicating the *last block* of $S_\mathbf{f}$ that is occupied by the file (i.e., the length of the shortest prefix of $S_\mathbf{f}$ that contains $\widehat{S}_\mathbf{f}$[14] To append a record to a file, the client retrieves the header block via the file-system interface, using the file-name $\mathsf{id}_\mathbf{f}$. Then it generates $S_\mathbf{f}$, and recovers the i^{th} block in $S_\mathbf{f}$, where i is the index stored in the header block. Then it checks if there is enough space in this last block, and if so adds the record there. Else, it generates κ more entries in $S_\mathbf{f}$, fetches those blocks from the clear storage, adds the record to the first empty block in this sequence,[15] and updates the index of the last block stored in the header file accordingly. Note that the number of blocks fetched is a constant on average provided a block is large enough to contain (say) κ records; the number of blocks written back is at most two (and on the average, close to 1).

Choice of parameters. We instantiate BSTORE-SSE with our SCATTERSTORE constructions. By choosing the parameter κ for SCATTERSTORE, we can ensure

[14]The first block of the data could also be stored in the header file. Note that then it is possible that the header block itself contains all the data of the file; in this case the index indicating the last block is set to 0.

[15]Unlike in the case of blind-storage, if no empty block is found among the blocks fetched, the client can go on to fetch more blocks. This also allows one to optimistically fetch a smaller number of blocks, without a significant penalty.

that a single search operation can typically be completed in one and half rounds of interaction. This is because the typical size of an index file could fit into a few blocks, and by choosing $\kappa = 80$ as we do in our experiments, the index file can often be retrieved without having to fetch more blocks. However, in the worst case (e.g., searching for the keyword "the," as we report), two and half rounds of interaction will be needed.

Security Analysis

Theorem 4.3.2.1. *Protocol* BSTORE-SSE *securely realizes* $\mathcal{F}_{\text{SSE}}$ *against honest-but-curious adversaries.*

Proof Sketch: The security of this scheme is fairly straightforward to establish, since it uses the blind storage scheme as a blackbox, and involves no other cryptographic primitive. All the information available to the server from the blind storage scheme as used in this construction (i.e., the access patterns of the index files) is easily derived from the information that the server is allowed to have in the searchable encryption scheme. In other words, a simulator can simulate to the server all the messages in the protocol using the information it obtains in the ideal world. The details are straightforward, and hence omitted. □

4.4 Implementation Details

We implemented prototypes of our blind storage and searchable encryption schemes. The code was written in C++ using open-source libraries. We used Crypto++ [88] for the block cipher (AES) and collision-resistant hash function (SHA256) implementations.

As our schemes only require upload and download interface and do not require any computation to be performed on the server, they can be implemented on commercially available cloud storage services. As a proof of concept, we further implemented a C++ API to interface with Dropbox's Python API. This enables a Dropbox user to use a C++ implementation of BSTORE-SSE (using SCATTERSTORE) with Dropbox as the server. In our Dropbox implementation, each *block* in the SCATTERSTORE scheme is kept as a file in Dropbox. We recommend using SCATTERSTORE with a block size that is a multiple of the block size

in the cloud storage provider's storage (typically, 4KB).

4.5 Searchable Encryption Evaluation

For concreteness, we will compare the performance of our SSE scheme with that of the recent scheme in [57], as one of the most efficient dynamic SSE schemes in the literature, implemented in a comparable setting. The more recent work of [7] offers a possibly more optimized version of this protocol (without dynamic functionality), but is harder to compare against experimentally, as the reported implementation was in a high performance computing environment. We remark that for the case of simple keyword searches (which is not the focus of [7]), the construction of [7] is similar to that of [53, 57], and is expected to show similar performance.

We focus on computational costs; space and communication overheads in the prior constructions are often not reported making a direct comparison hard.

- The computation times reported are for the client. In our case the server is devoid of any computation (beyond simple storage tasks) and hence this constitutes all the computation in the system. In contrast, in previous SSE schemes, the server's computation is often much more than that of the client. Thus it would already be a significant improvement if our client computation costs are comparable to the client computation costs in prior work. As we shall see, this is indeed the case.

- There are several possible engineering optimizations in the Blind-Storage scheme which can significantly improve the performance of the SSE scheme (for instance, the first one listed in Section 4.2.2 cuts down the time taken for the search operation by a factor of α or more). *None of these optimizations* have been implemented in the prototype used for evaluation.

Datasets. We use two datasets to evaluate our searchable encryption scheme, *emails* and *documents*.
1) For emails we use the Enron dataset [89] which was also used by [57] and several other works. From the Enron e-mail dataset, we selected a 256MB subset, consisting of about 383,000 unique keywords and 20,695,000 unique (document,

keyword) pairs. In the experiments involving smaller amounts of data, subsets of appropriate sizes were derived from these datasets.
2) For documents, we created a dataset with 1GB of four types of documents, namely PDF, Microsoft PowerPoint, Microsoft Word and Microsoft Excel. The documents were obtained by searching for English language documents with file-types *pdf*, *ppt*, *doc* and *xls*, using Google search. The resulting collection consists of 1556 documents (roughly evenly distributed among the four filetypes), with over 214,000 unique keywords and about 1,372,000 unique (document, keyword) pairs.

Experiments. The code was compiled without any optimizations on Apple Mac OS X. We used a well provisioned laptop – with Intel Core i7 3615QM processor, 8GB memory, running Mac OS X 10.9 – for the experiments, keeping in mind that the typical user of our system will use searchable encryption on a cloud via her personal computer, just the same way a cloud storage service like Dropbox is used. This is in contrast with prior research which typically evaluated their work on large servers with large amounts of memory.

As we shall see below, our scheme is highly scalable and practically efficient. We cannot offer a direct comparison between our performance speeds and that of [57], because of different hardware configuration and limited test equipment information presented in [57]. Nevertheless, our evaluation shows that our scheme should be significantly more efficient than that of [57].

4.5.1 Micro-benchmarks – File-keyword pair analysis

In [57], micro-benchmarks were used to evaluate the SSE operations. We do the same for the SSE.indexgen algorithm. (For our search, add and delete operations, the performance is essentially independent of the total number of file-keyword pairs already stored in the system, and this micro-benchmark does not provide a meaningful evaluation of these operations. These operations are evaluated differently, as explained below.)

Figure 4.9 shows micro-benchmarks for our scheme. The parameters used in the scheme are held constant, and are the same as detailed in the next section. Each data point is an average of 5 runs of SSE.indexgen. Note that the amortized per-pair time falls as the number of pairs increases, before tending to 1.58μs; this is because our SSE.indexgen operation involves encrypting the whole array D (which has the same size in all the experiments), and this overhead does not increase with

the number of pairs.

Compared to the time for index generation operation reported in [57], our performance is significantly better. [57] reports a per-pair time of 35μs for the same operation. Thus our index generation operation is an order of magnitude faster.

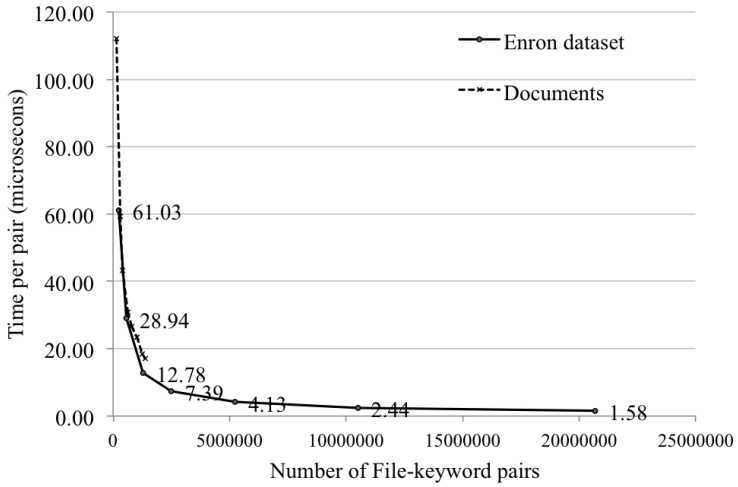


Figure 4.9: File/Keyword pair versus amortized time for SSE.indexgen. Time per file/keyword pair tends to 1.58μs, much better than the 35μs reported in [57] in a similar dataset.

4.5.2 Full evaluation

Each data point for Index Generation is the average of 5 runs of SSE.indexgen. Each data point for the Search is the average of 5 runs using the most frequent English word "the". Each data point for addition is the average of at least 5 runs.

Parameters Used

The parameters used for the experiments guarantee $p_{\text{err}} \leq 2^{-80}$ (recall that p_{err} is the probability of the scheme aborting and measures the security "error") if less than $1/8$ of the total blocks in D are filled and guarantee $p_{\text{err}} \leq 2^{-40}$ if less than $1/4$ of the total blocks in D are filled. We set $\kappa = 80$ and $\alpha = 4$, block size of D to 256 bytes, the total number of blocks in D to $n_{\mathsf{D}} = 2^{24}$.

Index Generation

Index generation is computationally the most expensive phase of any searchable encryption scheme. Our index generation performance measurements include encryption of documents and all other operations except the cost of plaintext index generation. Plaintext index generation performance is orthogonal to our contributions, doesn't reflect the performance of our system and is ignored by all prior work. Figure 4.10 shows our index generation performance on the email dataset. Our performance is much better when compared to that of [57], which takes 52 seconds to process 16MB of data. Our scheme can process 256MB (16 times more data) in about 35s. [57] extrapolates this to to 16GB of text e-mails without any attachments and, since the time for index generation scales roughly linearly with data, estimates that their index generation would take 15 hours; in contrast, it would take only 41 minutes in our scheme.

This matches our conclusion from the micro-benchmarks evaluation, that our index generation operation is at least an order of magnitude faster than that of [57].

Figure 4.11 shows the performance of our scheme on the document dataset.

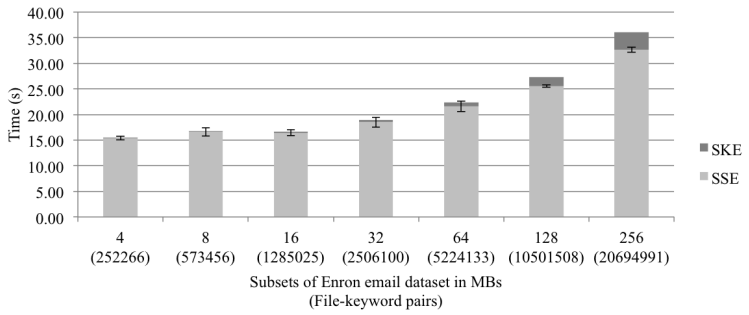


Figure 4.10: SSE.indexgen performance on email dataset with 99% confidence intervals: SKE stands for Symmetric Key Encryption and is the time required to encrypt the documents. All SKE costs are non-zero but some are very small.

Communication costs. The communication cost of initial index upload depends upon the parameters used for Blind-Storage, and specifically, the size of the array D. As mentioned above, in Section 4.5.2, the size of D was set to 1GB (2^{24} blocks of 256 bytes each) in our experiments. In comparison, the actual amount of index data for the 256MB subset of the email dataset consisted of 20,694,991 file-keyword pairs, which, using 4-byte fields for document IDs, translates to about 78MB of data. Given the small size of some of the index files, on formatting this data into 256-byte blocks for the Blind-Storage scheme, this resulted in about 178MB data.

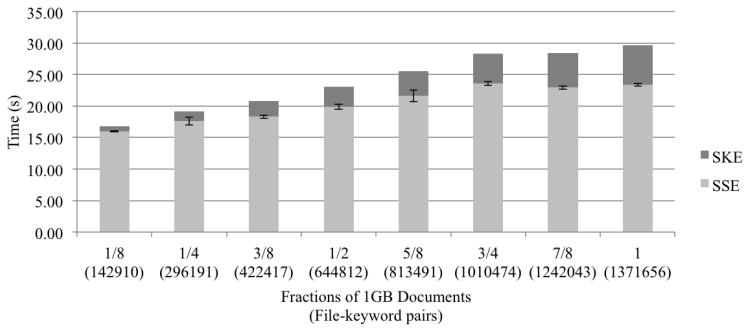


Figure 4.11: SSE.indexgen on the document dataset with 99% confidence intervals

For our choice of κ and α, $\gamma = 4$ is sufficient to bring p_{err} below 2^{-40}. That is, it would be sufficient to use about 712MB as the size of D. Hence the choice of 1GB as the size of D in our experiments leaves abundant room to add more documents later.

For the document dataset, there are only 1,371,656 file-keyword pairs, which translates to a plaintext index size of 5MB (with 4-byte document IDs). Thus the size of D could be as low as 20MB. Note that the document collection itself is of size 1GB in this case. For rich data formats, it will typically be the case that the communication overhead due to SSE.indexgen would be only a fraction of the communication requirement for the documents themselves.

Search

Figure 4.12 shows the search performance of our scheme excluding the final decryption of the documents. Figure 4.12 does include overhead incurred at search time to handle lazy delete. We searched for the most frequent English word "the" and it was present in almost all the documents. (The exact query word is not mentioned in the previous work we are comparing against, so we chose a worst-case scenario for our experiments.) Our scheme performed better than [57] for all data sizes. Their scheme needs 17 ms, 34 ms and 53 ms for 4MB, 11MB and 16MB subsets of the Enron dataset respectively. Our scheme consumed 5 ms, 11 ms and 25 ms for 4MB, 8MB and 16MB subsets of the Enron dataset respectively. The search time grows proportionately to the size of the response. Figure 4.13 shows the search performance on the document dataset.

Note that our scheme uses a lazy deletion strategy to handle removals. This lazy delete mechanism allows us to obtain vastly improved security guarantees

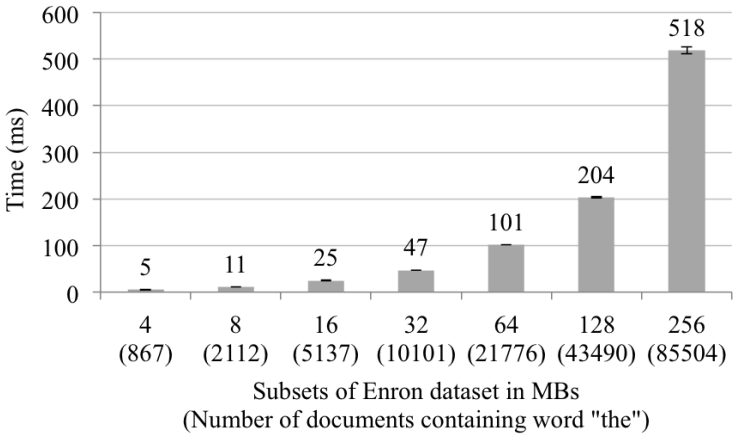


Figure 4.12: Search performance on the email dataset with 99% confidence intervals

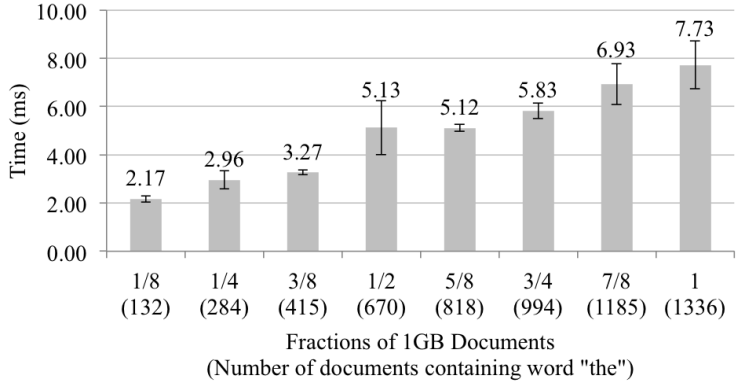


Figure 4.13: Search performance on the document dataset with 99% confidence intervals

by limiting the information leaked to the server (only for files uploaded during initial index generation). One might ask if this leads to any efficiency degradation during subsequent searches, since the actual updates to the index take place when a keyword that was contained in a deleted document is searched for later.

As it turns out (and as was experimentally confirmed), the overhead for searches does not significantly vary depending on whether the search operation involved a lazy deletion or not. This is because all search operations use the update mechanism of the underlying Blind-Storage scheme and the clear storage scheme. The efficiency of the update mechanism itself does not depend significantly on whether the file was modified or not. Indeed, in the case of Blind-Storage updates, it is important for the security that it must not be revealed to the server if a lazy deletion was involved or not.[16]

[16]We remark that our security model does not consider timing attacks. Depending on the

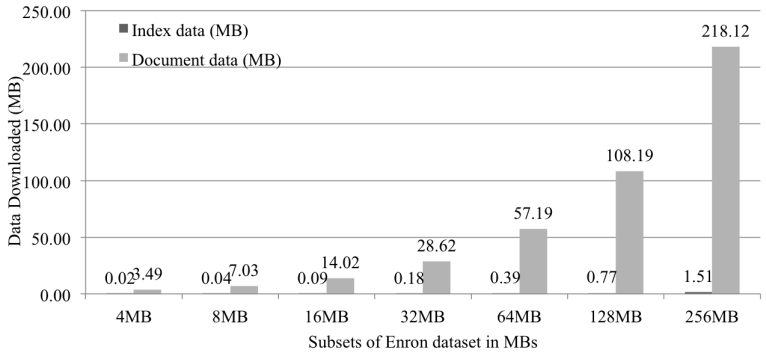


Figure 4.14: Communication needed for searching on the email dataset. The graph shows the size of the retrieved documents themselves alongside the extra communication incurred by our scheme.

Communication costs. As our scheme does not involve any server-side computation, we download slightly more data compared to [57]. But as shown in Figure 4.14, for the email dataset, the communication overhead is negligible compared to the size of the documents retrieved. The document dataset is much richer and contains much fewer keywords compared to the email dataset of the same size (1GB of documents in our dataset contains only 70MB of text), and therefore the overhead would be even lower for it.

Add

As opposed to [57] and other prior work, performance of our *add* operation does not depend upon the amount of data (i.e. the number of file-keyword pairs) already present in the searchable encryption system. Figure 4.15 shows the performance of addition of files of specified size when 256MB of data was initially indexed into the system.

Communication costs. We only need, on average, to download three blocks and upload two blocks per unique keyword in the document that is being added. (If the server supports an append operation that allows to append data to existing files on the server, we do not need to download any data during Add.)

implementation, we do not rule out a small dependence between the time taken and the extent of lazy delete computations involved. A serious implementation should take this into account. Since our SSE scheme is a relatively thin wrapper around the Blind-Storage mechanism, timing attacks can be effectively mitigated with relative ease.

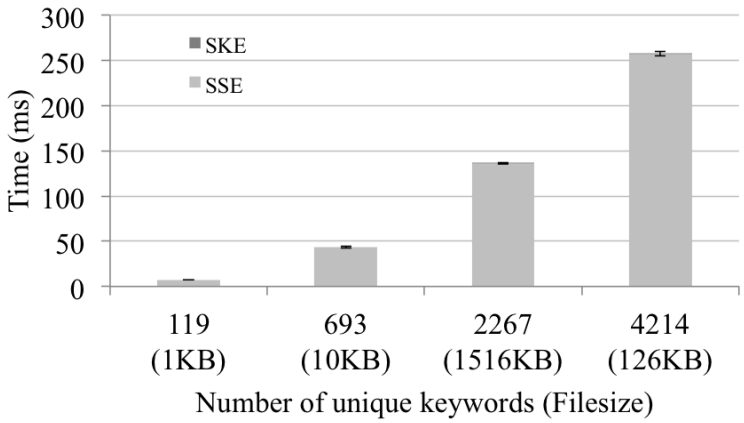


Figure 4.15: Add performance on email dataset with 99% confidence intervals. SKE costs are non-zero but very small.

Remove

The communication and computation cost of removing a document is virtually negligible, since it uses a lazy deletion strategy. Removal of a document in our scheme only requires the client to send a command to the server to delete the document from its file-system, and does not need any update to the searchable encryption index.

4.5.3 Summary

Evaluation of our scheme shows that it is more efficient, scalable and practical than prior schemes. Index generation in our scheme is more than *20 times* faster than that of [57]. Search operations are 2-3 times faster, in our experiments. Further, unlike [57], our addition and removal times are independent of the total number of file-keyword pairs, and is much more scalable. Removal in our scheme has virtually zero cost. We stress that several possible optimizations have not been implemented in this prototype.

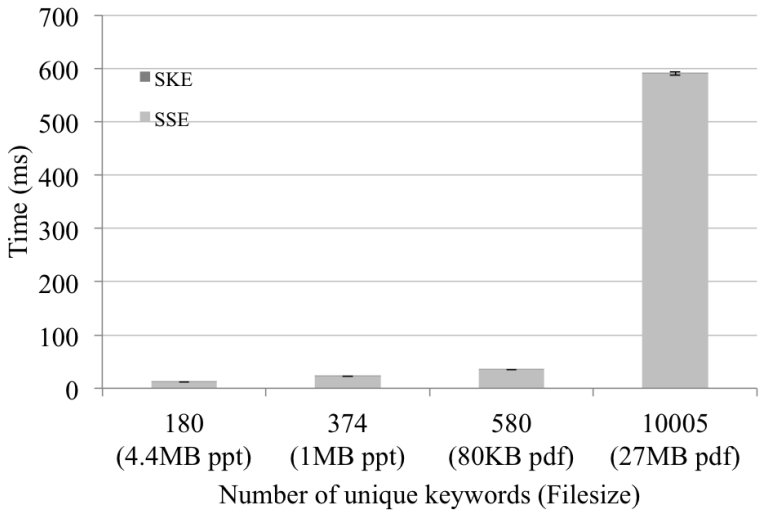
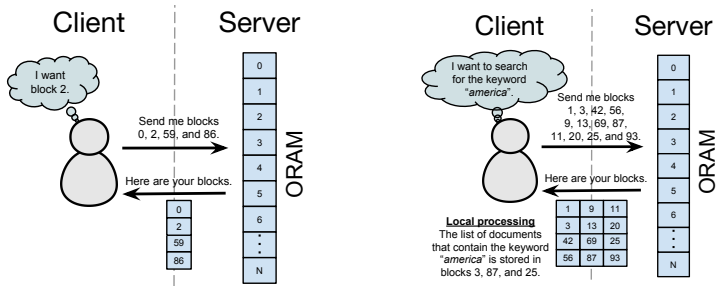


Figure 4.16: Add performance on document dataset with 99% confidence intervals: SKE costs are non-zero but very small.

4.6 Efficacy of Oblivious RAM in Searchable Encryption

Oblivious RAM (ORAM) enables accessing the memory without leaking the access pattern. A lot of effort is underway to make ORAM schemes efficient. Searching for the exact phrase "Oblivious RAM" on the Google Scholar returns 671 articles, 565 of them published since 2010. ORAM is a cryptographic primitive that allows accessing memory without leaking the access pattern; however, it provides a random access memory (RAM) interface that allows accessing a block of memory using its address. Therefore, the access pattern hiding properties of ORAM are only valid if memory is accessed single block at a time, which is not always the case with real applications. Many applications such as searchable encryption (SSE) or cloud storage[17], requires downloading multiple blocks at a time as illustrated in Figure 4.17, which leaks the number of blocks being accessed for a particular request. The number of blocks in turn leaks partial access pattern information. In SSE, each keyword query requires multiple blocks to be accessed. As different keywords require different number of blocks to be retrieved, the number of blocks retrieved leaks some information about the query. Similarly, in cloud storage, a file may consist of multiple blocks; therefore, downloading the file leaks its size

[17]By cloud storage, we mean services such as Dropbox and Google Drive that allow users to store their files.



(a) ORAM functionality **(b)** SSE on top of ORAM

Figure 4.17: ORAM functionality vs. SSE requirements. For each query, SSE accesses multiple blocks which leaks the number of blocks being accessed. As the number of blocks accessed depends upon the query, it leaks some information about the access pattern. The ORAM overhead of 4 shown is just for ease of exposition. (a). The client downloads multiple blocks (e.g., 4 in the figure) to access a single block from the ORAM. (b). The client needs to retrieve 3 blocks from the ORAM and therefore downloads 12 blocks. The server learns that the client downloaded 3 blocks.

(rounded off to the block size). Suppose that a file has a unique size, then every time the client accesses this file, the server observes, from the number of blocks being accessed, that the client is accessing the same file.

Symmetric Searchable Encryption (SSE) enables a client to store encrypted documents on a server and search over her encrypted documents to selectively retrieve the documents. In the SSE setting, the client prepares an inverted index for her documents and encrypts it using SSE, encrypts her documents using a symmetric cipher (such as AES), and sends both the encrypted index and the encrypted documents to the server. The server does not see the contents of the documents or the queries, but for each query it observes some information leakage: the search and document-access patterns (combination of search and document-access patterns is commonly referred to as access pattern). Access pattern is a significant leakage and may enable the server to infer information about the queries or the documents [9].

Preventing the leakage in SSE is one of the motivations for the applied ORAM research; however, no SSE construction that prevents the access pattern leakage has been proposed yet. A search for articles on the Google Scholar that contains both exact phrases "*Oblivious RAM*" and "*Searchable Encryption*" returns 214 articles. At least 18 ORAM papers cite Islam et al. 2012 paper titled "*Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation*" [9] to motivate the importance of eliminating leakage and the capability of ORAM to do so in applications such as SSE. Similarly, it is widespread in the SSE literature

that ORAM can completely eliminate leakage in SSE. We show that using ORAM to completely eliminate access pattern leakage or even achieve weaker notions of access pattern hiding either renders the communication performance worse than the trivial approach of streaming all of the outsourced data for each query or they do not provide any meaningful reduction in leakage. To the best of our knowledge, this is the first work studying the applicability of ORAM to SSE.

4.6.1 The fallacy of Composition

The fallacy of Composition occurs when one infers that something is true of the whole based on the fact that it is true for a part of the whole, without any justification for the inference. An example of the fallacy of Composition follows: Atoms are colorless. Cats are made of atoms. Therefore, cats are colorless.

The widespread fallacy of Composition of Oblivious RAM and Searchable Encryption is: *Searchable encryption leaks access pattern. Oblivious RAM eliminates access pattern. Therefore, there exists a method of using Oblivious RAM to eliminate access pattern in searchable encryption.* This fallacy is prevalent in ORAM and Searchable Encryption literature; however, for brevity we do not cite all of the papers.

4.6.2 Overview

We propose a new systematic methodology to study the applicability of ORAM to searchable encryption (SSE). We develop a baseline linear searchable encryption scheme: Linear–$\mathcal{LC}0$–SSE. It supports arbitrary queries and leaks absolutely no information, except an upperbound on the total size of the data stored on the server. For each query, Linear–$\mathcal{LC}0$–SSE, however, needs to stream all the outsourced data, storing only a small constant amount of data on the client at a time. The client needs to stream all the outsourced data, but she does *not* need to store all of it locally. She can stream in small chunks and discard any streamed chunk that does not satisfy the query as soon as the local search on it completes. This makes Linear–$\mathcal{LC}0$–SSE an excellent worst-case baseline to gauge the communication performance of any SSE scheme. If an SSE scheme needs to communicate more data then Linear–$\mathcal{LC}0$–SSE, then it is always better to use Linear–$\mathcal{LC}0$–SSE. Note that Linear–$\mathcal{LC}0$–SSE does *not* require ORAM.

We propose four new leakage classes: $\mathcal{LC}0, \mathcal{LC}1, \mathcal{LC}2,$ and $\mathcal{LC}3$. We also discuss $\mathcal{LC}4$ which is the leakage of standard SSE schemes [90, 2]. Each leakage class captures the information leaked to the server, with $\mathcal{LC}0$ leaking the least and $\mathcal{LC}4$ leaking the most information among the five classes. We prove that $\mathcal{LC}0$ is impossible to achieve without downloading all outsourced data for each and every query (communication required by Linear–$\mathcal{LC}0$–SSE). We propose single keyword SSE schemes for $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$. We empirically demonstrate that for a large fraction of queries, $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE perform worse than downloading the entire outsourced data. We emphasize that a small fraction of keywords are accessed most frequently (Zipf's law) and therefore these small number of keywords constitute a large fraction of the queries. Moreover, we demonstrate that $\mathcal{LC}3$ does not provide any meaningful reduction in leakage over $\mathcal{LC}4$. Our results show that more research is needed to develop techniques to reduce access pattern leakage in searchable encryption.

Contributions. We summarize our contributions below:

- **First work on composition of ORAM and SSE.** To the best of our knowledge, this is the first work studying the applicability of ORAM to reduce the access pattern leakage in SSE.
- **New Leakage Classes.** We propose four new leakage classes for SSE.
- **New Systematic Methodology.** We develop a new systematic methodology to study the applicability of ORAM to SSE. Although, we specifically developed the methodology for SSE, we believe that it can be useful for other applications as well. First, we design a baseline scheme, called Linear–$\mathcal{LC}0$–SSE, with a worst-case communication performance. Second, we propose static single keyword schemes for $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$. Third, we empirically study the communication performance of $\mathcal{LC}1$–SSE, $\mathcal{LC}2$–SSE, $\mathcal{LC}3$–SSE, and $\mathcal{LC}4$–SSE (Naveed's et al. scheme [2]) relative to Linear–$\mathcal{LC}0$–SSE.
- **Interesting findings.** We report interesting findings about the applicability of ORAM to the leakage prevention in SSE. Using our systematic methodology, we show that it is impossible to eliminate leakage in SSE without downloading the entire outsourced data. We show that $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE have query communication worse than that of Linear–$\mathcal{LC}0$–SSE for a large fraction of queries (a small fraction of keywords make up the large

fraction of queries). We also demonstrate that $\mathcal{LC}3$ and $\mathcal{LC}4$ leak almost the same amount of information.

- **Evaluation.** We provide a detailed evaluation of the query communication performance of the single keyword $\mathcal{LC}1$–SSE, $\mathcal{LC}2$–SSE, $\mathcal{LC}3$–SSE, and $\mathcal{LC}4$–SSE schemes using the Enron Email Corpus and the complete English Wikipedia Corpus.

Scope. We focus only on the applicability of ORAM to reduce access pattern leakage in SSE. ORAM has other applications such as secure multiparty computation and secure co-processor; however, these applications are out of scope of this thesis.

Most of the ORAM schemes work with a storage-only server with a few exceptions. Throughout the rest of the chapter, we assume the server to be a storage-only resource that allows the client to download and upload blocks of data.

Computation ORAMs. Recently ORAM schemes have been proposed that use computation on the server to reduce the communication overhead [91, 92, 93]. While these schemes have low communication overhead, they incur a large computation overhead on the server. In terms of respone time[18], the state of the art computation ORAM schemes incur at least as much overhead as the traditional ORAMs. The measurements in [93] report that it takes on the order of minutes to access a single block using Paillier cryptosystem. The computation time can be improved using NTRU cryptosytem at the cost of increased communication. While we focus on ORAM schemes without any computation, we believe that in terms of response time the efficiency arguments are valid for the computation ORAM schemes as well. Moreover, the access pattern leakage arguments hold for any type of ORAM.

4.6.3 Leakage Classes

We present five different leakage classes: $\mathcal{LC}0, \mathcal{LC}1, \mathcal{LC}2, \mathcal{LC}3$, and $\mathcal{LC}4$. $\mathcal{LC}0$ represents absolute minimum leakage and $\mathcal{LC}4$ captures the leakage of a typical static symmetric searchable encryption (SSE) scheme (e.g., [90, 2]). $\mathcal{LC}1, \mathcal{LC}2$, and $\mathcal{LC}3$ leak more information than $\mathcal{LC}0$ but less than $\mathcal{LC}4$. Table 4.1 shows all the leakage classes. To clearly explain the leakage, Table 4.1 shows the leakage for $\mathcal{LC}0, \mathcal{LC}1, \mathcal{LC}2$, and $\mathcal{LC}3$ relative to $\mathcal{LC}4$ which is the prevalent standard for

[18]The time it takes for an ORAM client to retrieve a block.

	Leakage Classes				
	$\mathcal{LC}0$	$\mathcal{LC}1$	$\mathcal{LC}2$	$\mathcal{LC}3$	$\mathcal{LC}4$
Setup Leakage					
Total combined size of index (if applicable) and all documents	✓	✓	✓	✓	✓
Total size of all documents	✗	✗	✓	✓	✓
Size of index	✗	✗	✓	✓	✓
Query Leakage For each queried keyword q:					
Search Pattern					
Number of times q is queried, i.e., the access frequency of q	✗	✗	✗	✗	✓
If q is the same or different from any of the keywords queried in the past	✗	✗	✗	✗	✓
Document-Access Pattern					
Identifiers of the documents that contain q	✗	✗	✗	✓	✓
Number of documents that contain q	✗	✗	✓	✓	✓
Size of each document that contain q	✗	✗	✗	✓	✓
Total size of all documents that contain q	✗	✓	✓	✓	✓

Table 4.1: Leakage Classes for Static Single Keyword Searchable Encryption. Padding can be used to hide the exact sizes and leak an upperbound on the sizes instead. ✗ shows the information that is *not leaked* and ✓ shows the information that is *leaked*.

SSE schemes in the literature [90, 2]; we believe that this juxtaposition make the difference in leakage easy to compare and understand.

For each leakage class we describe setup and query leakage. Setup leakage shows the information leaked when the client initially send the documents and index to the server and the query leakage shows the information leaked by the queries. Query leakage is divided into search pattern and document-access pattern: search pattern represents the leakage about the query keyword itself, while document-access pattern represents the leakage about the documents that contain the query keyword. Most of the literature combine search and document-access pattern and call it access pattern, however, for ease of exposition we explain them separately.

Padding can be used to hide the exact size and leak an upperbound on the size instead. For simplicity, we omit this fact from this point onwards unless necessary.

Leakage Class 0 ($\mathcal{LC}0$) represents the minimum possible leakage for an SSE scheme. It only leaks the combined size (with padding an upperbound on the size) of all documents and index (only if the SSE scheme uses index) during the initial outsourcing to the server; however, this information is impossible to hide given that the server is storing the data. No information whatsoever is leaked during queries, and hence, there is no query leakage.

Leakage Class 1 ($\mathcal{LC}1$) captures the minimum amount of leakage of an SSE scheme with the query communication complexity linear in the size of the documents that satisfy the query. As the query communication complexity is linear in the total size of the documents that satisfy the query, an upperbound on the total

size of the documents that satisfy the query is inherently leaked. Moreover, without appropriate padding, the exact total size of documents being retrieved is leaked; we consider this case for simplicity.

Leakage Class 2 ($\mathcal{LC}2$) leaks the size as well as the number of documents that satisfy the query. Moreover, during the setup the size of the index and total size of all documents is leaked.

Leakage Class 3 ($\mathcal{LC}3$) does not explicitly leak the search pattern, but does leak the document-access pattern. Document-access pattern implicitly leaks search pattern except in the following rare situation: If any two keywords q and q' appears in exactly the same set of documents, then the server is not able to distinguish between q and q'. This condition is rare in realistic data and therefore $\mathcal{LC}3$ is almost same as $\mathcal{LC}4$ which is leakage of standard SSE schemes. We explain $\mathcal{LC}3$ to demonstrate that a straightforward method of obliviously accessing only the index in an SSE scheme is not useful, we explain this in detail in Section 4.6.9. Fig. 4.26 shows that $\mathcal{LC}3$ is almost same as $\mathcal{LC}4$.

Leakage Class 4 ($\mathcal{LC}4$) captures the leakage of a static single keyword SSE scheme (e.g., [90, 2]). It leaks the complete search and document-access patterns.

4.6.4 Communication Baseline

In this section, we present a simple Linear–$\mathcal{LC}0$–SSE scheme, an SSE scheme with query communication complexity linear in the total size of all documents, as a communication performance baseline to study the communication performance of our single keyword $\mathcal{LC}1$–SSE, $\mathcal{LC}2$–SSE, and $\mathcal{LC}3$–SSE schemes we propose in Section 4.6.5.

Linear–$\mathcal{LC}0$–SSE scheme. We propose a Linear–$\mathcal{LC}0$–SSE scheme in Figure 4.18. The client encrypts her documents with any semantically secure symmetric-key encryption scheme (such as AES) and sends the encrypted documents to the server. Later, when the client wants to search her outsourced documents, she streams all of them. The client streams data in small chunks, searches them locally, and discards the streamed documents that do not satisfy the query.

Our Linear–$\mathcal{LC}0$–SSE scheme has the following properties:

- **Absolute minimum leakage ($\mathcal{LC}0$).** It has leakage class $\mathcal{LC}0$, which means

it leaks absolute minimum information.

- **Arbitrary queries.** It supports arbitrary type of queries[19].
- **Worst-case query communication.** It streams all outsourced documents to the client for each query.
- **Optimal query communication for $\mathcal{LC}0$.** It has optimal query communication. Theorem 4.6.6.1 shows that Linear–$\mathcal{LC}0$–SSE has optimal communication required to achieve $\mathcal{LC}0$.
- **Constant local storage.** It requires the client to store only a single document at any given instant; the client keeps the document if it satisfy the query and discards it otherwise. Therefore, in addition to the documents that satisfy the query, the client only needs to store a single more document.
- It does not need ORAM to achieve access pattern hiding.

Keygen

- The client generates a symmetric key K uniformly at random.

Setup

- The client encrypts all the documents as a single file using a semantically secure symmetric-key encryption scheme with key K, and sends it to the server.

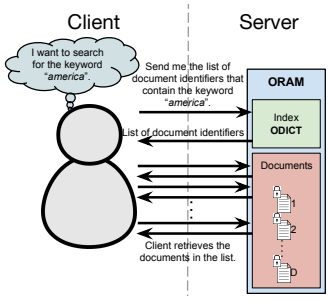
Search

- For each search query, the client streams all of the outsourced documents and searches locally. The client stores only a (small) constant amount of data, say a single document, at any given time. All documents that do not satisfy the query are discarded after the completion of local search.

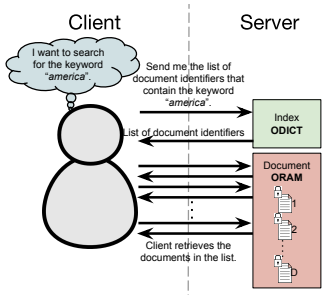
Figure 4.18: Linear–$\mathcal{LC}0$–SSE Scheme. Baseline for communication performance of SSE schemes we propose in Section 4.6.5.

Baseline Query Communication. Linear–$\mathcal{LC}0$–SSE supports arbitrary queries and has absolute minimum leakage ($\mathcal{LC}0$). The only problem Linear–$\mathcal{LC}0$–SSE has is that for each query it requires streaming all outsourced data to the client. Therefore, it serves as an excellent worst-case communication baseline for any

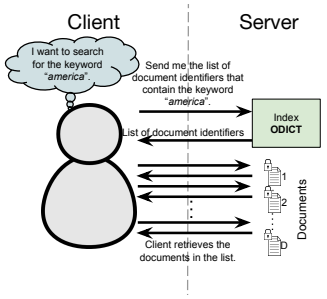
[19]Sublinear SSE schemes are designed for a specific type of queries such as single keyword and Boolean keyword queries.



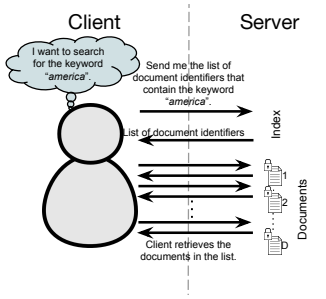
(a) Single Keyword $\mathcal{LC}1$-SSE



(b) Single Keyword $\mathcal{LC}2$-SSE



(c) Single Keyword $\mathcal{LC}3$-SSE



(d) Single Keyword $\mathcal{LC}4$-SSE. This captures leakage of standard SSE schemes such as [90, 2].

Figure 4.19: In all the schemes, the client searches for a keyword and retrieves a list of document identifiers that contain the keyword. Next the client retrieves the documents in this list from the server. *For simplicity, the figures are only showing the important details.*

SSE scheme. *Any SSE scheme with query communication worse than the Linear–$\mathcal{LC}0$–SSE scheme can be trivially replaced with the Linear–$\mathcal{LC}0$–SSE scheme.*

4.6.5 Constructions

In this section, we first show that achieving $\mathcal{LC}0$ is, in general, impossible with query communication better than that of Linear–$\mathcal{LC}0$–SSE. We construct single keyword SSE schemes for $\mathcal{LC}1$, $\mathcal{LC}2$, and $\mathcal{LC}3$ such that they have *optimal query communication* for the respective leakage class. These schemes capture simple ways of using Oblivious RAM as a blackbox to reduce leakage in SSE. We show in section 4.6.9 that query communication of $\mathcal{LC}1$-SSE and $\mathcal{LC}2$-SSE is worse than that of Linear–$\mathcal{LC}0$–SSE for a large fraction of queries (only a small number of keywords constitute most of the queries). Moreover, $\mathcal{LC}3$-SSE has acceptable query communication but as explained in section 4.6.9 it does not provide meaningful reduction in leakage compared to $\mathcal{LC}4$. Security of our schemes follows from the

security of the underlying ORAM and ODICT schemes; therefore, we omit the security proofs.

Oblivious Dictionary (ODICT). Sublinear SSE schemes require inverted index to function. Index needs a dictionary data structure and ORAM is not a dictionary data structure. Therefore, we use AVL tree based Oblivious Map scheme of Wang et al. [94] as an Oblivious Dictionary in our schemes. We consider that ODICT stores key value pairs and ODICT.Lookup(key) operation returns the value associated with the key.

4.6.6 $\mathcal{LC}0$–SSE

We show that a scheme with Leakage Class 0 ($\mathcal{LC}0$) necessarily requires downloading the entire outsourced data for each query. Therefore, completely eliminating the leakage with communication less than that of the Linear–$\mathcal{LC}0$–SSE scheme (i.e., streaming all of the outsourced data) is impossible.

Theorem 4.6.6.1. *The lower bound on the query communication complexity to achieve the Leakage Class 0 ($\mathcal{LC}0$) is $|\mathbb{D}|$, where $|\mathbb{D}|$ is the total size of all outsourced documents.*

Proof. We prove that if for any query the communication is less than $|\mathbb{D}|$, then the information leaked to the server is strictly more than $\mathcal{LC}0$. Suppose for a query q, the client retrieves documents with total size $|\mathbb{D}'| < |\mathbb{D}|$, then the server learns the size of the documents being retrieved $|\mathbb{D}'|$. Suppose the client has another query q' that is satisfied by all the documents. For an SSE scheme to be correct, it has to download all the documents for the query q', i.e., the size of the documents retrieved for the query q' needs to be $|\mathbb{D}|$. As the size of the data retrieved for the query q and the query q' are different, the server can distinguish the query q from the query q', which is strictly more leakage than $\mathcal{LC}0$. Moreover, the server learns an upperbound on the total size $|\mathbb{D}'|$ of the documents that match the query q. This shows that the only way to prevent this leakage is to retrieve all documents for every query. Therefore, we conclude that $|\mathbb{D}|$ is the optimal communication for the Leakage Class 0 ($\mathcal{LC}0$). □

4.6.7 $\mathcal{LC}1$–SSE

$\mathcal{LC}1$-SSE prevents as much leakage as possible while keeping the query communication less than the communication of Linear–$\mathcal{LC}0$–SSE. We propose a single keyword $\mathcal{LC}1$-SSE scheme to study the query communication required to achieve $\mathcal{LC}1$.

Single Keyword $\mathcal{LC}1$–SSE Scheme. We present our Single Keyword $\mathcal{LC}1$–SSE scheme in fig. 4.20. The client uses an ORAM O to store both the index ODICT and the documents. To query a keyword, the client looks up the inverted index stored in ODICT D_O to retrieve the list L of document identifiers that contain the query keyword. She then retrieves all the documents in the list L from the ORAM O.

Query Communication. Let n and d be the number of documents and the total size of the documents that satisfy a query, e be the size of n document identifiers, o be the overhead of the ORAM, co be the overhead of the ODICT (where c shows the number of ORAM accesses required for a single ODICT lookup), and B be the blocksize used in ORAM and ODICT, then the communication overhead of $\mathcal{LC}1$-SSE is $co \times \max(B, \lceil \frac{e}{B} \rceil) + o \times \max(B, \lceil \frac{d}{B} \rceil)$. We demonstrate using real datasets in Section 4.6.9 that the optimal communication single keyword $\mathcal{LC}1$–SSE scheme requires more communication than Linear–$\mathcal{LC}0$–SSE for a large fraction of the queries (a small number of keywords constitute a large fraction of queries).

4.6.8 $\mathcal{LC}2$–SSE

We propose a single keyword $\mathcal{LC}2$–SSE scheme to investigate Leakage Class 2 ($\mathcal{LC}2$). Our single keyword $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE schemes differ due the fact that the former stores both index ODICT and documents in the same ORAM while the later stores index ODICT and documents separately as shown in fig. 4.19.

Single Keyword $\mathcal{LC}2$–SSE Scheme. We describe a single keyword $\mathcal{LC}2$ scheme in Figure 4.21. As shown in Figure 4.19b, it uses an ODICT to store index and an ORAM to store the documents. To search for a keyword q, the client retrieves a list L of document identifiers of documents that satisfy q from the ODICT D. Next, the client retrieves the documents in list L from the document ORAM O.

Query Communication. Let n and d be the number of documents and the total size of the documents that satisfy a query, e be the size of n document identifiers,

Keygen.

- The client generates a symmetric key K_O uniformly at random and use it in all symmetric key operations of ODICT D_O and ORAM O.

Setup.

- The client setups an ORAM O on the server. She uses ORAM O to store both an ODICT D_O for the index and the documents.
- The client creates an inverted index for all documents.
- The client writes the index in the ODICT D_O. This can be done during ORAM O setup.
- The client writes all the documents in the ORAM O. This can also be done during ORAM O setup.

Search.

- To query a keyword q, the client queries the ODICT D_O using D_O.Lookup(q) and retrieves the list L of documents that contain the keyword.
- The client retrieves all the documents in list L from ORAM O.

Figure 4.20: Single Keyword $\mathcal{LC}1$-SSE Scheme

o be the overhead of the ORAM, o' be the overhead of the ODICT, and B be the blocksize used in ORAM and ODICT, then the communication overhead of $\mathcal{LC}2$-SSE is $o' \times \max(B, \lceil \frac{e}{B} \rceil) + o \times \max(B, \lceil \frac{d}{B} \rceil)$. As $\mathcal{LC}1$–SSE stores both index ODICT and documents in the same ORAM and $\mathcal{LC}2$–SSE stores them separately, therefore, the size of the ORAM in $\mathcal{LC}1$–SSE is bigger than index ODICT and document ORAM in $\mathcal{LC}2$–SSE. Moreover, the ORAM and ODICT access increases with their size, the $\mathcal{LC}2$–SSE has slightly better query communication; however, it may not worth the extra amount of information being leaked.

$\mathcal{LC}3$–SSE.

We propose a single keyword $\mathcal{LC}3$–SSE scheme to study Leakage Class 3 ($\mathcal{LC}3$). Our single keyword $\mathcal{LC}3$–SSE scheme uses ODICT for the index but stores document using plain encryption without ORAM.

Single Keyword $\mathcal{LC}3$–SSE Scheme. We present a Single Keyword $\mathcal{LC}3$–SSE scheme in Figure 4.22. This scheme uses ODICT for the inverted index and symmetric key encryption to store the documents.

Keygen.

- The client generates a symmetric key K uniformly at random, which is used for encrypting blocks of ODICT D and ORAM O.

Setup.

- The client creates an inverted index for all documents.
- The client writes the index in an ODICT D stored on the server. This can be done during ODICT D setup.
- The client writes all the documents in an ORAM O stored on the server. This can be done during ORAM O setup.

Search.

- To query keyword q, the client queries ODICT D using D.Lookup(q) and retrieves the list L of document identifiers that contain the keyword.
- The client retrieves all the documents in list L from ORAM O.

Figure 4.21: Single Keyword $\mathcal{LC}2$-SSE Scheme

Query Communication Complexity. Let n and d be the number of documents and the total size of the documents that satisfy a query, e be the size of n document identifiers, o' be the overhead of the ODICT, and B be the blocksize used in ODICT, then the communication overhead of $\mathcal{LC}3$-SSE is $o' \times \max(B, \lceil \frac{e}{B} \rceil) + d$.

4.6.9 Evaluation

We have implemented our protocols to evaluate the query communication; however, we only simulate the ORAM accesses, which we believe is enough to evaluate the query communication overhead. We present detailed and comprehensive experiments.

In the first set of experiments fig. 4.23, which we call realistic experiments, we use simulated PathORAM ($4\log(N)$ overhead, where N represents the total number of blocks) [95] for the document ORAM and the Wang's et al. AVL tree based Oblivious Map ($(4\log(N))^2$ overhead) [94] for the index ODICT.

In the second set of more conservative experiments fig. 4.24, which we call optimistic experiments, we use simulated ideal ORAM with $\log(N)$[20] overhead

[20]No existing ORAM scheme have this overhead. A loose ORAM bound proved by [96] is $\log(N)$.

Keygen

- The client generates two keys K and K' uniformly at random. K is used to encrypt blocks of ODICT D and K' to encrypts the documents.

Setup:

- The client creates an inverted index for all documents.
- She writes the index in an ODICT D. This is done during ODICT D setup.
- She encrypts each document with key K using a semantically secure symmetric key encryption and send the encrypted documents to the server.

Search:

- To query a keyword q, the client queries the ODICT D using D.Lookup(q) and retrieves the list L of documents that contain the keyword.
- The client retrieves all the documents in the list L from the server.

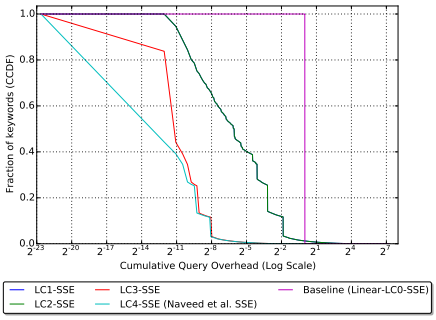
Figure 4.22: Single Keyword $\mathcal{LC}3$-SSE Scheme

for the document ORAM. Furthermore, we assume that a single ODICT lookup requires a single ORAM accesses.

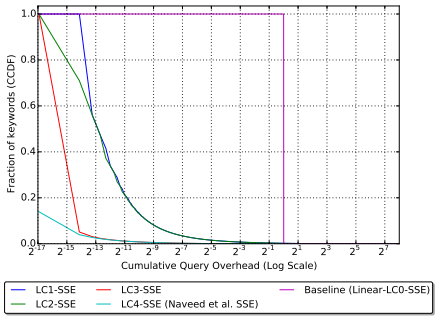
We use a blocksize of 4KB throughout.

Before presenting our results, we describe the preliminaries required to understand our results.

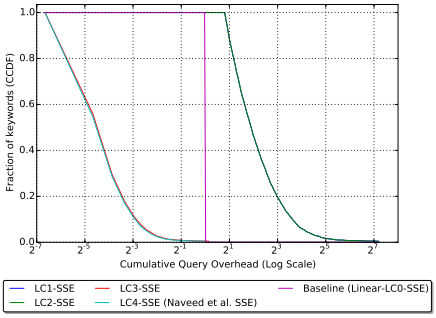
- **Empirical CCDF Plots.** We present our evaluation results using *empirical* Complementary Cumulative Distribution Function (CCDF) plots. For example, a point (x, y) in figs. 4.23 and 4.24 indicate that the query overhead is at least x for y fraction of the keywords.
- **Query overhead.** Query overhead of a query is the ratio of the amount of data retrieved in an SSE scheme to the total size of all the outsourced documents. This allows us to compare the query communication of our SSE schemes to our baseline Linear–$\mathcal{LC}0$–SSE scheme: if the ratio is greater or equal to 1, then the SSE scheme does not perform better than our baseline Linear–$\mathcal{LC}0$–SSE.
- **Keyword Frequency.** We plot keyword frequencies of all the keywords in the Enron email corpus and the English Wikipedia corpus in fig. 4.25; which shows that keywords follow Zipf's law.



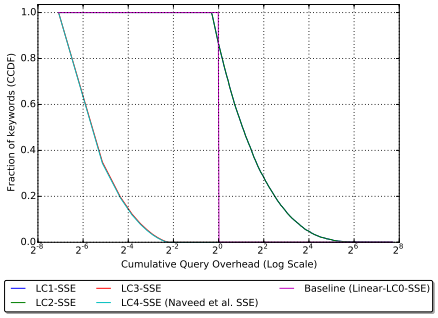
(a) Enron Email Corpus (all $628,908$ keywords)



(b) English Wikipedia Corpus (all $4,400,034$ keywords)



(c) Enron Email Corpus (most frequent $5,000$ keywords)

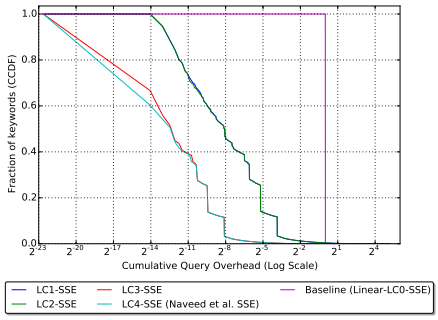


(d) English Wikipedia Corpus (most frequent $10,000$ keywords)

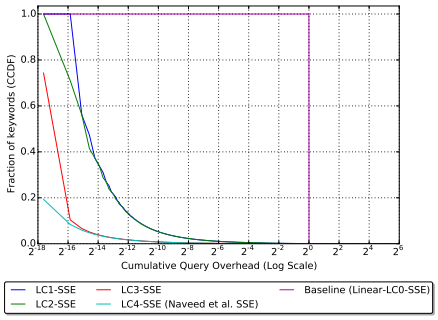
Figure 4.23: Cumulative Query Overhead using PathORAM for the document ORAM and AVL tree based Oblivious Map [94] for the Index ODICT. Query overhead of a query is the ratio of the amount of data retrieved in an SSE scheme to the total size of all the outsourced documents. A point (x, y) in the plots indicates that the query overhead is at least x for y fraction of the keywords.

- **Query Pattern.** The more frequent the keyword is the more frequently it will be queried (we have removed all stopwords). As shown in fig. 4.25, a small number of keywords have very high frequency, therefore, these small number of keywords constitutes most of the queries. This fact is important in understanding the query overhead; in addition to query overhead plots for the all the keywords, we present query overhead plots zoomed in at the tail to clearly show the overhead for the most frequent keywords that make up a large fraction of the queries.

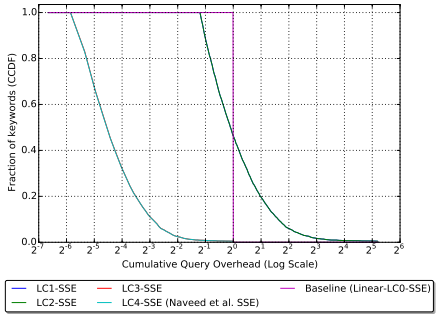
Datasets. We use two datasets: the complete English Wikipedia corpus and the Enron email dataset. The English Wikipedia corpus contains 4,825,180 distinct articles and is 9.8GB in size. Enron email dataset [89] contains emails from



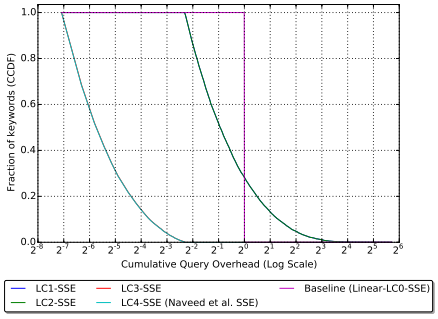
(a) Enron Email Corpus (all $628,908$ keywords)



(b) English Wikipedia Corpus (all $4,400,034$ keywords)



(c) Enron Email Corpus (most frequent $5,000$ keywords)



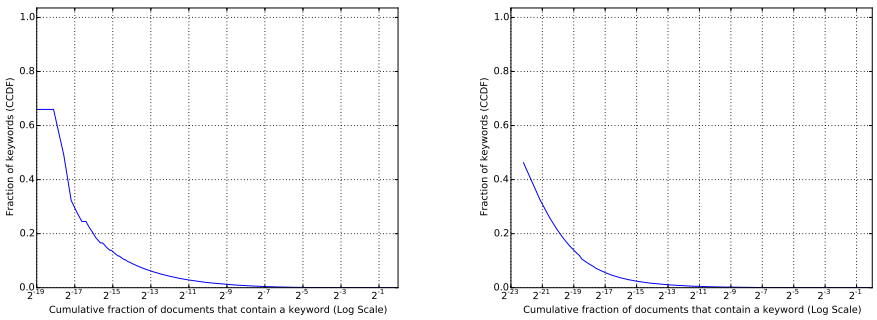
(d) English Wikipedia Corpus (most frequent $10,000$ keywords)

Figure 4.24: Cumulative Query Overhead using an ideal ORAM overhead of $\log(N)$, where N is the total number of blocks, for both the documents ORAM and index ODICT. We assume that a single ODICT lookup requires a single ORAM access. Query overhead of a query is the ratio of the amount of data retrieved in an SSE scheme to the total size of all the outsourced documents. A point (x, y) in the plots indicates that the query overhead is at least x for y fraction of the keywords.

150 Enron employees, mostly senior managers. The dataset was made public by the Federal Energy Regulatory Commission during its investigation of Enron. The dataset is 1.32GB in size and has 517,424 emails. Enron dataset has been extensively used to evaluate searchable encryption schemes [90, 97, 2].

We removed all stopwards and non-alphabetical characters, and converted all keywords to the lowercase. After this preprocessing, English Wikipedia corpus had 4,400,034 keywords and Enron data had 628,908 keywords.

Realistic Experiments. As explained above we use PathORAM for the documents and AVL tree based Oblivious Map for the index ODICT. $\mathcal{LC}3$–SSE does not use document ORAM and $\mathcal{LC}4$–SSE does not use both document ORAM and index ODICT.



(a) Enron Email Corpus **(b)** English Wikipedia Corpus

Figure 4.25: Keyword Frequency. A point (x, y) in the plots indicates that the fraction of the number of documents that contain a keyword is at least x for y fraction of the keywords.

Fig. 4.23a shows the query overhead for all keywords in the Enron Email Corpus. $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE have the same query overhead and therefore their curves are overlapping. $\mathcal{LC}3$–SSE query overhead is slightly more than that of $\mathcal{LC}4$–SSE. Moreover, $\mathcal{LC}1$–SSE and $\mathcal{LC}2$–SSE query overhead is more than 1 for a large fraction of queries (a small number of keywords constitutes a large fraction of queries). Fig. 4.23c shows zoomed in view of the tail of fig. 4.23a for $5,000$ most frequent keywords, which shows that for all of the $5,000$ most frequent keywords the query overhead is at least 1.75.

Fig. 4.23b and fig. 4.23d shows similar results for the English Wikipedia corpus.

Optimistic Experiments. Fig. 4.24 shows results for our optimistic experiments, where we use ideal ORAM and ODICT, both with the communication overhead of $\log(N)$, where N is the number of blocks. The results are better than our realistic experiments, however, a large fraction of queries still has query overhead of more than 1.

Leakage Analysis of $\mathcal{LC}3$–SSE and $\mathcal{LC}4$–SSE. Our experiments show that $\mathcal{LC}3$–SSE query overhead is almost same as $\mathcal{LC}4$–SSE; therefore, we investigate whether $\mathcal{LC}3$–SSE provide meaningful reduction in leakage compared to $\mathcal{LC}4$–SSE. As explained in Section 4.6.3 $\mathcal{LC}3$–SSE prevents explicit leakage of search pattern, but leaks search pattern indirectly from the document access pattern. Specifically, the only information that is not leaked about the search pattern from the document-access pattern is whether the client is querying for two keywords that appear in exactly the same set of documents. Fig. 4.26 shows the fraction of keywords for which there are at least a fraction of other keywords appearing in the same set of

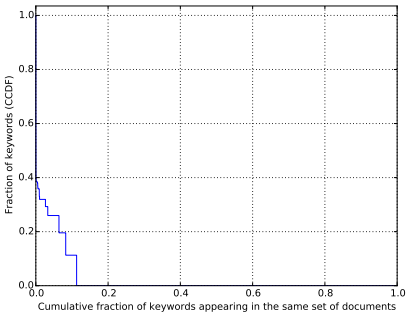


Figure 4.26: Leakage Advantage of $\mathcal{LC}3$–SSE over $\mathcal{LC}4$–SSE for Enron Email Corpus. A point (x, y) in the plots indicates that the fraction of keywords that appear in the same set of documents is at least x for y fraction of the keywords.

documents. As it can been seen, a very small fraction of keywords have a very small fraction of other keywords that appear in the same set of documents; therefore, we conclude that $\mathcal{LC}3$ leaks almost as much as $\mathcal{LC}4$. This implies that oblivious index accesses alone does not provide a meaningful reduction in leakage if the documents are accessed in a non-oblivious fashion. Therefore, for any meaningful reduction in leakage both the index and documents accesses need to be oblivious.

Chapter 5

A Practical Model for Computing on Encrypted Data

In the previous chapter, we developed a novel model and constructions to search over encrypted data. While search is useful in many applications, there are many other applications that require much richer functionality than just searching over the encrypted data. In this chapter we develop a new model to allow an authorized party to learn plaintext result of computation on encyprted data. The propsed model, called Controlled Functional Encryption, is a variant of Functional Encryption, but allows for construction of very efficient and secure schemes.

Suppose volunteers would like to offer their genomic data for research, as long as a strong privacy policy can be enforced without having to trust individual researchers. We remark that beyond being an illustrative example for us, abuse of sensitive data by researchers who ignore or are unaware of the limits set by the data owner is in fact a real problem, as evidenced by the case of the Havasupai tribe against the Arizona State University [98, 99]. In 1989, researchers from Arizona State University partnered with the Havasupai Tribe, a community with high rates of Type II Diabetes, to study links between genes and diabetes risk. When the researchers were not successful in finding a genetic link, they used the DNA from blood samples for other unrelated studies such as schizophrenia, migration, and inbreeding, all of which are taboo topics for the Havasupai [98]. Such unfortunate incidents can be avoided if researchers do not have direct access to this data, but can only carry out computations on this data that are subject to restrictions specified by the data owners. The restriction can include a limit on the total amount of information (number of bits) revealed to the researchers by a contributed piece of data over its life-time, a restriction on the kind of functions that can be computed on the contributed data or a requirement that certain amount of noise be added to any information computed from this data, a restriction on the number or class of researchers who can access it, etc. On the other hand, the researchers are typically not willing to publicly reveal the specific functions they are interested in. Solving this problem satisfactorily for all parties calls for leveraging tools from modern

cryptography.

Where existing approaches fail. One approach to solving this problem would be to use *secure two-party computation*, given the recent advances in practical implementations of this tool [20, 21, 22, 23, 24, 25, 26, 27]. While this would indeed address all the privacy concerns, this solution is not suited for our scenario for a couple of reasons: firstly, this requires each volunteer to interact separately with each researcher interested in using their genomic data; secondly, the researchers need to decide what all functions they need to compute by the time they interact with each volunteer, which will prevent them from adapting their functions as research progresses.

Another potential (albeit currently experimental) solution offered by modern cryptography would be to use *functional encryption* or FE – first formalized in [11], which covers popular cryptographic primitives like Identity Based Encryption (IBE), Attribute Based Encryption (ABE), etc. as special cases [100, 101, 102, 103, 104, 67, 105, 106, 64]. An FE scheme would allow volunteers to encrypt their data in such a way that for each function, a researcher can use a key issued by a trusted authority to compute just that function of the data. This shifts the responsibility of limiting the information revealed about each genome from individual volunteers to an authority, *without the need to trust the authority with the genomic data itself*. However, functional encryption does not provide a suitable solution to our problem for a few reasons. Firstly, functional encryption schemes tend to be extremely inefficient. Also, importantly, functional encryption currently relies on relatively new and untested cryptographic assumptions. Further, to ensure that when the authority issues a key for one ciphertext, it does not allow computing the function on other ciphertexts (belonging to the same data-owner, or different data-owners), one will need to employ additional mechanisms and also use functional encryption for a more complex function family than is originally required.

Our Contribution: Controlled Functional Encryption. These limitations lead us to formulating a new notion of functional encryption — called *controlled functional encryption* (C-FE) — which addresses all the above privacy and usability issues, as well as relies only on mature and efficient cryptographic tools (unlike the standard notion of functional encryption). Our main contributions in this work are:

1. *Definitions.* We present carefully formulated security definitions for various forms of C-FE. We use simulation-based security definitions, which give comprehensive security guarantees against corrupt data users (clients).

2. *Constructions.* We design theoretical and practical constructions of C-FE schemes achieving these definitions for specific and general classes of functions. Our constructions, even for arbitrary functions, are practically efficient and rely only on well-studied cryptographic primitives.

3. *Applications and Performance Evaluation.* We confirm that our proposed constructions are indeed practical by evaluating their performance on large scale data. We also discuss the applications of C-FE to personalized medicine, patient similarity, paternity test and kinship test.

It is instructive to compare C-FE with FE: both rely on a central authority to issue keys that have customized functionality. FE has the feature that one key can be used on several ciphertexts, where as C-FE forbids this. C-FE has several advantages over FE, in situations where it is applicable. Firstly, C-FE avoids various impossibility results that affect FE. In particular, simulation-based security definitions are typically impossible to achieve for functional encryption (e.g., [107, 11, 108]), unless severe limitations are placed on the number of keys that can be issued (e.g., [63, 64]). Secondly, as we show in this work, even for arbitrary functions, C-FE admits more practical constructions than are known to be possible for FE (e.g., [105, 106, 109]); also FE constructions often use relatively untested assumptions (e.g., sub-exponential Learning With Errors assumption, or assumptions related to multi-linear maps) which are hard to rely on given fast improvements in cryptanalysis (e.g., [110]), whereas C-FE constructions can be based on much more well understood cryptographic primitives like public-key encryption and oblivious transfer. In short, we present C-FE as a practical tool, with strong security guarantees and efficient constructions based on mature cryptographic primitives, that has direct applications to several practical scenarios for which concepts like secure multi-party computation and FE do not provide satisfactory solutions.

5.1 Overview

Defining C-FE. In C-FE, the parties can be *Data Owners*, *Clients* or a *Central Authority*. The clients wish to compute functions of the data belonging to individual data owners. The data owners will contribute their data to the system once, with attached policy parameters (e.g., number of times it can be used in a computation)

and go offline. They rely on the central authority to ensure that various policies are enforced regarding what functions of their data can be computed by the clients.[1] Further, the (honest-but-curious) central authority itself should not learn any information about the data (except policy parameters explicitly specified by the data owner), and the data owners should not learn anything about how their data is being used, except for the fact that their policies will be enforced.

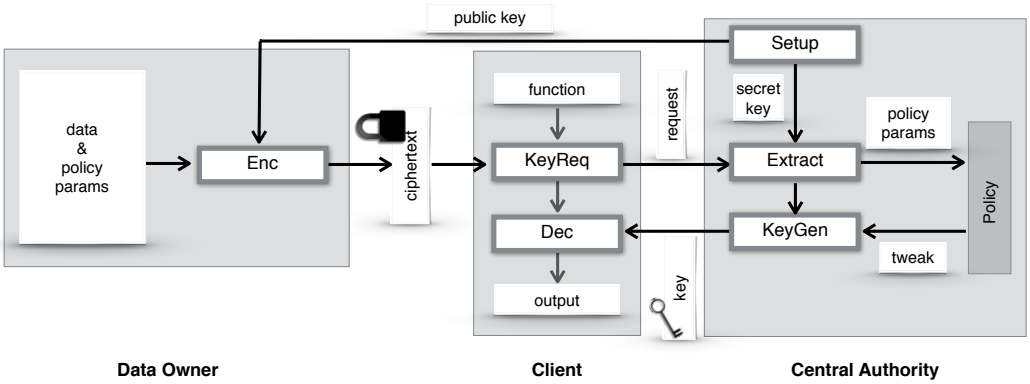


Figure 5.1: Components of a C-FE scheme, described formally in Section 5.2.

Each time a client wishes to evaluate a function f of a piece of data x given to it (as a ciphertext), it makes a "key-request" to the authority. From this key-request, the authority can recover the policy parameter λ attached to x, and decide (using arbitrary logic extraneous to the C-FE scheme) whether to issue a key or not. If it issues a key, the client can use it to extract $f(x)$ (and nothing else).

We shall require security against clients who behave arbitrarily (i.e., not necessarily following the scheme); but the central authority is considered to be honest-but-curious (i.e., possibly monitored by a passive, eavesdropping adversary, but otherwise, following the specified scheme). We require that a corrupt (honest-but-curious) authority does not collude with any other parties. The other parties may collude arbitrarily. On a technical note, our main security definition is essentially Universally Composable (UC) security, except for the above restrictions on the corruption of the authority.

There are several optional variations to this basic set of requirements that is

[1] The policies themselves are extraneous to the C-FE scheme.

important in many applications, and can be accommodated in our constructions (often with little overhead).

1. *Tweaks.* We may allow the authority to *tweak* the function, possibly using randomness hidden from the client. For instance, the authority can add noise to an outcome. More generally, tweaks can be used to restrict the class of functions that can be computed (without the authority learning the exact function being computed).

2. *Function Hiding.* We can consider the requirement that the authority should not learn the specific function computed by the client on the data owner's data, or that it learns only partial information about the function.[2]

3. *Pattern Hiding.* Another optional security requirement would be to allow a client to hide from the authority whether or not, in two separate computations, it is computing with the same piece of data from a data owner. Instead, if we require that the authority is told when the same piece of data is computed on again, we say the C-FE scheme is *pattern revealing*.

We remark that if a scheme is indeed pattern hiding, then certain policies like 'allow researchers to compute only 10 functions on my data' cannot be enforced. This is just the price one has to pay in order to provide more security to the clients.

We also remark that the various design choices above are made to not only allow practically efficient constructions, but also to suit the security and usability requirements of the applications that motivated this work. In particular, the non-interactive nature (encryptions instead of protocols) and the offline nature of data owners make a C-FE scheme much easier to deploy in these applications than a solution relying on, say, secure multi-party computation.

Constructing C-FE. We present a collection of constructions meeting different levels of security, efficiency and generality.

1. First, we present a construction for the inner product function, wherein the data owner's input is a vector of integers and the function to be computed by the client is also specified by a similar vector; the output of the function is the inner product of these two vectors.[3] This construction is extremely efficient, and is

[2]Note that allowing the authority to learn more information about the function would make it easier to enforce more complex policies, at some privacy cost to the clients. In settings like research on genomic data, the data from the data owners are considered much more sensitive than the inputs from the clients (researchers), and the latter may be trusted with a central authority.

[3]The inner product is defined over the ring of integers modulo some integer N. In typical applications, like set-intersection cardinality or set-union cardinality, N is chosen to be large relative to the individual entries in the vectors and the dimension of the vectors, so that the outcome is the

based solely on (CCA2 secure) public-key encryption (instantiated using a hybrid encryption scheme involving RSA-OAEP for key encapsulation and AES for data encapsulation). This construction allows for an additive tweak, but does not allow function hiding or pattern hiding.

This is a primitive with numerous applications, as various other functions can be reduced to the secure computation of the inner product function, possibly with tweaks. For instance, set-intersection cardinality (size of the intersection of two sets) can be modeled as the inner product of binary vectors representing the two sets, modulo a number N that is no less than the dimension of the vectors.

2. Next we present a *general construction* relying on the powerful *garbled circuit* construction due to Yao (or more generally, using any *decomposable randomized encoding* [111]). This construction admits any arbitrary function family, with arbitrary tweaks. More precisely, for *any* (efficiently computable) function F, the client obtains the outcome $F(f, x, w)$, where x is the data, f is the function specification from the client, and w is the tweak specification from the authority (with pre-determined bounds on the size of x, f and w). Prior to this, the authority obtains the policy parameter λ attached to x, and can use this to determine whether to issue a key to the client or not, and if so what value of w to use.

There are three variants of this construction, with different levels of efficiency.

(a) *Without Function Hiding or Pattern Hiding.* This variant relies solely on CCA2 secure public-key encryption and a block-cipher (for implementing the garbled circuits).

(b) *With Function Hiding but not Pattern Hiding.* This variant additionally requires an oblivious transfer (OT) primitive. Several practical OT schemes are known, and are used along with garbled circuits in practical implementations of secure 2-party computation.

(c) *With Function Hiding and Pattern Hiding.* This variant is similar to the above construction, but the CCA2 secure encryption is replaced by *Rerandomizable Replayable CCA* encryption. This is a relatively recent primitive, currently with only one construction in the literature [112], which achieves provable security based on the well-studied Decisional Diffie Hellman (DDH) assumption. Its efficiency is comparable to that of (unoptimized) OT schemes, but it is likely that further (possibly heuristic) efficiency improvements are possible.

inner product over integers (not modulo any integer). We point out that this function is different from the inner product *predicate* considered in the functional encryption literature, which only computes whether the inner product is zero or not: the two functions are incomparable.

An important extension that naturally applies to all the above variants is that of *multi-input* C-FE: this allows the client to compute a function not just on a single piece of data obtained from a data owner, but also a function that takes as input multiple such pieces of data. One can also obtain straightforward extensions that employ digital signatures to authenticate the information provided by the data owners (when there is a public-key infrastructure). For simplicity, we do not discuss these extensions in detail as they are relatively straightforward.

We point out that many recent advances in cryptography like Functional Encryption currently rely on new and largely untested hardness assumptions (related to bilinear pairings and lattices). Given the fast improvements in cryptanalysis (e.g., [110]), these should still be considered at an experimental stage, and several years from practical adoption. Unlike these tools, the above constructions of C-FE use only mature cryptographic tools and assumptions (public-key and symmetric-key encryption, DDH assumption, OT), that can be readily deployed today.

Applications and Performance Evaluation. Firstly, we revisit the example considered at the beginning of this section. Suppose that each volunteer wishes to enforce a limit on the total number of one-bit functions that can be computed on their genome. Given a (pattern revealing) C-FE scheme for the family of functions that the researchers are interested in (with or without function hiding depending on the requirements of the researchers, and with or without tweaks depending on the privacy requirements of the data owners), we let each volunteer play the role of a data owner, each researcher a client, along with a central authority that enforces the policy; the policy parameter λ attached to each genome will include a bound on number of times that genome can be used by a researcher. When a researcher wants to evaluate a one-bit function on an input, she makes a key-request. The authority recovers the policy parameter specified by the genome owner, and using the pattern information so far, determines if the key-request should be honored or not. If it decides to respond, it uses the C-FE scheme to send the key to the researcher.

We point out that this solution simultaneously resolves all the competing privacy and usability requirements in our scenario.

Other applications that motivated this work include personalized medicine, patient-similarity, and paternity and kinship testing. The function families that need to be supported in these applications are varied: examples include inner product between the input vector and a vector given by the client (over integers),

Hamming or Levenshtein distance of the input vector with a given vector, Smith-Waterman algorithm for aligning a sequence with a given sequence, etc.[4] We evaluate the performance of our general construction when instantiated for each of these function families. We use recent off-the-shelf implementations of Yao's garbled circuit protocol (along with OTs used in these implementations) and standard public-key and symmetric-key encryption algorithms (RSA-OAEP and AES). Our evaluations confirm that the schemes are highly practical.

We also evaluate the performance of our Superfast inner product C-FE scheme, which proves remarkably efficient (at the cost of not being function hiding) given that it involves only standard encryption and decryption operations.

Adversary model. The two most widely studied corruption models in literature are honest-but-curious and malicious[5]. Honest-but-curious parties follow their designated protocol diligently but try to learn the secrets of other parties from the interactions they have with them. On the other hand, malicious parties behave the way they like: not only they try to learn more information, but also execute any protocol of their choice in order to do so. In this work, our constructions provide security against honest-but-curious authorities and malicious clients.

Notation. We use κ to denote the security parameter. A function is negligible in κ (denoted $negl(\kappa)$) if it is smaller than the inverse of any polynomial, for all large enough values of κ. A probabilistic polynomial time algorithm, denoted in short by PPT , is an algorithm whose running time is bounded by some polynomial in κ on all inputs. This algorithm may use random coin tosses during its execution.

We use $\mathbb{Z}_N$ to denote the ring of integers modulo N (consisting of 0 and first $N-1$ positive integers with addition and multiplication defined modulo N). A vector of ℓ elements in this ring is denoted by $\vec{x} = (x_1, x_2, \ldots, x_\ell)$, where every $x_i \in \mathbb{Z}_N$. Encryption of a vector $\vec{x}$ means encryption of the concatenation of the elements of $\vec{x}$.

We use $[1, n]$ to denote the set $\{1, 2, \ldots, n\}$. For two strings a and b, $a \circ b$ denotes their concatenation and $a \oplus b$ denotes their bit-wise XOR. We denote the inner-product of two vectors $\vec{x}$ and $\vec{y}$ as $\langle \vec{x}, \vec{y} \rangle$

On using honest-but-curious third party. We assume the existence of an honest-but-curious third party (central authority) that is trusted not to collude with the

[4] In the multi-input variant, we can also support these functions when both arguments to the function are inputs from data owners; the performance will be essentially identical in the two cases.

[5] Honest-but-curious adversaries are also referred to as semi-honest or passive. Malicious adversaries are sometimes called active.

party computing a function (client). In general, the whole public key infrastructure (PKI) depends upon trusted third parties called certificate authorities (CAs). PKI is widely used for securing communication on Internet. CAs are trusted not to collude with the adversary, but still be secure enough against attacks. This type of non-collusion assumption is widely used in literature. Indeed, primitives like Identity-Based Encryption, Attribute-Based Encryption, etc., all involve a central authority trusted to not collude with other parties in the system. Nikolaenko et al. [113, 114] use similar non-collusion assumption where they trust the third party (garbled circuit generator) not to collude with the garbled circuit evaluator. Protocols for outsourcing multi-party computation [115, 116, 117, 118] use similar non-collusion assumptions to outsource evaluation of the garbled circuits.

5.2 Controlled Functional Encryption

In this section, we formally define controlled functional encryption (C-FE) and the security models under which we study it.

Our definition of C-FE differs from the definition of FE [11] in a crucial way. In FE, once a client receives a key for a function f from the authority, it can use the key to decrypt any number of ciphertexts it wants. However, in our *controlled* setting, every time a client wants to decrypt a ciphertext CT_x with f, a key is requested from the authority. The authority generates a *one-time* key which could only be used to compute $f(x)$. In order to decrypt a different ciphertext, a new key request must be submitted. Consequently, we have an additional algorithm KeyReq in our definition of C-FE below. Further, the authority will need to extract the policy parameter attached to CT_x from the key request, before it can decide whether to honor the request; for this, an algorithm Extract is used.

Syntax. We start by defining the syntax and (perfect) correctness requirement for the six algorithms that form a C-FE scheme. The role of these algorithms is illustrated in Figure 5.1.

Definition 5.2.1. *A controlled functional encryption (C-FE) scheme for a function family F_κ defined over $(\mathcal{F}_\kappa, \mathcal{X}_\kappa, \Lambda_\kappa)$ consists of six* PPT *algorithms (*Setup, Enc, KeyReq, Extract, KeyGen, Dec*) satisfying the following correctness condition for*

all $\kappa \in \mathbb{N}$, $x \in \mathcal{X}_\kappa$, $\lambda \in \Lambda_\kappa$ and $f \in \mathcal{F}_\kappa$. Consider the following experiment:

$$\begin{aligned}(\mathsf{MPK}, \mathsf{MSK}) &\leftarrow \mathsf{Setup}(1^\kappa)\\ \mathsf{CT} &\leftarrow \mathsf{Enc}(x, \lambda, \mathsf{MPK})\\ (\rho, \zeta) &\leftarrow \mathsf{KeyReq}(\mathsf{CT}, f)\\ (\lambda', \xi) &\leftarrow \mathsf{Extract}(\rho, \mathsf{MSK})\\ \tau &\leftarrow \mathsf{KeyGen}(\xi)\\ z &\leftarrow \mathsf{Dec}(\zeta, \tau)\end{aligned}$$

Then we require $\lambda' = \lambda$ (the authority receives the policy parameter correctly) and $z = F_\kappa(f, x)$ (the decryption yields the correct output) with probability 1.

Note that instead of writing $f(x)$, we have denoted it as $F_\kappa(f, x)$, where F_κ could be considered a function family, and f specifies which function in the family should be applied to x.

The above syntax does not explicitly accommodate a function-revealing or pattern-revealing C-FE scheme. In these variants, the output of Extract will contain not just λ, but also f and/or a unique identifier that can identify CT. To accommodate tweaks, we modify the definition of KeyGen to take a tweak $w \in \mathcal{W}_\kappa$ as an additional input, and change the final correctness requirement to $z = F_\kappa(f, x, w)$.

Security Definition. To cleanly capture the security guarantees of C-FE, we use an *ideal functionality*, in the spirit of Universally Composable (UC) security [119]. In UC security, ideal functionality is simply an ideal trusted party that captures the security guarantees: the various parties in the system (data owners, clients, authority) can learn only as much information as they can learn when interacting with this trusted party, and can influence the system only as much as the trusted party lets them. Our security definition is also along the lines of UC security, except that we consider an adversarial model in which the authority can only be corrupted passively (honest-but-curious) and only by itself (no collusion); while clients can be actively corrupted and may collude with each other.

Ideal World. We define the C-FE ideal functionality $\mathcal{F}$ that interacts with several "data owners," several "clients," and one "authority" as follows. (We omit routine details like initializing a session and how parties can join a session.) $\mathcal{F}$ can come in four modes of security, depending on whether it is function-hiding or

function-revealing, and whether it is pattern-hiding or pattern-revealing.

1. A data-owner can upload a pair (x, λ) to $\mathcal{F}$. Then $\mathcal{F}$ picks a handle h (a κ-bit random string) and sends it to all the clients. Internally, $\mathcal{F}$ records (h, x, λ) in a table.

2. A client can send an evaluation request (f, h) to $\mathcal{F}$. Then $\mathcal{F}$ proceeds as follows:

   (a) First, depending on its mode of security, $\mathcal{F}$ sends one of the following to the authority.

   - (f, h, λ), if function and pattern-revealing.
   - (h, λ), if function-hiding and pattern-revealing.
   - (f, λ), if function-revealing and pattern-hiding.
   - λ, if function and pattern-hiding.

   (b) Then it awaits for a command from the authority, either denying or allowing evaluation. In the former case, it sends a special symbol $\perp$ to the client. In the latter case, it receives a tweak w from the authority and sends $F(f, x, w)$ to the client.

Real World. In the "real world" execution, interaction with $\mathcal{F}$ is replaced by calls to a C-FE scheme, as follows. To initialize the system, the authority runs Setup first and publishes MPK (for the data owners). Instead of uploading (x, λ) to $\mathcal{F}$, a data owner would run Enc and privately communicate the resulting ciphertext CT to all the clients (in an implementation, an access controlled bulletin-board could be used to do this). Instead of sending an evaluation request to $\mathcal{F}$, a client would run KeyReq and send its output to the authority; the authority, instead of receiving λ (and possibly f and/or a handle for CT), uses Extract to obtain it. Then it can use an external decision process to decide whether or not to honor the key-request, and if it is to be honored, what the tweak value w should be. Then, instead of sending w to $\mathcal{F}$, the authority runs KeyGen to obtain a key that it sends to the client. Instead of receiving the output from $\mathcal{F}$, the client would compute it using Dec.

Definition 5.2.2. *A* C-FE *scheme is simulation secure if for every adversary* Adv *in the real world who actively corrupts a subset of clients or only passively corrupts the authority alone, there is a simulator* Sim *in the ideal world execution, which also corrupts only the same set of parties, and produces an output identically distributed to* Adv*'s output in the real world.*

A more precise definition refers to "environments" in which the execution — real or ideal — takes place; for more details, we refer the reader to [119] and subsequent formulations of UC security.

5.3 Constructions

In this section, we describe two C-FE schemes secure under the definitions discussed above. The first one is a simple and extremely efficient scheme for the inner-product functionality $\mathbf{F}_{\mathsf{IP}}$. It provides practical solutions to common problems like weighted average, hamming distance, etc. The second one is a general construction for any polynomial-time computable functionality. Though not as efficient as the first one, this construction is still practical for many problems of interest (as demonstrated by our experiments) and provides more security.

5.3.1 Inner Product

We first describe an efficient and simple way to compute inner-product between two vectors in our controlled functional encryption setting. Our scheme is secure against malicious clients and honest-but-curious authorities in the non-function hiding security model. Once again, note that we are able to compute the actual value of inner product, and not just whether it is zero or not.

Let $\mathbf{F}_{\mathsf{IP}} = \{f_{\vec{v}} \mid \vec{v} \in \mathbb{Z}_N^\ell\}$ denote the inner-product function family, where $f_{\vec{v}}(\vec{x}) = \langle \vec{v}, \vec{x} \rangle = \sum_{i=1}^{\ell} v_i \cdot x_i \mod N$ for all $\vec{x} \in \mathbb{Z}_N^\ell$. Note that to specify a function in this family, one only needs to provide the index $\vec{v}$. (For simplicity we have omitted the security parameter κ in the description.)

Overview. We first provide an overview of the construction. To encode an input vector $\vec{x}$, the data owner chooses a random vector $\vec{r}$ of the same dimension as $\vec{x}$ over $\mathbb{Z}_N$. It then outputs $(\vec{y}, \sigma)$, where $\sigma = Enc_{PK}(\vec{r})$ and Enc is a CCA2 secure PKE, and $\vec{y} = \vec{x} + \vec{r}$ (all operations over $\mathbb{Z}_N$). Note that $\vec{y}$ and $\vec{r}$ form an additive secret sharing of $\vec{x}$, and neither by itself contains any information about $\vec{x}$. The key-request computing $\langle \vec{x}, \vec{v} \rangle$ consists of $(\sigma, \vec{v})$. The authority will decrypt σ to obtain $\vec{r}$ and returns $\langle \vec{v}, \vec{r} \rangle$ to the client. The client locally computes $\langle \vec{v}, \vec{y} \rangle$ and adds it to the negative of the value obtained from the authority, to obtain

$$\langle \vec{v}, \vec{y} \rangle - \langle \vec{v}, \vec{r} \rangle = \langle \vec{v}, \vec{x} \rangle.$$

Even if the client sends the same σ to the authority several times, it is easy to show that the client's view can be simulated perfectly in an ideal world, where the client only obtains $\langle \vec{v}, \vec{x} \rangle$ for the values of $\vec{v}$ it sent to the authority. As in the general construction, using CCA2 secure encryption ensures that a malicious client obtains no advantage by sending a σ it did not obtain from a data owner.

Note that the authority's computation here involves a decryption, and an inner product computation, and the client's computation involves merely an inner product computation. Further, this scheme can exploit the sparsity of $\vec{v}$, i.e. the client's computation is proportional to the number of non-zero elements in $\vec{v}$. In fact, even if all vector elements are non-zero, our performance evaluations show that this construction is extremely fast for genomic-scale inputs.

Construction. A (function and pattern revealing) C-FE scheme Π_{IP} for the function family $\mathbf{F}_{\mathsf{IP}}$ is presented in Figure 5.2 and a proof of security is described below. Correctness of Π_{IP} follows easily from construction and the linearity of dot product:

$$\langle \vec{v}, \vec{y} \rangle - \tau = \langle \vec{v}, \vec{x} + \vec{r} \rangle - \langle \vec{v}, \vec{r} \rangle = \langle \vec{v}, \vec{x} \rangle.$$

Note that we can allow the authority to add noise to the outcome using a tweak: $\mathsf{KeyGen}_{\mathsf{IP}}$ will take the noise w as an additional input and set $\tau = \langle \vec{v}, \vec{r} \rangle - w$, so that the outcome obtained by the client is $\langle \vec{v}, \vec{x} \rangle + w$.

Protocol Π_{IP}

- $\mathsf{Setup}_{\mathsf{IP}}(1^\kappa)$: Run $\mathsf{Setup}_{\mathsf{CCA2}}(1^\kappa)$ to obtain $(\mathsf{PK}, \mathsf{SK})$. Set MPK and MSK to be PK and SK respectively.
- $\mathsf{Enc}_{\mathsf{IP}}(\vec{x}, \lambda, \mathsf{MPK})$: Choose ℓ numbers $r_1, r_2, \ldots, r_\ell$ uniformly at random from $\mathbb{Z}_N$. Let $\vec{r} = (r_1, r_2, \ldots, r_\ell)$. Output the ciphertext $\mathsf{CT} = (\vec{x} + \vec{r}, \mathsf{Enc}_{\mathsf{CCA2}}(\vec{r}, \lambda, \mathsf{MPK}))$.
- $\mathsf{KeyReq}_{\mathsf{IP}}(\mathsf{CT}, \vec{v})$: Let $\mathsf{CT} = (\vec{y}, \sigma)$. Output $\rho = (\sigma, \vec{v})$ and $\zeta = (\vec{v}, \vec{y})$.
- $\mathsf{Extract}_{\mathsf{IP}}(\rho, \mathsf{MSK})$: Let $\rho = (\sigma, \vec{v})$. Run $\mathsf{Dec}_{\mathsf{CCA2}}(\sigma, \mathsf{MSK})$ to obtain $(\vec{r}, \lambda)$. Output $((\lambda, \sigma), \xi)$ where σ is used to reveal the pattern, and $\xi = (\vec{v}, \vec{r})$.
- $\mathsf{KeyGen}_{\mathsf{IP}}(\xi)$: Let $\xi = (\vec{v}, \vec{r})$. Output $\tau = \langle \vec{v}, \vec{r} \rangle$.
- $\mathsf{Dec}_{\mathsf{IP}}(\zeta, \tau)$: Let $\zeta = (\vec{v}, \vec{y})$. Output $\langle \vec{v}, \vec{y} \rangle - \tau$.

Figure 5.2: Superfast Construction for computing actual inner product.

Proof of Security. A simple simulator, combined with the CCA2 security of the encryption scheme can be used to prove the security. The interesting case is when

a client is corrupt. To translate the ideal world view to the real world view of the client, we consider the following simulator.

First, the simulator picks a pair of keys $(\mathsf{MPK}, \mathsf{MSK})$ to simulate the setup. Then, when it receives a handle h, it creates a simulated ciphertext $\mathsf{CT}' = (\vec{y_h}, c_h)$, where $\vec{y_h}$ is a random vector (which is identically distributed as $\vec{x} + \vec{r}$ in the real ciphertext), and $c_h \leftarrow \mathsf{Enc}_{\mathsf{CCA2}}(0^{|(x,\lambda)|}, \mathsf{MPK})$ is a ciphertext encrypting a string of zeros (which will be indistinguishable from the encryption of (x, λ)). Later, if the client sends out a key-request of the form $(\sigma, \vec{v})$, the simulator checks if $\sigma = c_h$ for some handle h. There are two cases:

1. $\sigma = c_h$ for some h: Then the simulator forwards an evaluation request $(h, \vec{v})$ to the ideal functionality, which will return $z = \langle \vec{v}, \vec{x} \rangle$. The simulator creates a simulated value $\tau' = \langle \vec{v}, \vec{y_h} \rangle + z$. In this case, τ' is *identically distributed* as the value τ the client receives in the real execution.

2. There is no h s.t. $\sigma = c_h$: In this case the simulator acts like the authority. i.e., It decrypts σ and (if the decryption is valid) obtains some vector $\vec{r}$; then it returns $\tau' = \langle \vec{v}, \vec{r} \rangle$.

In the second case above the ciphertext σ sent by the client to the authority does not correspond to any (simulated) ciphertext it received (from any simulated data owner). However, the client could have created σ in an arbitrary fashion, without necessarily encrypting a value r that it knows. To argue that the simulation is indistinguishable from the real execution, we need to argue that the dummy ciphertexts c_h created by the simulator remain indistinguishable from real ciphertexts, even though the simulator carries out decryptions of arbitrary ciphertexts σ that the adversary presents (other than the dummy ciphertexts themselves). This is possible, thanks to the CCA2 security of the encryption scheme: a distinguisher between the real execution and the simulation can be turned into an adversary that distinguishes the encryptions of real messages and dummy messages, with the help of a decryption oracle (to which it never sends challenge ciphertexts).

5.3.2 General construction

In this section, we construct a controlled FE scheme Π for any polynomial-time computable family of functions $\mathcal{F}$, which take inputs from the domain $\mathcal{X}$. Without loss of generality, we assume that an element $f \in \mathcal{F}$ can be represented by ℓ bits, and an element $x \in \mathcal{X}$ can be represented by k bits, where both ℓ and k are some

polynomial in κ.

Overview. We start by sketching an intuitive construction, which is neither secure nor efficient enough, and then describe how to fix these issues. For simplicity, we start with a function revealing, pattern revealing construction, with no tweaks, for general function evaluation. That is, we are given an arbitrary (efficiently computable) function F so that the client should be able to compute $F(f, x)$, if the authority, after seeing λ (and pattern information), allows the client to do so, where $(x; \lambda)$ is a piece of data and associated policy parameter from a data owner, and f is a function specification from the client.

First, the authority runs a Setup algorithm: it picks an encryption-decryption key pair $(\mathsf{MPK}, \mathsf{MSK})$ for a public-key encryption scheme, and publishes the public key. The encryption algorithm used by the data owner takes (x, λ) and creates two ciphertexts (α, σ), obtained by encrypting (x, λ) respectively, under PK. The client will receive (α, σ) and when it wants to evaluate $f(x)$, it sends a key-request to the authority consisting of (f, σ). The authority can recover λ by decrypting σ; it can also update the pattern information, if necessary, by comparing λ with previous key requests it received. Then, if it decides to honor the key-request, it engages in a one-round 2-party secure computation (using garbled circuits and a one-round OT protocol, for instance) to compute the following function F'. F' takes α as input from the client and (f, SK) as input from the authority, and outputs $F(Dec_{SK}(\alpha), f)$ to the client. Note that $Dec_{SK}(\alpha) = x$. This would fit the syntax of C-FE, if the key-request includes the first message from the client to the authority in the secure computation protocol. As for security, since a secure computation protocol is used, the client should learn nothing other than $F(f, x)$.

There are two major problems with this solution:

1. Firstly, it is not secure against malicious clients. In particular, suppose the client feeds not α, but a related ciphertext α' to the secure computation protocol. Then the client can potentially learn $F(x', f)$ where $x' = Dec_{SK}(\alpha')$ is related to x.

2. Secondly, the above solution has a serious drawback in terms of efficiency. Often F is a very simple function (like inner product or Hamming distance), and can be very efficiently implemented using a 2-party secure computation protocol. However, the secure computation protocol used above is not for evaluating F, but for evaluating F'. Note that F' involves a (public-key)

decryption operation. This decryption applies to a ciphertext of the entire input x. This makes the scheme vastly inefficient and often impractical.

The first problem is easy to fix. To thwart all such malleability attacks we can use a CCA2 secure public-key encryption scheme. (One should also bind α and σ together using a random nonce, so that the client cannot replace the policy of one data owner with that of another; our final solution will not have this structure, and hence will not need the use of a random nonce.)

To address the second problem, we need to ensure that the secure computation is for F itself, and not a function like F'. To achieve this, we take a closer look at how a garbled circuit based 2-party computation protocol proceeds. The client can evaluate a garbled circuit only for an input for which it holds the requisite "labels": each bit position of input has two labels associated with it, corresponding to the values 0 and 1, that are picked at random by the garbled circuit generator. To let a client evaluate the circuit on a k-bit input x (belonging to the generator) that the client does not know, the generator sends the k labels corresponding to x, without revealing whether each label corresponds to value 0 or 1.

In our case, unfortunately, the garbled circuit is generated by the authority who also does not know x, and it will not be able to send just the relevant labels. However, it can take the help of the data owner (who is offline), as follows.

Recall that the data owner encodes x as (α, σ) where α is given to the client and σ to the authority. σ will contain a set of $2k$ keys for symmetric-key encryption (SKE), encrypted under the authority's public-key. The authority will encrypt *all the* $2k$ *labels* (2 per bit position of x) under these keys. Further, the message inside σ will also specify a random order for the 2 encrypted labels for each bit position. The authority will send all the encrypted labels to the client in this order. Note that σ contains no information about x itself. Now, the client needs to recover only the k labels corresponding to x from these $2k$ encrypted labels. For this, α will include k SKE keys (in the clear), one out of the two keys for each bit position, corresponding to the bit value of x at that position. α will also indicate, for each bit position, whether the given key corresponds to the first or the second one in the pair of encrypted labels that will be sent to the client by the authority. This allows the client to decrypt exactly the k labels corresponding to the bit values of x. Note that the client's view too does not contain any information about x, since which one in a pair of encrypted labels corresponds to the bit value 0 and which one to 1 is not known to the client.

We have another construction which is actually a slight optimization of the above scheme, that avoids the $2k$ encryptions by the authority and the k decryptions by the client, by having σ and α specify the labels themselves. The authority will pick all other labels for the garbled circuit freshly, but the $2k$ labels corresponding to the input wires for x remain the same.

Note that when f is known to the authority, the only labels that the authority cannot send directly to the client are those for x. In this case, the entire scheme is based only on public and symmetric-key encryption schemes. However, if we require function hiding, the authority will need to transfer the correct labels corresponding to f via oblivious-transfer.

The variants can be easily accommodated in this construction. Firstly, the message in the ciphertext σ can also contain the policy parameters λ associated with a piece of data x. Tweaks are simply additional inputs to F that the authority can hard-wire into the circuit, while creating the garbled circuit. To allow pattern hiding, we replace CCA2 secure encryption with Homomorphic Encryption with CCA security [120].

Construction. For the sake of simplicity, we ignore the policy parameter λ in the construction; this lets us merge the algorithms $\mathsf{Extract}$ and KeyGen (ignoring the need to extract λ). Let $(\mathsf{Setup}_{\mathsf{SKE}}, \mathsf{Enc}_{\mathsf{SKE}}, \mathsf{Dec}_{\mathsf{SKE}})$ be a symmetric key encryption scheme which generates keys of length κ and encrypts κ-length messages. Also, let Π_{OT} be a one-round oblivious transfer protocol where the first message is sent from chooser to sender and the second message from sender to chooser (for more details, see the paragraph "oblivious transfer" in Chapter 2). Using these tools along with others, we present a formal construction of Π in Figure 5.3. The two messages (from the client to the authority, and back) in the OT protocol are combined with key-request and key-generation algorithms, so that the syntax of C-FE is respected. A proof of security is given below. We also present an alternate construction, which is slightly more efficient, in Appendix 5.4.

Proof of security. We provide a sketch of the proof of security of Π. Once again, the interesting case is when a client is malicious. Let $\mathsf{Sim}_{\mathsf{GC}}$ be the simulator $\mathcal{S}$ of the projective prv.sim secure garbling scheme[6] with circuit being the side information i.e. $\mathsf{PrvSim}_{\mathcal{G},\phi,\mathcal{S}}$ where $\phi(f) = f$, as described in [19]. We construct a simulator Sim which uses $\mathsf{Sim}_{\mathsf{GC}}$ to simulate the view of a corrupt client Adv in the

[6]The simulation based garbling schemes (prv.sim) and indistinguisbility based garbling scheme (prv.ind) schemes of [19] are equivalent in our setting.

Protocol Π

- $\mathsf{Setup}(1^\kappa)$: Run $\mathsf{Setup}_{\mathsf{CCA2}}(1^\kappa)$ to obtain $(\mathsf{PK}, \mathsf{SK})$. Set MPK and MSK to be PK and SK respectively.

- $\mathsf{Enc}(x, \mathsf{MPK})$: For $i \in [1, k]$ and $b \in \{0, 1\}$, run $\mathsf{Setup}_{\mathsf{SKE}}(1^\kappa)$ to obtain a key r_i^b. Let $\hat{r} = r_1^0 \circ r_1^1 \circ r_2^0 \circ r_2^1 \ldots \circ r_k^0 \circ r_k^1$. Choose a uniformly random k-bit string v, and let $u = x \oplus v$. Now, let r_u denote $r_1^{u_1} \circ r_2^{u_2} \ldots r_k^{u_k}$, where u_i is the ith bit of u. Output the ciphertext $\mathsf{CT}_x = ((r_u, u), \mathsf{Enc}_{\mathsf{CCA2}}(\hat{r} \circ v, \mathsf{MPK}))$.

- $\mathsf{KeyReq}(\mathsf{CT}, f)$: Let $\mathsf{CT} = (\alpha, \sigma)$ and $f = (f_1, f_2, \ldots, f_\ell)$. For $j \in [1, \ell]$, run the first step of the oblivious transfer protocol Π_{OT} with chooser's input being f_j. Let M_j be the first message output by this protocol and R_j be the coin tosses used. Now, let M denote $(M_1, \ldots, M_\ell)$ and R denote $(R_1, \ldots, R_\ell)$. Output $\rho = (\sigma, M)$ and $\zeta = (\alpha, f, M, R)$.

- $\mathsf{KeyGen}(\rho = (\sigma, M), \mathsf{MSK})$: Run $\mathsf{Dec}_{\mathsf{CCA2}}(\sigma, \mathsf{MSK})$ to obtain $(r_1^0, r_1^1), \ldots, (r_k^0, r_k^1)$ and v. Consider the circuit C which takes as input a k-bit string z and an ℓ-bit function g, and computes $F(g, z)$. Construct a projective $\mathsf{PrvSim}_{\mathcal{G}, \phi(f)=f, \mathcal{S}}$ garbled version of this circuit as described in [19] and call it $\hat{C}$, choosing keys at random for each and every wire of the circuit, including input wires. Let the key pairs corresponding to the k-bit string z be $(t_1^0, t_1^1), \ldots, (t_k^0, t_k^1)$. For $i \in [1, k]$ and $b \in \{0, 1\}$, run $\mathsf{Enc}_{\mathsf{SKE}}(t_i^b, r_i^{b \oplus v_i})$ to obtain a ciphertext $c_i^{b \oplus v_i}$, where v_i is the ith bit of v.

  Let the key pairs corresponding to the ℓ-bit function input g be $(s_1^0, s_1^1), \ldots, (s_\ell^0, s_\ell^1)$. Parse M as $(M_1, \ldots, M_\ell)$. For $j \in [1, \ell]$, run Π_{OT} with sender's input being (s_j^0, s_j^1) and message received from chooser being M_j. Let M'_j be the second message output by this protocol. Let M' denote $(M'_1, \ldots, M'_\ell)$. Output $\tau = (\hat{C}, (c_1^0, c_1^1, \ldots, c_k^0, c_k^1), M')$.

- $\mathsf{Dec}(\zeta, \tau)$: Parse τ as $(\hat{C}, (c_1^0, c_1^1, \ldots, c_k^0, c_k^1), (M'_1, \ldots, M'_\ell))$ and ζ as $((r_u, u), f, (M_1, \ldots, M_\ell), (R_1, \ldots, R_\ell))$. For $j \in [1, \ell]$, run Π_{OT} with chooser's input f_j, coin tosses R_j, first round message M_j, and second round message M'_j, to obtain the key for the jth bit of g. Parse r_u as $r_{u,1} \ldots r_{u,k}$. To find the key for the ith bit of z, run $\mathsf{Dec}_{\mathsf{SKE}}(c_i^{u_i}, r_{u,i})$ to obtain $t_i^{x_i}$. Now, evaluate $\hat{C}$ to obtain the value of $f(x)$.

Figure 5.3: General C-FE Construction

ideal world.

Sim runs Adv internally as a black-box. He first executes $\mathsf{Setup}(1^\kappa)$ to obtain $(\mathsf{MPK}, \mathsf{MSK})$, and sends MPK to Adv. When he receives a handle h from the ideal functionality, he chooses $(r_1^0, r_1^1), \ldots, (r_k^0, r_k^1)$ and v at random. Let $\hat{r} = r_1^0 \circ r_1^1 \circ r_2^0 \circ r_2^1 \ldots \circ r_k^0 \circ r_k^1$, and $\alpha = (r_1^0 \circ r_2^0 \ldots r_k^0, 0^{|x|})$. Sim provides (α, σ) as ciphertext to Adv, where $\sigma = \mathsf{Enc}_{\mathsf{CCA2}}(\hat{r} \circ v, \mathsf{MPK}))$.

When Adv initiates a key request $\rho = (\sigma', M)$, Sim first extracts an f from M

(note that since Π_{OT} is a UC-secure protocol, this can be done). He then checks whether $\sigma' = \sigma$ for some h or not. If $\sigma' \neq \sigma$ for any h, Sim simply runs KeyGen with $(\sigma', M, \mathsf{MSK})$, and returns the output to Adv. In case there is equality for some h_x, he sends f and h_x to Eval, and obtains $F(f, x)$. He now invokes $\mathsf{Sim}_{\mathsf{GC}}$ with inputs $(f, F(f, x))$ to obtain a fake garbled circuit $\hat{C}_{\mathsf{Fake}}$. Having obtained the circuit, Sim runs the rest of KeyGen with $(r_1^0, r_1^1), \ldots, (r_k^0, r_k^1)$ and v (decrypted value of σ) and returns $(\hat{C}_{\mathsf{Fake}}, (c_1^0, c_1^1, \ldots, c_k^0, c_k^1), M')$ to Adv. This completes the description of Sim.

Note that in the ideal world, when Sim receives a key request with σ', he generates a fake garbled circuit (GC) as opposed to a real circuit (if Sim creates a real GC, Adv would evaluate it to recover $f(0^{|x|})$, and distinguish the two worlds). However, a fake GC has the property that no matter what combination of keys a party uses for the input wires of the bits of x, the circuit always evaluates to $F(f, x)$. Therefore, even if Adv uses keys corresponding to $0^{|x|}$ to evaluate $\hat{C}_{\mathsf{Fake}}$, it will still recover $F(f, x)$. Hence, it cannot distinguish between a fake and a real GC.

Correctness follows easily from construction.

5.4 Alternate General Construction

Before a formal description, we give some intuition about the construction. At a high-level, our plan is to let the client use a garbled circuit generated by the authority to evaluate the function on the input x. The circuit in question evaluates the function F, which takes x and f as inputs (and optionally, a tweak w, which we ignore for simplicity). f is known to the client, and in the function-revealing case, to the authority as well. But note that neither the client nor the authority knows the input x (which was generated by a data owner).[7] Thus, it is not clear how a garbled circuit for F can be used in our setting.

In our solution, the data owner will arrange for the client to have the labels corresponding to the input x, without either the client or the authority knowing x. The key idea is that the authority does not pick all the labels for all the wires in the garbled circuit itself. Instead, the data owner would specify the labels used

[7] Further, as we consider malicious client which may attempt to alter the input on which the garbled circuit is evaluated, which rules out a simple solution in which x is kept secret-shared between the client and the authority (unless, cryptographic operations are incorporated into the garbled circuit).

for the input wires corresponding to input x. More precisely, suppose x is k bits long. Then the data owner picks $2k$ random labels $\{r_0^i, r_1^i\}_{i=1}^k$ and encrypts them (using a CCA2 secure public-key encryption scheme) for the authority. The encryption is required because it is important that these labels are not all known to the client. During the key-request, the client is expected to send this part of the ciphertext to the authority. While a garbled circuit is generated, for each wire u in the circuit, two freshly chosen random labels $(R_0^{(u)}, R_1^{(u)})$ are required; but for the wires corresponding to the input x_i (i.e., the i^{th} bit of x), the authority uses the pair (r_0^i, r_1^i) instead. (Labels for all the other wires are picked freshly.) Now, the data owner knows both x and the labels for the input wires used in the garbled circuit. So it can simply provide the correct labels to the client as part of the encryption of x. More precisely, the client will be given the k labels $r_1^{x_1}, \ldots, r_k^{x_k}$. On obtaining the garbled circuit, it simply evaluates the garbled circuit using these labels for the wires corresponding to x.

We remark that typically, it is important that a garbled circuit is not reused – i.e., evaluated on different inputs. Our solution could be viewed as a safe way of reusing *parts* of a garbled circuit. In particular, if different functions are evaluated on the same input x, the same labels can be used for x. (In a standard 2-party computation, this observation could be used to replace multiple OT protocol invocations, with a single OT protocol; in our case, since the data owner is offline, this property is crucial.)

5.5 Implementation and Evaluation

We implemented our C-FE constructions: general as well as Superfast construction. Experiments were conducted on synthetic data, however, size of the data was inspired by the applications discussed in Section 5.6. We evaluated our general construction on a powerful machine but used a laptop to evaluate our Superfast inner-product construction.

5.5.1 Superfast Inner-Product Construction

Our Superfast inner-product construction is implemented in Java. We use RSA-OAEP [121] (RSA-Optimal Asymmetric Encryption Padding) implementation

Protocol Π'

Since Π' is a slight modification of Π, we only describe what changes need to be made to the algorithms of Π in order to obtain the ones for Π'.

- $\mathsf{Setup}'(1^\kappa)$: Stays the same as Setup.
- $\mathsf{Enc}'(x, \mathsf{MPK})$: Randomly pick $2k$ bit strings of length κ each. Let (r_i^0, r_i^1) denote the ith pair among them, where $1 \leq i \leq k$. Let $r_x = r_1^{x_1} \circ r_2^{x_2}, \ldots \circ r_k^{x_k}$, where x_i denotes the ith bit of x. Also, let $\hat{r} = r_1^0 \circ r_1^1 \circ r_2^0 \circ r_2^1 \ldots \circ r_k^0 \circ r_k^1$ be the concatenation of all the randomly chosen strings (in the specified order). Output the ciphertext $\mathsf{CT}_x = (r_x, \mathsf{Enc}_{\mathsf{CCA2}}(\hat{r}, \mathsf{MPK}))$.
- $\mathsf{KeyReq}'(\mathsf{CT}, f)$: Stays the same as KeyReq.
- $\mathsf{KeyGen}'(\rho = (\sigma, M), \mathsf{MSK})$: Run $\mathsf{Dec}_{\mathsf{CCA2}}(\sigma, \mathsf{MSK})$ to obtain $(r_1^0, r_1^1), \ldots, (r_k^0, r_k^1)$. Consider a circuit C which takes as input a k-bit string z and an ℓ-bit function g, and computes $F(g, z)$. Construct a garbled circuit $\hat{C}$ for C, but with (r_i^0, r_i^1) as keys for the input bit z_i ($i \in [1, k]$). All the other keys required for creating the garbled circuit are chosen at random. M' is computed in the same way as KeyGen. Output $\tau = (\hat{C}, M')$.
- $\mathsf{Dec}'(\zeta, \tau)$: Key for the jth bit of g ($j \in [1, \ell]$) is obtained in the same way as described in Dec. On the other hand, key for the ith bit of z is part of ζ. Now, evaluate $\hat{C}$ to obtain the value of $f(x)$.

Figure 5.4: Alternate C-FE Construction

of the Java Cryptography Extension (JCE) and evaluated the construction with 1024-bit, 2048-bit and 4096-bit keys.

Experiments. Our Superfast inner-product construction is very light, so we used a laptop – with Intel Core i7 3615QM processor, 8GB memory and Mac OS X 10.9 – for the experiments.

Vector sizes. We used data vector of size 4,000,000[8] integers (4-byte) and varied the function vector size according to the different applications discussed in Section 5.6. We also used data vectors of size 40,000,000[9] integers for the evaluation.

Encryption in Superfast scheme has two parts: additive secret sharing of plaintext's field elements and public key-encryption of one share of each plaintext element. We concatenate several vector elements to be encrypted in one block to avoid blow up in the ciphertext size. Ideally, size of ciphertext in our scheme should be double the size of plaintext due to additive secret sharing. As, we use

[8]The number is based on the fact that each human has 4,000,000 variants.

[9]4 million variants in each individual can appear at 40 million different locations.

RSA-OAEP, padding makes size of the ciphertext little bit more than double. Per block padding size is same for different key sizes, but, as larger keys have larger block sizes, the ciphertext size decreases as the key size increases, as shown in Table 5.1.

Plaintext size (4-byte alphabet)	4,000,000			40,000,000
Plaintext size(MB)	15.26			152.59
Key size (bits)	1024	2048	4096	1024
Ciphertext size(MB)	38.51	33.68	31.95	385.10
Encryption time(s)	12.86	15.70	24.81	126.04

Table 5.1: Encryption time and ciphertext size of Superfast C-FE scheme: As, encryption time and ciphertext size in our scheme depend only on the plaintext size, we present it separately from Table 5.2. Experiments were performed on a **laptop** with Intel Core i7 3615QM processor, 8GB memory and Mac OS X 10.9. Each measurement is an average of 10 runs. We evaluated 4,000,000 and 40,000,000

Key size (bits)	Key generation				Decryption
	Function vector size (4 byte alphabet)	Time (s)	Data sent (KBytes)	Data received (Bytes)	Time (ms)
1024	1,000	1.21	132.46	8	0.07
2048	1,000	6.55	256.21	8	0.07
4096	1,000	42.52	501.26	8	0.07
1024	10,000	11.66	1295.96	8	0.04
2048	10,000	61.95	2422.85	8	0.04
4096	10,000	377.23	4414.68	8	0.04
1024	20,000	22.37	2529.26	8	0.07
2048	20,000	116.83	4551.33	8	0.08
4096	20,000	658.67	7732.90	8	0.17
1024	4,000,000	226.19	55059.63(=53.77MB)	8	20.58
2048	4,000,000	495.31	50118.00(=48.94MB)	8	22.81
4096	4,000,000	1512.44	483444.50(=47.21MB)	8	23.03
1024	40,000,000	1947.32	240209.11(=234.60MB)	8	19.90

Table 5.2: Performance evaluation of our Superfast inner-product construction: Experiments were performed on a **laptop** with Intel Core i7 3615QM processor, 8GB memory and Mac OS X 10.9. Each measurement is average of 10 runs.

We present the performance of Superfast scheme in Table 5.1 and Table 5.2. Function vector is chosen randomly to account for worst case scenarios. As, we pack several secret shares in each public key encryption by concatenating them in a single block, sequential function vectors would result in a very small number of public key decryptions at the authority for key generation. Our plaintext has millions of elements, so, very small random function vectors (e.g., with 1000 elements) would result in number of public key decryptions same as the size of the

function vector. But, as the function vector size increases the number of public key decryptions decreases. The maximum possible size of the function vector is when it is equal to the size of plaintext. As, shown in Table 5.2, this case is much efficient than small function vectors. Note that in practical scenarios function vectors are not random and our scheme will perform much better than this analysis. The key request message size depends upon the function vector size. Function key size is constant in our scheme. Decryption stage is very efficient and constitutes simple additions and multiplications and only depends upon the plaintext size.

5.5.2 General C-FE scheme

Our general construction is based on Yao's garbled circuits. We use FastGC [122] for Yao's garbled circuits implementation. Our general C-FE construction also requires hybrid encryption, which we implement using AES (with 128-bit key) and RSA-OAEP (with 4096-bit key). To implement hybrid encryption we use Java Cryptography Extension (JCE). For AES, we use AES counter-mode implementation of JCE, and for RSA we use RSA-OAEP implementation of JCE. RSA-OAEP is a non-malleable encryption scheme secure against IND-CCA2 attacks in the random oracle model [121]. Moreover, our prototype is single-threaded. We implement the alternate construction (**with function hiding**) described in Appendix 5.4, which is slightly more efficient. This construction requires minor modifications to the garbled circuit implementation, so in principle it is very easy to reproduce our results. We require extra code for *(i)* hybrid encryption, and *(ii)* partial reuse of the wire labels.

Experiments. We tested our general C-FE construction on a Dell PowerEdge R720 computer with dual Intel Xeon E5-2670 (2.60GHz) processors (computer has 16 cores in total, but our implementation is sequential) and 128GB of memory. Scientific Linux 6.3 (kernel version: 2.6.32) was installed on the computer. We allocated 30GB heap memory for the garbled circuit generator and another 30GB heap memory for the garbled circuit evaluator. We note that FastGC Java implementation is memory intensive and even with 30GB heap memory (maximum allowable by JVM is less than 32GB), we ran out of memory. Kreuter et al. introduced PCF (Portable Circuit Format) framework which is much better in terms of memory consumption, partially because PCF is written in C++ [23]. Both generator and evaluator programs were executed simultaneously on the same machine. Amount

Problem	Encryption		Keygen				Decryption
	Time(ms)	Time(ms)	Decryption(ms)		GC Gen(s)	Comm	GC Eval(s)
	1024	4096	1024	4096	Offline[a], **Online**		Offline[a],**Online**
Hamming 10,000bits	2.87	3.74	5.56	59.30	3.81, **0.30**	919.09KB	2.19, **0.31**
Hamming 16,000bits	4.15	5.15	7.27	61.09	3.95, **0.47**	1470.61KB	2.70, **0.45**
Hamming 20,000bits	5.16	6.09	8.10	61.88	6.19, **0.55**	1838.46KB	4.70, **0.52**
Hamming 60,000bits	14.88	15.66	18.12	72.92	11.15, **1.52**	5515.99KB	88.34, **0.44**
Hamming 1,500,000bits	364.33	379.27	408.94	453.59	469.78, **45.81**	135MB	426.50, **44.26**
Levenshtein 100x100 (2-bit alphabet)	0.35	1.52	3.75	57.22	1.47, **3.47**	10725.26KB	0.59, **3.50**
Levenshtein 1000x1000 (2-bit alphabet)	0.90	1.75	3.96	55.87	1.57, **498.38**	1534MB	0.64, **498.40**
Levenshtein 20000x50 (2-bit alphabet)	0.36	2.46	1.00	55.80	1.64, **650.73**	2098MB	0.70, **650.45**
Levenshtein 20000x200 (2-bit alphabet)	0.35	1.10	2.17	55.31	1.24, **2714.76**	8402MB	0.80, **2714.75**
SmithWaterman 50x50 (32-bit alphabet)	0.35	1.05	2.55	56.03	2.47, **197.51**	580MB	1.58, **197.54**
AES (16 Bytes)	0.50	1.05	3.73	56.91	1.50, **0.17**	306.33KB	0.64, **193.10**
Dot Product 100x100 (8- & 2-bit alphabets)	0.34	1.04	2.62	55.55	2.62, **51.80**	61.78KB	1304, **77.40**
Dot Product 1000x1000 (8- & 2-bit alphabets)	0.79	1.47	2.94	56.09	60.59, **0.29**	617.76KB	592.51, **0.33**
Dot Product 2500x2500 (8- & 2-bit alphabets)	1.70	2.24	4.12	57.15	531.46, **1.08**	1700.62KB	686.71, **1.11**

Table 5.3: Performance evaluation of our general Controlled Functional Encryption (C-FE) with **function-hiding** construction: Experiments were performed on a Dell R720 computer with dual Intel Xeon E5-2670 (2.60GHz) processors, 128GB of memory and Scientific Linux 6.3. Each measurement is average of 10 runs. Note the negligible performance overhead due to hybrid encryption on top of plain garbled circuits. Evaluator's computation is only $\frac{1}{4}$th of the generator's computation. Both evaluator and generator are taking approx. the same time, because due to pipelining, evaluator is waiting for the partial circuit to be generated and delivered before it can evaluate it. 1024 and 4096 in the third row of the table shows the public key sizes (in bits) used for hybrid encryption.

[a]The offline cost shown in the table is a **one-time cost** and is only incurred for the first time and does *not* need to be repeated for every computation of the same function. It can be done well before the online phase.

of data transfered between generator and evaluator was recorded to report network bandwidth usage.

We evaluated performance of our system on very large problem instances. As shown in Table 5.3 performance of our scheme is negligible over plain garbled circuits. We evaluated the general scheme with *function hiding* capability. It is clear from Table 5.3 that our scheme has negligible overhead on top of plain garbled circuits. Note that **Offline** computation time is a one-time cost that's incurred first time the computation is done and there is no computation cost when the same computation is repeated again. The offline computation includes generating plaintext digital circuit from the code.

Hamming distance. We evaluated our schemes with Hamming distance problems of size upto 1.5 million bits. Most of the commercial services such as 23andme provides human SNP profile with 0.5 million SNPs (recently they started to provide 1 million SNPs). To compare two SNP profiles, direct Hamming distance doesn't work as each SNP is of two bits. We designed the following simple encoding after which we can use Hamming distance to compute the similarity: SNP can have value of either 0, 1 or 2, we represent 0 as 001, 1 as 010 and 2 as 100. After this encoding, computing Hamming distance will give us number of common SNPs between two SNP profiles. As, we represent each SNP with 3 bits, for a 0.5 million SNP profile, we require 1.5 million bits. Therefore, we conducted experiments with 1.5 million bits and our results shows that it requires less than 8 minutes and 135MB network communication to find similarity between two SNP profiles of 0.5 million SNPs each (using our encoding scheme). Note that our Superfast scheme can also be used for computing Hamming distance, but it doesn't provide function hiding, the general scheme provides function hiding but takes more time.

Levenshtein Distance. Levenshtein distance is also a commonly used similarity measure and is computationally much more involved than Hamming distance. Levenshtein distance is computed using dynamic programming algorithm. We ran relatively large problem instance; finding Levenshtein distance between 20,000 and 200 letters strings (from a 2-bit alphabet). Levenshtein distance requires a lot of time and bandwidth but this is due the inefficiency of underlying garbled circuits.

Inner-product. We implemented inner-product functionality into FastGC family to compare the performance of our Superfast C-FE scheme and general C-FE scheme. We used simple modular multiplier circuit [123] to realize multipliers for our inner-product implementation. Inner-product computation in our general

model incurs large one-time cost, but is efficient in the online stage. As, can be seen in the last row of Table 5.3, inner-product have reasonable computation and communication cost.

Other problems: We also tested our general function C-FE scheme on other problems such as AES and SmithWaterman score. Results are shown in Table 5.3.

5.6 Applications

Personalized Medicine. Personalized medicine is a revolutionary concept in healthcare. Different from a "one-size-fits-all" approach, it enables physicians to prescribe medicine based on the patients' genomic build-up. Several cryptographic protocols have been proposed for personalized medicine [124, 125]. These protocols are inefficient and incur very high computation and communication costs. Recently, additive homomorphic encryption based protocols have been proposed [126, 127, 128]. These schemes are relatively efficient, but they are very interactive. They require the patient to be online and possess a computer that will be used during a disease susceptibility test. This makes them more difficult to deploy in practice. More seriously, patient computers could get compromised, resulting in leaking their genomic data. We show that our Superfast C-FE schemes can be used to achieve a much more practical scheme: it doesn't require any direct interaction with the patient[10], is much more computationally and storage efficient, and securer as we are using non-malleable public-key encryption as opposed to homomorphic encryption which allows the ciphertext to be arbitrarily modified.

Using our technique, DNA is first digitized through sequencing or genotyping by an external agency. This sequencing agency can encrypt the patient's genome under our Superfast C-FE scheme with a public key issued by the authority and then publish the ciphertext. Later, when a medical unit wants to do some disease susceptibility test, it obtains the encrypted genome and asks the authority for a one-time function key corresponding to the required disease-susceptibility test. The maximum number of disease markers for a disease susceptibility test are no more than 50 Single Nucleotide Polymorphisms (single nucleotide variation between two species) (SNPs) [129]. We conducted our experiments with 1000 SNPs disease

[10]If patient wishes, she can opt to be asked by the authority for permission to conduct test using email, SMS, phone call or some other method; alternatively patient can decide beforehand which parties are allowed to learn which functions.

test to show the efficiency of our scheme. As shown in Table 5.2, our scheme can compute disease susceptibility tests very efficiently.

Patient Similarity. Suppose Alice is suffering from a cancer and her physician wants to search (on a nation-wide scale) for another patient with similar symptoms and genetic build-up treated for the same cancer, in a hope that if some therapy and treatment worked or didn't work, it would help to treat Alice's cancer. Hospitals are typically reluctant to share data with each other without proper security protection, due to concerns about privacy and liability. Putting effective protection in place is highly nontrivial, given the scale of the problem: there are 5723 registered hospitals in United States [130] and more than 15 million patients suffering from cancer in US alone [131]. It can be difficult for hospitals to even share the data such as the total number of diabetic patients to form cohorts for medical studies. In this situation, sharing genomic data is extremely far-fetched.

Many schemes [132, 133, 134, 135, 136, 137, 138] have been proposed for measuring similarity between genomes but they are designed for comparing two genomes and none of them can actually gracefully scale to handle the complete human SNP profile similarity comparison (i.e. 4 million letters of 2-bit alphabet). Comparing one SNP profile to potentially thousands or hundred of thousands is out of question for current schemes. We show that our Superfast C-FE scheme can efficiently support complete human SNP profile comparison and is highly scalable to be used for comparison with a very large population.

Each human has 4 million SNPs. Comparing similarity of the complete 4 million SNP profile requires 226 seconds in our scheme and is highly parallelizable. Assuming that the pricing model of the authority is similar to Amazon EC2, similarity comparison of the complete genome would cost only $0.014 per single 4 million SNPs profile comparison. Most of the time complete SNP profile comparison is not required and that's why we also conducted experiments with small function vector size (10,000 and 20,000). Comparison with function vector of size 20,000 can be done in 22.37 seconds and it would cost $0.0014 per comparison. Finding a similar genome in a collection of 100,000 genomes would cost only $1414 when comparing all 4 million SNPs, while it would cost only $140 when comparing any random 20,000 SNPs.

Paternity and Kinship. For privacy-preserving paternity and kinship tests, Baldi et al. make use of both cryptographic tools (i.e., private set intersection) and biological tools (e.g., emulating the Restriction Fragment Length Polymorphism

chemical test in software) [124]. Furthermore, their subsequent work demonstrates a framework for conducing such tests on a Android smartphone [125]. Baldi et al. have a very elegant idea of exploiting domain knowledge to bring privacy-preserving genomic computation to the world of plausibility. Their scheme is based on private-set intersection protocol and is not general enough to be used for other types of genomic computation. Moreover, they assume that user is storing her own genome which is not a very practical assumption. Also, their scheme requires access to the complete genome (i.e. 3 billion letters), which is very expensive. Moreover, they only show how they can find paternity test, we can also support other relations such as sibling, uncle, cousin, etc. Our approach can support paternity and kinship inference using human SNP profile that can be obtained for less than $100 (e.g., from 23andme). Our constructions can be used to implement kinship inference algorithms described in [139].

Chapter 6

Conclusion

Secure and efficient computation on encrypted data could protect data against powerful adversaries, however, state-of-the-art schemes are either secure or efficient but not both. In this thesis, we first analyzed the security of the practical encryption schemes used in encrypted database systems that allow computation on encrypted data and found that these schemes are not secure enough for real applications. Second, we developed a new model for searching on encrypted data and designed a more secure and efficient symmetric searchable encryption scheme. Finally, we developed a new model that allows more general computation, including any polynomial-time computable function, on encrypted data, and developed very efficient schemes using this model.

Many encrypted database (EDB) systems have been proposed in the last few years as cloud computing has grown in popularity and data breaches have increased. The state-of-the-art EDB systems for relational databases can handle SQL queries over encrypted data and are competitive with commercial database systems. These systems, most of which are based on the design of CryptDB (*SOSP 2011*), achieve these properties by making use of property-preserving encryption schemes such as deterministic (DTE) and order-preserving encryption (OPE). In Chapter 3, we study the concrete security provided by such systems. We present a series of attacks that recover the plaintext from DTE- and OPE-encrypted database columns using only the encrypted column and publicly-available auxiliary information. We consider well-known attacks, including frequency analysis and sorting, as well as new attacks based on combinatorial optimization. We evaluate these attacks empirically in an electronic medical records (EMR) scenario using real patient data from 200 U.S. hospitals. When the encrypted database is operating in a steady-state where enough encryption layers have been peeled to permit the application to run its queries, our experimental results show that an alarming amount of sensitive information can be recovered. In particular, our attacks correctly recovered certain OPE-encrypted attributes (e.g., age and disease severity) for more than 80% of the

patient records from 95% of the hospitals; and certain DTE-encrypted attributes (e.g., sex, race, and mortality risk) for more than 60% of the patient records from more than 60% of the hospitals.

In Chapter 4, we developed a new model to search on encrypted data, which is much more secure than property-preserving encryption. Dynamic Searchable Symmetric Encryption allows a client to store a dynamic collection of encrypted documents with a server, and later quickly carry out keyword searches on these encrypted documents, while revealing minimal information to the server. In this paper we present a new dynamic SSE scheme that is simpler and more efficient than existing schemes while revealing less information to the server than prior schemes, achieving fully adaptive security against honest-but-curious servers. We implemented a prototype of our scheme and demonstrated its efficiency on datasets from prior work. Apart from its concrete efficiency, our scheme is also simpler: in particular, it does not require the server to support any operation other than upload and download of data. Thus the server in our scheme can be based solely on a cloud storage service, rather than a cloud computation service as well, as in prior work. In building our dynamic SSE scheme, we introduce a new primitive called *Blind Storage*, which allows a client to store a set of files on a remote server in such a way that the server does not learn how many files are stored, or the lengths of the individual files; as each file is retrieved, the server learns about its existence (and can notice the same file being downloaded subsequently), but the file's name and contents are not revealed. This is a primitive with several applications other than SSE, and is of independent interest.

Oblivious RAM (ORAM) is a tool proposed to hide access pattern leakage, and there has been a lot of progress in the efficiency of ORAM schemes; however, less attention has been paid to study the applicability of ORAM for cloud applications such as symmetric searchable encryption (SSE). Although, searchable encryption is one of the motivations for ORAM research, no in-depth study of the applicability of ORAM to searchable encryption exists. We initiate the formal study of using ORAM to reduce the access pattern leakage in searchable encryption. We propose four new leakage classes and develop a systematic methodology to study the applicability of ORAM to SSE. We develop a worst-case communication baseline for SSE. We show that completely eliminating leakage in SSE is impossible. We propose single keyword schemes for our leakage classes and show that either they perform worse than streaming the entire outsourced data (for a large fraction of queries) or they do not provide meaningful reduction in leakage. We present de-

tailed evaluation using the Enron email corpus and the complete English Wikipedia corpus. Our results suggest that we need new tools to reduce the access pattern leakage in searchable encryption.

In Chapter 5, we developed a new model for computing on encrypted data that enables construction of very efficient protocols. Motivated by privacy and usability requirements in various scenarios where existing cryptographic tools (like secure multi-party computation and functional encryption) are not adequate, we introduce a new cryptographic tool called *Controlled Functional Encryption* (C-FE). As in functional encryption, C-FE allows a user (client) to learn only certain functions of encrypted data, using keys obtained from an authority. However, we allow (and require) the client to send a fresh key request to the authority every time it wants to evaluate a function on a ciphertext. We obtain efficient solutions by carefully combining CCA2 secure public-key encryption (or Homomorphic Encryption with CCA security, depending on the nature of security desired) with Yao's garbled circuit. Our main contributions in Chapter 5 include developing and formally defining the notion of C-FE; designing theoretical and practical constructions of C-FE schemes achieving these definitions for specific and general classes of functions; and evaluating the performance of our constructions on various application scenarios.

Future Research

Application Informed Cryptographic Models for Computing on Encrypted Data.

The purpose of scientific modeling is to capture the real world as realistically as possible. On the one hand, most cryptographic primitives model extremely hard scenarios of the problems, leading to inefficient schemes. On the other, most real applications can be modeled more simply, exploiting unique opportunities offered by these applications. This gap in modeling prevents cryptographic schemes from being used in real applications. I plan to bridge this gap by developing models that capture real applications faithfully and allow for efficient constructions. Controlled Functional Encryption, discussed in Chapter 5 is one example of a practical model for computing on encrypted data. We plan to investigate an adaptation of a fully homomorphic encryption model where the client can do a small amount of work and interact with the server but outsource the majority of the computation to

the server. This model can support many applications, including secure cloud computation. We envision developing a complier that would take legacy programs and generate code for the client and server automatically, so the potential adopters do not have to port their applications. We are also developing a secure data outsourcing solution for sensitive applications, such as electronic medical records, where access patterns, data sizes, and timing information leaks sensitive patient information such as patients' diseases and hospital quality measurements, such as the number of patients died in the hospital. Based on existing techniques, such as Oblivious RAM, which only hide access patterns, an efficient solution that hides length and timing information as well seems formidable, but by developing novel techniques that exploit application domain information, we have quite promising preliminary results. We believe that application informed models for computing on encrypted data will lead to faster translation into practice.

Making Cryptography Efficient through Principled Security Relaxations.

The traditional goal of cryptography is to develop perfectly secure schemes; however, recently there has been a lot of progress on cryptographic schemes with weaker security guarantees, such as property-preserving encryption and symmetric searchable encryption, which intentionally leak information for efficiency. Due to their efficiency, such schemes are gaining tremendous interest from industry, government, and research community. The fundamental question that arises is, *what are the security implications of such leakage for real applications?* Little has been done to explore this question. We plan to conduct an in-depth study of the implications of information leaked by such schemes. We are still far from developing practical leakage-free schemes for problems such as encrypted search; for example, we show that even the most efficient Oblivious RAM scheme cannot reduce leakage in symmetric searchable encryption with performance better than streaming all outsourced data [140]. Therefore, we plan to develop reasonable leakage notions and a framework to understand the implications of such leakage for real applications. Finally, we plan to develop efficient cryptographic schemes for such reasonable leakage notions. We are currently working on understanding leakage of property-preserving and searchable encryption. We are also developing more expressive, secure, and efficient searchable encryption schemes.

References

[1] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *ACM Conference on Computer and Communications Security (CCS)*, 2015, pp. 644–655.

[2] M. Naveed, M. Prabhakaran, and C. A. Gunter, "Dynamic searchable encryption via blind storage," in *S&P*, 2014, pp. 639–654.

[3] M. Naveed, S. Agrawal, M. Prabhakaran, X. Wang, E. Ayday, J.-P. Hubaux, and C. Gunter, "Controlled functional encryption," in *ACM Conference on Computer and Communications Security (CCS)*, 2014, pp. 1280–1291.

[4] M. Bellare, A. Boldyreva, and A. O'Neill, "Deterministic and efficiently searchable encryption," in *CRYPTO*, 2007, pp. 535–552.

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *SIGMOD*, 2004, pp. 563–574.

[6] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill, "Order-preserving symmetric encryption," in *EUROCRYPT*, 2009, pp. 224–241.

[7] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *CRYPTO*, 2013.

[8] S. Faber, S. Jarecki, H. Krawczyk, Q. Nguyen, M. Rosu, and M. Steiner, "Rich queries on encrypted data: Beyond exact matches," in *Computer Security–ESORICS 2015*. Springer, 2015, pp. 123–145.

[9] M. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: ramification, attack and mitigation," in *NDSS*, 2012.

[10] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, 2009, pp. 169–169.

[11] D. Boneh, A. Sahai, and B. Waters, "Functional encryption: Definitions and challenges," TCC, Tech. Rep., 2011.

[12] A. C.-C. Yao, "How to generate and exchange secrets," in *FOCS*, 1986, pp. 162–167.

[13] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[14] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern, "RSA-OAEP is secure under the RSA assumption," in *CRYPTO*, Jan. 2001, no. 2139, pp. 260–274. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-44647-8_16

[15] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations among notions of security for public-key encryption schemes," in *CRYPTO '98*, no. 1462, pp. 26–45. [Online]. Available: http://link.springer.com/chapter/10.1007/BFb0055718

[16] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searching on encrypted data," in *S&P*, 2000, pp. 44–55.

[17] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: Protecting confidentiality with encrypted query processing," in *SOSP*, 2011, pp. 85–100.

[18] C. Peikert, V. Vaikuntanathan, and B. Waters, "A framework for efficient and composable oblivious transfer," in *CRYPTO 2008*, 2008, no. 5157, pp. 554–571. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-540-85174-5_31

[19] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *CCS*, 2012, pp. 784–796.

[20] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay-secure two-party computation system." in *USENIX Security*, 2004, pp. 287–302.

[21] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits." in *USENIX Security*, vol. 201, no. 1, 2011.

[22] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries," in *USENIX Security*, 2012, pp. 14–14.

[23] B. Kreuter, B. Mood, A. Shelat, and K. Butler, "Pcf: A portable circuit format for scalable two-party secure computation," *USENIX Security*, 2013.

[24] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *CRYPTO*, 2013, pp. 18–35.

[25] Y. Lindell, "Fast cut-and-choose based protocols for malicious and covert adversaries," in *CRYPTO*, 2013, pp. 1–17.

[26] Y. Huang, J. Katz, and D. Evans, "Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution," in *IEEE S&P*, 2012, pp. 272–284.

[27] S. Zahur and D. Evans, "Circuit structures for improving efficiency of security and privacy tools," in *IEEE S&P*, 2013, pp. 493–507.

[28] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan, "Orthogonal security with cipherbase." in *CIDR*, 2013.

[29] I. H. Akin and B. Sunar, "On the difficulty of securing web applications using CryptDB," in *PriSec*, 2014.

[30] I. A. Al-Kadit, "Origins of cryptology: The Arab contributions," *Cryptologia*, vol. 16, no. 2, pp. 97–126, 1992.

[31] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[32] T. Brekne, A. Ârnes, and A. Øslebø, "Anonymization of ip traffic monitoring data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies," in *PETs*, 2006, pp. 179–196.

[33] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme," in *ICNP*, 2002, pp. 280–289.

[34] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart, "Leakage-abuse attacks against searchable encryption," in *To appear at the ACM Conference on Communications and Computer Security (CCS '15)*. ACM, 2015.

[35] T. Sanamrad, L. Braun, D. Kossmann, and R. Venkatesan, "Randomly partitioned encryption for cloud databases," in *DBSec XXVIII*, 2014, pp. 307–323.

[36] Y. Elovici, R. Waisenberg, E. Shmueli, and E. Gudes, "A structure preserving database encryption scheme," in *Secure Data Management*, 2004, pp. 28–40.

[37] H. Kadhem, T. Amagasa, and H. Kitagawa, "Mv-opes: Multivalued-order preserving encryption scheme: A novel scheme for encrypting integer value to many different values," *IEICE TRANSACTIONS on Information and Systems*, vol. 93, no. 9, pp. 2520–2533, 2010.

[38] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving queries on encrypted data," in *ESORICS*, 2006, pp. 479–495.

[39] H. Kadhem, T. Amagasa, and H. Kitagawa, "A secure and efficient order preserving encryption scheme for relational databases." in *KMIS*, 2010, pp. 25–35.

[40] S. Lee, T. Park, D. Lee, T. Nam, and S. Kim, "Chaotic order preserving encryption for efficient and secure queries on databases," *IEICE transactions on information and systems*, vol. 92, no. 11, pp. 2207–2217, 2009.

[41] N. Chenette, A. O'Neill, G. Kollios, and R. Canetti, "Modular order-preserving encryption, revisited," 2015.

[42] W. Jansen and T. Grance, "Guidelines on security and privacy in public cloud computing," *NIST special publication*, pp. 800–144, 2011.

[43] S. Paquette, P. T. Jaeger, and S. C. Wilson, "Identifying the security risks associated with governmental use of cloud computing," *Government Information Quarterly*, vol. 27, no. 3, pp. 245 – 253, 2010.

[44] P. Brudenall, B. Treacy, and P. Castle, "Outsourcing to the cloud: data security and privacy risks," *Financier Worldwide and Hunton & Williams*, 2010.

[45] R. Ostrovsky, "Efficient computation on oblivious RAMs," in *STOC*, 1990, pp. 514–523.

[46] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *J. ACM*, vol. 43, no. 3, pp. 431–473, 1996.

[47] B. Pinkas and T. Reinman, "Oblivious RAM revisited," in *CRYPTO*, 2010, pp. 502–519.

[48] E. Stefanov, E. Shi, and D. Song, "Towards practical oblivious RAM," in *NDSS*, 2012.

[49] E. Stefanov and E. Shi, "Oblivistore: High performance oblivious cloud storage," in *IEEE S&P*, 2013, pp. 253–267.

[50] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data." in *IEEE S&P*, 2000, pp. 44–55.

[51] E.-J. Goh, "Secure indexes," IACR ePrint Cryptography Archive, Tech. Rep. 2003/216, 2003.

[52] Y.-C. Chang and M. Mitzenmacher, "Privacy preserving keyword searches on remote encrypted data." in *ACNS*, 2005, pp. 442–455.

[53] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *CCS*, 2006, pp. 79–88.

[54] P. van Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Workshop on Secure Data Management (SDM)*, 2010, pp. 87–100.

[55] M. Chase and S. Kamara, "Structured encryption and controlled disclosure," in *ASIACRYPT*, 2010, pp. 577–594.

[56] K. Kurosawa and Y. Ohtaki, "UC-secure searchable symmetric encryption," in *Financial Cryptography and Data Security (FC)*, 2012.

[57] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *CCS*, 2012, pp. 965–976.

[58] S. Kamara and C. Papamanthou, "Parallel and dynamic searchable symmetric encryption," in *Financial Cryptography and Data Security, FC (2013)*, 2013.

[59] E. Stefanov, C. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," 2014.

[60] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rocsu, and M. Steiner, "Dynamic searchable encryption in very large databases: Data structures and implementation," 2014.

[61] P. Golle, J. Staddon, and B. R. Waters, "Secure conjunctive keyword search over encrypted data." in *ACNS*, 2004, pp. 31–45.

[62] S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner, "Outsourced symmetric private information retrieval," in *CCS*. ACM, 2013, pp. 875–888.

[63] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Functional encryption with bounded collusions via multi-party computation," in *CRYPTO*, 2012, pp. 162–179.

[64] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich, "Reusable garbled circuits and succinct functional encryption," in *STOC*, 2013, pp. 555–564.

[65] K.-M. Chung, J. Katz, and H.-S. Zhou, "Functional encryption from (small) hardware tokens," in *ASIACRYPT*, 2013, pp. 120–139.

[66] A. Sahai and H. Seyalioglu, "Worry-free encryption: functional encryption with public keys," in *CCS*, 2010, pp. 463–472.

[67] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Functional encryption with bounded collusions from multiparty computation," in *CRYPTO*, 2012.

[68] "Tpm reset attack," http://www.cs.dartmouth.edu/~pkilab/sparks/.

[69] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *CCS*, 2006, pp. 79–88.

[70] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *JACM*, vol. 43, no. 3, pp. 431–473, 1996.

[71] "Google Encrytped Big Query," https://github.com/google/encrypted-bigquery-client.

[72] "Always Encrypted," https://msdn.microsoft.com/en-us/library/mt163865(v=sql.130).aspx.

[73] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[74] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2012.

[75] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, 1955.

[76] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, 1957.

[77] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman, "Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation," in *EUROCRYPT*, 2015, pp. 563–594.

[78] R. A. Popa, F. H. Li, and N. Zeldovich, "An ideal-security protocol for order-preserving encoding," in *S&P*, 2013, pp. 463–477.

[79] F. Kerschbaum and A. Schroepfer, "Optimal average-complexity ideal-security order-preserving encryption," in *CCS*, 2014, pp. 275–286.

[80] "Fifth Annual Benchmark Study on Privacy and Security of Healthcare Data," http://www.ponemon.org/blog/criminal-attacks-the-new-leading-cause-of-data-breach-in-healthcare, accessed: 2015-05-15.

[81] "HCUP Databases. Healthcare Cost and Utilization Project (HCUP). 2008-2009. Agency for Healthcare Research and Quality, Rockville, MD." www.hcup-us.ahrq.gov/databases.jsp.

[82] "OpenEMR," http://www.open-emr.org/, accessed: 2015-05-15.

[83] "Hospital Discharge Data Public Use Data File," http://www.dshs.state.tx.us/THCIC/Hospitals/Download.shtm.

[84] "Hospital Inpatient Discharges (SPARCS De-Identified): 2012," https://health.data.ny.gov/Health/Hospital-Inpatient-Discharges-SPARCS-De-Identified/u4ud-w55t.

[85] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," *Journal of Computer Security*, vol. 19, no. 5, pp. 895–934, 2011.

[86] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," Electronic Colloquium on Computational Complexity (ECCC) TR01-016, 2001, previous version "A unified framework for analyzing security of protocols" available at the ECCC archive TR01-016. Extended abstract in FOCS 2001.

[87] O. Goldreich, *Foundations of Cryptography: Basic Applications*. Cambridge University Press, 2004.

[88] "Crypto++," http://www.cryptopp.com/.

[89] "Enron dataset," https://www.cs.cmu.edu/~enron/.

[90] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *CCS*, 2012, pp. 965–976.

[91] T. Mayberry, E.-O. Blass, and A. H. Chan, "Efficient private file retrieval by combining oram and pir," in *NDSS*, 2014, pp. 1–11.

[92] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs, "Onion oram: A constant bandwidth blowup oblivious ram," Cryptology ePrint Archive, 2015. http://eprint. iacr. org/2015/005, Tech. Rep.

[93] T. Moataz, T. Mayberry, and E.-O. Blass, "Constant communication oram with small blocksize," in *CCS*, 2015.

[94] X. Wang, K. Nayak, C. Liu, E. Shi, E. Stefanov, and Y. Huang, "Oblivious data structures." in *CCS*, 2014, p. 185.

[95] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: An extremely simple oblivious ram protocol," in *CCS*, 2013, pp. 299–310.

[96] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *JACM*, vol. 43, no. 3, pp. 431–473, 1996.

[97] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rocsu, and M. Steiner, "Highly-scalable searchable symmetric encryption with support for boolean queries," in *CRYPTO 2013*, 2013, pp. 353–373.

[98] "Havasupai tribe and the lawsuit settlement aftermath," http://genetics.ncai.org/case-study/havasupai-Tribe.cfm.

[99] "Indian tribe wins fight to limit research of its dna," http://www.nytimes.com/2010/04/22/us/22dna.html?pagewanted=all&_r=1&.

[100] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *EUROCRYPT*, 2005, pp. 457–473.

[101] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *CCS*, 2006, pp. 89–98.

[102] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE S&P*, 2007, pp. 321–334.

[103] J. Katz, A. Sahai, and B. Waters, "Predicate encryption supporting disjunctions, polynomial equations, and inner products," in *EUROCRYPT*, 2008, pp. 146–162.

[104] T. Okamoto and K. Takashima, "Fully secure functional encryption with general relations from the decisional linear assumption," in *CRYPTO 2010*, 2010, no. 6223, pp. 191–208. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-14623-7_11

[105] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Attribute-based encryption for circuits," in *STOC*, 2013, pp. 545–554.

[106] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," *STOC*, pp. 40–49, 2013.

[107] A. O'Neill, "Definitional issues in functional encryption," Cryptology ePrint Archive, Report 2010/556, 2010.

[108] S. Agrawal, S. Gurbanov, V. Vaikuntanathan, and H. Wee, "Functional encryption: New perspectives and lower bounds," in *Crypto*, 2013.

[109] S. Garg, C. Gentry, S. Halevi, A. Sahai, and B. Waters, "Attribute-based encryption for circuits from multilinear maps," in *CRYPTO 2013*, 2013, no. 8043, pp. 479–499.

[110] A. Joux, "Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields," in *EUROCRYPT*, 2013, pp. 177–193.

[111] Y. Ishai, "Randomization techniques for secure computation," *Secure Multi-Party Computation*, vol. 10, pp. 222–248, 2013.

[112] M. Prabhakaran and M. Rosulek, "Rerandomizable rcca encryption," in *CRYPTO*, 2007, pp. 517–534.

[113] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *IEEE S&P*, 2013, pp. 334–348.

[114] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh, "Privacy-preserving matrix factorization," in *CCS*, 2013, pp. 801–812.

[115] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," in *USENIX Security*, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2534766.2534792 pp. 289–304.

[116] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor, "For your phone only: custom protocols for efficient secure function evaluation on mobile devices," *SCN*, 2013.

[117] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation." *IACR Cryptology ePrint Archive*, vol. 2011, p. 272, 2011.

[118] S. Kamara, P. Mohassel, and B. Riva, "Salus: a system for server-aided secure function evaluation," in *CCS*, 2012, pp. 797–808.

[119] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*, 2001, pp. 136–145.

[120] M. Prabhakaran and M. Rosulek, "Homomorphic encryption with cca security," in *Automata, Languages and Programming*, 2008, pp. 667–678.

[121] M. Bellare and P. Rogaway, "Optimal asymmetric encryption," in *EUROCRYPT*, 1995, pp. 92–111.

[122] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols," in *NDSS*, 2012.

[123] "A combinational multiplier using the xilinx spartan ii fpga," http://ecen3233.okstate.edu/PDF/Labs/Combinational%20Multiplier.pdf.

[124] P. Baldi, R. Baronio, E. De Cristofaro, P. Gasti, and G. Tsudik, "Countering gattaca: efficient and secure testing of fully-sequenced human genomes," in *CCS*, 2011, pp. 691–702.

[125] E. De Cristofaro, S. Faber, P. Gasti, and G. Tsudik, "Genodroid: are privacy-preserving genomic tests ready for prime time?" in *WPES*, 2012, pp. 97–108.

[126] E. Ayday, J. L. Raisaro, and J.-P. Hubaux, "Privacy-enhancing technologies for medical tests using genomic data," in *NDSS*, 2013.

[127] E. Ayday, J. L. Raisaro, P. J. McLaren, J. Fellay, and J.-P. Hubaux, "Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data," in *HealthTech*, 2013.

[128] E. Ayday, J. L. Raisaro, J. Rougemont, and J.-P. Hubaux, "Protecting and evaluating genomic privacy in medical tests and personalized medicine," in *WPES*, 2013.

[129] "List of genetic diseases with associated genes and snp's," http://www.eupedia.com/genetics/genetic_diseases.shtml.

[130] "Fast facts on US hospitals," http://www.aha.org/research/rc/stat-studies/fast-facts.shtml.

[131] "Cancer facts and statistics," http://www.cancer.org/research/cancerfactsstatistics/.

[132] S. Jha, L. Kruger, and V. Shmatikov, "Towards practical privacy for genomic computation," in *IEEE S&P*, 2008, pp. 216–230.

[133] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls, "Privacy-preserving matching of DNA profiles," *IACR Cryptology ePrint Archive*, vol. 2008, p. 203, 2008.

[134] D. Eppstein, M. T. Goodrich, and P. Baldi, "Privacy-enhanced methods for comparing compressed DNA sequences," *arXiv preprint arXiv:1107.3593*, 2011.

[135] D. Szajda, M. Pohl, J. Owen, B. G. Lawson, and V. Richmond, "Toward a practical data privacy scheme for a distributed implementation of the smith-waterman genome sequence comparison algorithm." in *NDSS*, 2006.

[136] M. Blanton, M. J. Atallah, K. B. Frikken, and Q. Malluhi, "Secure and efficient outsourcing of sequence comparisons," in *ESORICS*, 2012, pp. 505–522.

[137] M. J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *International Journal of Information Security*, vol. 4, no. 4, pp. 277–287, 2005.

[138] R. Wang, X. Wang, Z. Li, H. Tang, M. K. Reiter, and Z. Dong, "Privacy-preserving genomic computation through program specialization," in *CCS*, 2009, pp. 338–347.

[139] A. Manichaikul, J. C. Mychaleckyj, S. S. Rich, K. Daly, M. Sale, and W.-M. Chen, "Robust relationship inference in genome-wide association studies," *Bioinformatics*, vol. 26, no. 22, pp. 2867–2873, 2010.

[140] M. Naveed, "The fallacy of composition of oblivious ram and searchable encryption," Cryptology ePrint Archive, Report 2015/668, 2015.