BUILDING PRACTICAL SYSTEMS THAT COMPUTE ON ENCRYPTED DATA

by

RALUCA ADA POPA

Master of Engineering, Massachusetts Institute of Technology (2010)
Bachelor of Science, Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2014

Signature redacted

Author ....
Department of Electrical Engineering and Computer Science

Signature redacted

Certified by ....
Nickolai Zeldovich
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Signature redacted

Accepted by ....
Leslie A. Kolodziejski
Professor of Electrical Engineering
Chair, Department Committee on Graduate Students

BUILDING PRACTICAL SYSTEMS THAT COMPUTE ON ENCRYPTED DATA

by

RALUCA ADA POPA

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

Theft of confidential data is prevalent. In most applications, confidential data is stored at servers. Thus, existing systems naturally try to prevent adversaries from compromising these servers. However, experience has shown that adversaries still find a way to break in and steal the data.

This dissertation shows how to protect data confidentiality even when attackers get access to all the data stored on servers. We achieve this protection through a new approach to building secure systems: building practical systems that compute on encrypted data, without access to the decryption key. In this setting, we designed and built a database system (CryptDB), a web application platform (Mylar), and two mobile systems, as well as developed new cryptographic schemes for them. We showed that these systems support a wide range of applications with low overhead. The work in this thesis has already had impact: Google uses CryptDB's design for their new Encrypted BigQuery service, and a medical application of Boston's Newton-Wellesley hospital is secured with Mylar.

Thesis Supervisor: Nickolai Zeldovich
Title: Associate Professor of Computer Science and Engineering

To my family.

Contents

Previously Published Material

Chapter 3 revises a previous publication [PRZB11]:
Raluca Ada Popa, Catherine M.S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. *CryptDB: Protecting confidentiality with encrypted query processing.* In the Proceedings of the ACM Symposium on Operating Systems Principles *(SOSP)*, 2011.

Chapter 4 revises a previous publication [PSV$^+$14]:
Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, M. Frans Kaashoek, and Hari Balakrishnan. *Building Web Applications on Top of Encrypted Data using Mylar.* In the Proceedings of the USENIX Symposium of Networked Systems Design and Implementation *(NSDI)*, 2014.

The content in Chapter 5 is based on both the above publication [PSV$^+$14] and this publication [PZ13]:
Raluca Ada Popa and Nickolai Zeldovich. *Multi-Key Searchable Encryption.* In the Cryptology ePrint Archive, 2013/508.

The content in Chapter 6 revises a previous publication [GKP$^+$13a]:
Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. *Reusable garbled circuits and succinct functional encryption.* In the Proceedings of the ACM Symposium on the Theory of Computing (*STOC*), 2013.

Acknowledgments

I always struggle writing acknowledgements because words cannot do justice to my gratitude. But let me give it a try.

I am deeply thankful to my advisor, Nickolai Zeldovich. I learned a lot from him, ranging from skills such as holding a high bar for one's research all the way to good implementation design. Even though some of my ideas initially sounded impossible to achieve, he not only allowed me to pursue my interests, but also supported me and worked alongside me to make them possible. I had a lot of fun working with him! I'm embarrassed to say that I wrote a list of the things I appreciate about Nickolai and I plan to use the list when advising my students.

I am very much grateful to my collaborator and mentor Hari Balakrishnan. I thank him for his enthusiastic support, for his wise advice in important decisions in my life, and for the wonderful time I spent working with him.

Shafi Goldwasser, a collaborator and mentor, immediately welcomed me to the world of cryptography, and supported me in my interest to bridge cryptography and systems. I am deeply thankful for her support, for her helpful advice, and for the wonderful time we spent together working on interesting cryptography problems.

My other cryptography mentors, in addition to Shafi, were Yael Kalai and Vinod Vaikuntanthan. The three of them taught me how to do cryptography research! To mention only a few examples of how they helped me, I still remember the day Yael gave me line-by-line comments on my paper draft and the long days of intense cryptography work with Vinod in Toronto (which were fueled primarily by sushi).

I am equally grateful to Frans Kaashoek. Even though I was not his student, Frans gave me precious feedback and advice, and took a significant amount of time from his schedule to help me. I have also enjoyed working with him on Mylar and I sure wish I had started working with him earlier!

Ron Rivest and Barbara Liskov were my first mentors in research during my undergraduate at MIT and they continued to help and support me during my eight years at MIT, for which I am very thankful.

I am grateful to Silvio Micali for his excellent and unique way of teaching the cryptography class; his class ignited my passion for cryptography. I am also thankful for his generous advice.

Many others helped develop ideas in this work or provided support or advice. Some of them are, in alphabetic order: Dan Boneh, Srini Devadas, Linda Griffith, John Guttag, Manolis Kellis, Sam Madden, David Mazières, Muriel Médard, Robert Morris, Martin Rinard and Ronitt Rubinfeld. Thank you to Úlfar Erlingsson and the folks at sql.mit.edu for trying out CryptDB. My wonderful colleagues and office mates from the PDOS, database, PMG, and crypto group contributed in many ways to my enjoyable time in graduate school.

◇

A truly special place in my career and in my heart belongs to my best friend and my love, Alessandro Chiesa. He has been a constant source of support, of love, of joy, and a wonderful colleague ever since our first classes together in undergraduate at MIT.

I am grateful to Liviu and Cristina Iftode who generously advised me to attend college in the US and helped me settle in. Without them, my career path might have been quite different.

I am always thankful to my dear family back in Romania, my mom Maria Tatiana Chereches and my brother Alex Dan Popa, for their unconditional love, support and help, as well as for the joy they bring me. I am also thankful to my dear relatives, and to my wonderful friends who make me enjoy a lot of aspects of life outside of work.

CHAPTER 1

Introduction

This dissertation shows how to protect data confidentiality by taking a new approach to building secure systems: building practical systems that compute on encrypted data.

□ 1.1 Motivation

Leakage of confidential data plagues many computing systems today. For example, last year marks a peak in data breaches: about 740 million records [All14] were exposed, the largest number so far. Most applications store sensitive data at servers, so preventing data leakage from servers is a crucial task towards protecting data confidentiality.

One potential approach [Bla93, Fer06] to protecting confidentiality is to encrypt the data stored at the server and never send the decryption key to the server. This approach is promising because an attacker at the server can access only encrypted data and thus does not see the data content. However, this approach cannot be applied for most systems, such as databases, web applications, mobile applications, machine learning tools, and others. The reason is that these systems need to compute on the data and simply encrypting the data no longer allows such computation. Hence, existing systems have a different strategy.

In fact, almost all systems deployed today follow *the same* strategy: try to prevent attackers from breaking into servers. The reasoning behind this strategy is that, if attackers cannot break into servers, they cannot access the data at these servers. This strategy has been implemented using a multitude of approaches: checks at the operating system level, language-based enforcement of a security policy, static or dynamic analysis of application code, checks at the network level, trusted hardware, and others.

Nevertheless, data *still leaks* with these approaches. It turns out that *attackers eventually break in*. To understand why it is hard to prevent attackers from getting access to server data, we now provide examples of common attackers and the reason they manage to subvert existing systems. First, hackers notoriously break into systems by exploiting software bugs and gain access to sensitive data. They succeed because software is complex and hard to make bug free. In some cases, the hackers even manage to get administrative access to the servers [Sla11], thus gaining access to all the data stored there. The second threat appears in the context of cloud computing. More and more companies are outsourcing their data to external clouds, so their sensitive data becomes readily available to curious cloud administrators [Tat10]. Most existing protection mechanisms do not prevent such attackers because some cloud employees have full access to the servers and can simply bypass these mechanisms. Finally, according to recent news, the government has been accessing a lot of private data even without subpoena. In some cases, the government leverages physical access to the servers, making it hard to protect the data stored on these.

Hence, despite existing protection systems, attackers eventually break in or bypass the protection, thus getting access to the data stored on servers.

□ 1.2 Our contributions

The work in this thesis protects data confidentiality even against attackers who *get access to all server data*.

We take a different approach from previous systems. We assume that all server data will leak and aim to protect confidentiality even then. The strategy is to have the server store encrypted data and process and compute on it *efficiently*, without receiving the decryption key. This approach protects data confidentiality because the server receives only encrypted sensitive data: even if an attacker breaks into the server and reads all the data stored there, the data is encrypted and the server never receives the decryption key. In particular, the data remains encrypted during computation.

The cryptographic community has already proposed computing on encrypted data [RAD78, Gen09]. However, the proposed schemes, such as fully homomorphic encryption (FHE), are prohibitively slow for most systems: they are currently orders of magnitude slower than unencrypted computation. A significant challenge lies in building *practical systems*.

We have designed and implemented practical systems that provide a rich functionality over encrypted data: a database system (CryptDB [PRZB11]), a web application framework (Mylar [PSV$^+$14]), mobile systems (PrivStats [PBBL11] and VPriv [PBB09]), and a cloud storage system (CloudProof [PLM$^+$11]). As part of these projects, we contributed new system designs as well as constructed new encryption schemes: mOPE [PLZ13] and ADJ-JOIN [PZ12] for order and join operations in CryptDB, and multi-key search [PZ13] for searching in Mylar. We also constructed functional encryption schemes [GKP$^+$13a], [GKP$^+$13b] that enable computing any function over encrypted data.

Our systems have already had impact, as detailed in Chapter 7. Most notably, Google uses CryptDB's design for their new Encrypted BigQuery service, and a medical application of Boston's Newton-Wellesley hospital is secured with Mylar.

□ 1.3 How to build practical systems

Building practical systems that compute on encrypted data is a challenging task. One reason is that systems run complex software that manipulates sensitive data. Using cryptographic schemes that compute complex functions over encrypted data results in a prohibitively slow system. Another reason is that many systems take advantage of fast search data structures (such as database indexes), and a practical system must preserve this performance over encrypted data. Yet another reason is that

Meta-strategy for building practical systems that compute on encrypted data.

Step 1. Understand the system and its common use cases. In particular, one should identify basic operations (called *primitives*) that enable a wide class of applications.

▷ It turns out that, for many systems, only a small set of primitives is needed to support significant functionality. For example, to support a large class of SQL database applications in CryptDB (called OLTP [Cou] applications), one needs to support only six primitives: get/put, sum, $=$, $\geq$, join by equality, and keyword search.

Step 2. Identify an *efficient* encryption scheme that enables each primitive.

▷ Since each encryption scheme is targeted at one *specific* operation, it can be efficient! For example, in CryptDB, we used the Paillier cryptosystem [Pai99] for addition.

▷ This step must be completed with care. One should understand the various tools for computing on encrypted data, and especially the tradeoff they provide between functionality, security and performance. To aid in this endeavor, in Chapter 2, we present the various ways one can compute on encrypted data with a focus on their practical use and on the tradeoff they provide. Unfortunately, not all desired encryption schemes have already been constructed, so one has to design new ones: for example, we designed mOPE [PLZ13] and ADJ-JOIN [PZ12] for order and join operations in CryptDB, and multi-key search [PZ13] for searching in Mylar.

Step 3. Design and build a system that uses these encryption schemes as building blocks.

▷ There are a few challenges here. One challenge is that different encryption schemes cannot be combined arbitrarily; for example, even though a function may use only a combination of $+$ and $\geq$, this function may not be supported. The second challenge is to provide both a meaningful level of security for the overall system and good overall performance, despite the existing tension between security and performance. The question here is usually how to find a meaningful tradeoff between security and performance.

Figure 1-1: Strategy to building practical systems that compute on encrypted data. The paragraphs preceded by ▷ provide a discussion of the corresponding step.

redesigning these systems in drastically different ways than the unencrypted design and requiring significant changes to applications precludes adoption.

Given these challenges, how do we build *practical* systems that compute on encrypted data?

The solution comes from a *synergy of systems and cryptography* research. As discussed, if one directly applies existing cryptography to a system, one uses a generic encryption scheme such as fully homomorphic encryption to capture all the computations performed by the system. Unfortunately, this approach results in a prohibitively slow system. The insight is to leverage the fact that there are *fast specialized encryption schemes*. Namely, there are encryption schemes that target one specific operation, typically a simple operation, and are fast at performing this operation over encrypted data.

Since a system supports many operations, one might need to use a set of such specialized encryption schemes and compose them to support more complex operations. Unfortunately, this goal is challenging because different encryption schemes cannot be composed arbitrarily. Nevertheless, an understanding of the target system and how it is used by applications enables finding techniques for composing these encryption schemes.

We have developed a strategy for designing practical systems that compute on encrypted data that incorporates this insight, shown in Fig. 1-1. While the second step of this strategy may require new cryptography, the third step requires inventing new systems techniques. For example, in CryptDB, we designed a new query rewriting algorithm to support queries with nested or more complex operations and a way to adjust the encryption of the data on the fly, while in Mylar, we designed a method for verifying the key distribution performed by an untrusted server.

To sum up, enabling systems to run efficiently over encrypted data requires not only an understanding of the system, but also tailoring and creating cryptography for the system.

□ 1.4 Systems we built

Within the vision underlined above, we designed and built a set of systems that differ in the overall functionality they provide: CryptDB for securing databases, Mylar for securing web applications, and VPriv and PrivStats for securing location privacy services. Fig. 1-2 shows how the threat models of these systems relate to each other. In particular, in all systems, an attacker has access to all the data at the servers. These servers store encrypted data and compute on it. The servers are different based on the functionality of the system. The client owns the data and is able to decrypt it. Our work on functional encryption could be viewed as a generalization of the computation in these systems. We note that our systems focus on protecting data confidentiality and do not guarantee integrity, freshness or availability of the data. Such properties are an interesting future work.

We now describe briefly each system, explaining how it follows the meta-strategy above, and two new encryption schemes we designed.

◇ 1.4.1 CryptDB: Database confidentiality

We designed and built CryptDB [PRZB11, PRZB12], a practical database management system (DBMS) that protects the confidentiality of data in databases: the database (DB) is encrypted and the DB server computes SQL queries over the encrypted database without decrypting it. The data and query results are decrypted only at the client application, inside a CryptDB proxy. The threat

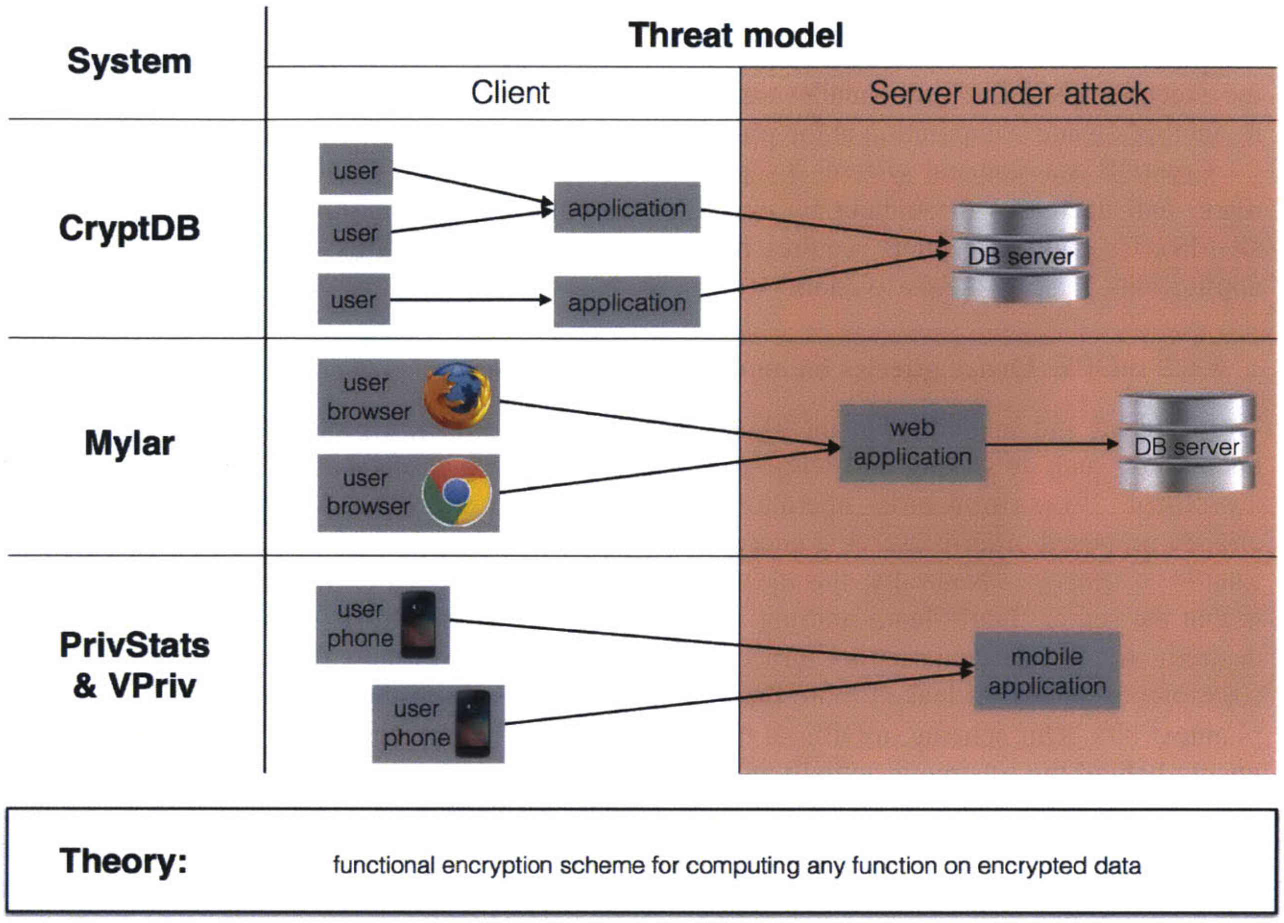

Figure 1-2: Overview of the different systems built within the vision of this thesis and how their threat models relate to each other.

model [PRZB11, CJP+11] is that the DB server is under attack, but the application and the CryptDB proxy are in a trusted location.

Running a DBMS over encrypted data is a challenging task because SQL consists of more than a hundred different operations and because applications expect high performance from DBMSs. To meet such expectations, CryptDB follows the strategy we presented above: it uses an efficient encryption scheme for each of a set of basic operations (get/put, equality, order comparison, sum, join, and keyword search) and combines the resulting six encryption schemes to support most SQL queries. We designed two of these encryption schemes, mOPE [PLZ13] for order comparison and ADJ-JOIN [PZ12] for join, because there did not exist suitable encryption schemes. These six encryption schemes provide a range of tradeoffs between security and functionality: from strong security with little ability for computation to weaker (but well-defined) security with richer functionality.

CryptDB maximizes security for a set of queries issued by the application using a new technique called *onions of encryptions* [PRZB11]; at a high level, CryptDB stacks *different* encryption schemes on top of each other while respecting certain functionality and security properties. CryptDB enables data owners to control the level of security on onions and thus protect sensitive fields with strong encryption schemes. When CryptDB cannot support a function (either because it must maintain a certain security level or because the function is too complex), we propose splitting the execution [SSPZ12] into smaller executions that can be handled by the server and doing some re-encryption and computation at the proxy.

CryptDB is a practical system: it supports all queries from TPC-C, an industry standard benchmark, and decreases throughput by only 26% as compared to vanilla MySQL (a popular SQL DBMS). Crucially, CryptDB requires no changes to existing DBMSs, and almost no change to applications, making it easy to adopt.

◇ 1.4.2 mOPE: Order queries on an encrypted database

We designed and implemented mOPE [PLZ13] (*mutable* OPE), a protocol for computing order operations, such as sorting and range queries, on an encrypted database; CryptDB needed such a protocol. To perform order operations efficiently on an encrypted database, the literature has already proposed order-preserving encryption (OPE): an encryption scheme where, if $x \leq y$, then $\mathsf{Enc}(x) \leq \mathsf{Enc}(y)$. Naturally, the ideal security of such as scheme, as defined in the literature, is that the server should learn nothing about the data other than order (which is the functionality desired). Achieving ideal security turned out to be a difficult task: more than a dozen OPE schemes were proposed, but all leaked more than order; even the state-of-the-art scheme leaks half of the plaintext bits. Our scheme, mOPE, is the first to achieve ideal security and is also practical. The insight behind the scheme is actually not from cryptography, but from systems: we observed that, in a database setting, it is acceptable to have a ciphertext depend on a few other ciphertexts in the database and to correct a small number of ciphertexts occasionally, whereas standard encryption schemes do not consider this model. Updating ciphertexts is what we call *mutation*, and we proved it is required to overcome inherent difficulties with standard OPE schemes.

◇ 1.4.3 Mylar: Securing web applications

CryptDB assumes the application server is trusted, but in some settings, one wants to protect against attacks to the application server as well, such as when hosting a web application server on a cloud. Therefore, we designed and built Mylar, a platform for writing web applications that protects data confidentiality against attackers who compromise all servers: both the application and database servers. Mylar stores encrypted data on the server, and encrypts/decrypts it only in users' browsers using each user's key.

In comparison to CryptDB's setting, two conceptual challenges arise. First, the data that the server must process is no longer encrypted with only one entity's key. Second, we aim to protect against active adversaries; for example, attackers can modify data or insert Javascript in a webpage to extract plaintext data from user browsers.

Importantly, Mylar enables common web application functionality. Mylar embraces the recent shift towards client-side applications that permits processing one specific user's data in his own browser. However, some computation cannot be run in a user's browser. To enable such computation, we apply our meta strategy: we identified two core operations which enable a broad class of web applications. These are keyword search and data sharing. The first operation is implemented using our new multi-key searchable encryption scheme [PZ13].

For data sharing, Mylar enables users to share data securely and dynamically even in the presence of active adversaries at the server, through a new mechanism for distributing and certifying keys.

It also ensures that an adversary did not modify the client-side application code via a new code certification scheme.

Mylar is practical: porting 6 web applications to Mylar required changing just 35 lines of code on average, and the performance overheads were modest and largely did not affect user experience.

◇ 1.4.4 Multi-key search: search over data encrypted with different keys

We designed the multi-key searchable encryption [PZ13] for Mylar, but it can also be used standalone. The motivation for the scheme is that the search computation in Mylar's setting must be run on the server because it can access a large amount of data, which would be costly to download in a client's browser. Consider that a user has access to n documents, all of which are shared with different users and hence encrypted with different keys. Hence, one needs a searchable encryption scheme that can search over data encrypted with different keys. No such practical encryption scheme existed for our setting, so we designed the multi-key search encryption scheme. The scheme allows a server to search an encrypted word over all the documents a user has access to, without the server learning the word searched for or the contents of the documents. An alternative to multi-key search is to invoke a regular searchable encryption scheme (single-key) n times, once for each key, but we demonstrated this approach to be less efficient.

◇ 1.4.5 PrivStats and VPriv: Securing mobile systems

We built two systems, PrivStats [PBBL11] and VPriv [PBB09], that protect location privacy of users in a mobile system from an untrusted server. The mobile setting consists of sensors or smartphones uploading car movement information or information relevant to a social network to the server. The server can compute useful aggregate statistics in PrivStats (e.g. traffic congestion, popularity of a social location, events) or functions of each client's path in VPriv (e.g., tolling cost in a month) by only handling encrypted data.

□ 1.5 Impact

CryptDB has already had impact. We provide a summary here, and delegate a complete discussion to Chapter 7. Google recently deployed a system (Encrypted BigQuery) for performing SQL-like queries over an encrypted database following (and giving credit to) CryptDB's design. Their service uses most of the encryption building blocks from CryptDB, as well as rewrites queries and annotates the schema similarly to CryptDB. The software giant SAP AG implemented a system called SEEED, which is CryptDB on top of their HANA database server. Researchers at MIT's Lincoln Labs use CryptDB's design for their D4M Accumulo NoSQL engine employing four of CryptDB's building blocks. Moreover, volunteering users of sql.mit.edu, a SQL server at MIT, are running their Wordpress instances on top of our CryptDB source code. Finally, on the academic front, CryptDB spurred a rich line of work.

Also, Mylar was used to secure a medical application of Newton-Wellesley Hospital from Boston. This is a web application collecting private information from patients suffering from the disease endometriosis. The Mylar-enhanced application was tested on real patients of endometriosis and is now in alpha deployment.

□ 1.6 Theoretical work on functional encryption

One natural question with our meta-strategy above is how long and for how many other systems will we manage to use it successfully? We managed to apply it to four systems so far, but new systems may perform quite different operations. Our theoretical work on functional encryption in this thesis seeks to answer an important subquestion: can we design encryption schemes for computing *any* new operation over encrypted data?

In [GKP$^+$13a] and [GKP$^+$13b], we show that the answer is yes: one can design a functional encryption scheme for any function. Briefly, in functional encryption, anyone can encrypt a value x with a master public key mpk and obtain $\mathsf{Enc}(x)$. The holder of the master secret key can provide keys for functions, for example sk_f for function f. Anyone with access to a key sk_f and $\mathsf{Enc}(x)$ can compute f on $\mathsf{Enc}(x)$ and obtain the result of the computation in plaintext form: $f(x)$; they learn *nothing else about x than $f(x)$*. Chapter 2 explains why we chose to study functional encryption as opposed to homomorphic computation: it turns out that functional encryption provides a more practical model of computation because it avoids certain unacceptable worst-case costs.

We devised two functional encryption schemes for general functions, where the model of computation differs. The first scheme is for functions represented as circuits [GKP$^+$13a]. The second scheme is for functions represented as Turing machines [GKP$^+$13b], which comes closer to real programs because Turing machines are more realistic than circuits. For example, one can use loops without having to unroll each loop into the maximum number of steps it can run for. In both these schemes, we required that the number of functions to be computed be fixed a priori, a restriction removed by subsequent work.

We also showed that our first FE scheme is a powerful primitive: using it, we solved a 30-year old question in cryptography, how to **reuse garbled circuits**. Garbled circuits have been used in numerous places in cryptography. In a garbling scheme, the holder of a secret key can garble a function f into a circuit G_f, called the garbled circuit, and can also encode an input x into $\mathsf{Enc}(x)$. G_f hides the function f and $\mathsf{Enc}(x)$ hides x, but using G_f and $\mathsf{Enc}(x)$, an untrusted server can compute $f(x)$, while learning nothing else about f or x. Since Yao's first garbling scheme in 1986, all garbling schemes designed required that the garbled circuit be ran only once (on one input) or else security gets compromised. This was a waste: the effort to garble f into G_f was useful for only one input. Using our scheme, a garbled circuit can be used an arbitrary number of times.

Our FE results are proofs of concept and are not efficient. Typically, one needs to design the encryption scheme from scratch to achieve practicality. However, in cryptography, knowing that a desired scheme is possible is already a crucial step towards designing it – the reason is that there is a sensitive border between possibility and impossibility. There are numerous examples in cryptography when researchers did not work on an important problem for many years out of belief that it is not solvable; once there was a sense of possibility, lots of results came about. For example, the FHE result in 2009 [Gen09] came after a decade of almost no work on the topic and spurred hundreds of new results. The recent results in IO obfuscation share a similar story.

□ 1.7 Thesis roadmap

The rest of the thesis is organized as follows. In Chapter 2, we present the ways in which one can compute on encrypted data in a practical system. This knowledge not only forms the basis of our later discussions, but it also hopefully helps someone else design practical systems that compute on encrypted data using our meta-strategy.

We then describe CryptDB in Chapter 3 and Mylar in Chapter 4. To illustrate a new encryption scheme we designed for these systems, we then present our searchable encryption scheme that Mylar builds on in Chapter 5. Then, we present our work on functional encryption in Chapter 6.

In Chapter 8, we present related work: we discuss both work related to our global approach in this thesis and work related to a specific contribution.

Finally, in Chapter 9, we conclude and discuss future directions.

CHAPTER 2

Approaches for computing on encrypted data in a practical system

Many non-cryptography researchers think that the only tool for computing on encrypted data is fully homomorphic encryption (FHE) [Gen09]. Many of them have also heard that FHE is overwhelmingly impractical, thus ruining their hopes for practical computation on encrypted data. However, in fact, there is a whole set of other tools for computing on encrypted data, some of them offering promising performance in some settings.

In this chapter, we present the ways in which one can compute on encrypted data in a practical system. These tools or methods provide an interesting spectrum over three coordinates: functionality, security, and performance. Unfortunately, none of these tools alone suffices to enable running the systems we have today over encrypted data. They typically fail to achieve a desired goal with respect to at least one of these three coordinates.

Nevertheless, this thesis shows that understanding the tradeoffs these tools provide in functionality, security, and performance, coupled with an understanding of the targeted system, enables building systems that achieve all three coordinates. Hence, below we describe each tool or method from a functionality, security and performance standpoint, describing the tradeoff it strikes in this space, and the settings in which it might be practical. Since our focus is this tradeoff, we keep the cryptographic presentation of each tool informal, and instead reference outside formal presentations.

We divide these tools and methods in two categories: tools that leak (virtually) nothing about the data and tools that reveal a well-defined function of the data. Many times, the information revealed in the second category helps improve performance drastically.

□ 2.1 Tools with no leakage

◇ 2.1.1 Fully homomorphic encryption

Functionality. FHE [Gen09] is a public-key encryption scheme [Gol04]. Using a key generation algorithm, anyone can create a pair of keys: a secret key sk and a public key pk. Anyone can encrypt a value x using the public key pk and obtain a ciphertext $\widehat{x} \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$, where the $\widehat{\ }$ notation indicates encryption with FHE.

An evaluation algorithm Eval can evaluate functions over encrypted data. A function f to be evaluated is typically represented as a boolean circuit of polynomial size in the input size; often, the gates in such a circuit compute either addition or multiplication in a finite field. Using the public key, anyone can evaluate f by computing $\mathsf{Eval}(\mathsf{pk}, \widehat{x_1}, \ldots, \widehat{x_n}, f)$ and obtain $\widehat{f(x_1, \ldots, x_n)}$, the encrypted computation result. This computation result, as well as any other FHE ciphertext, can be decrypted with the secret key sk.

The current FHE schemes enable running virtually any function over encrypted data.

Security. FHE provides strong security guarantees, called *semantic security* [Gol04]. Intuitively, semantic security requires that any adversary holding only the public key and a ciphertext cannot learn any information about the underlying plaintext, other than its length.

Performance and use cases. Following Gentry's scheme [Gen09], there have been a lot of FHE schemes proposed [Gen09, DGHV10, SS10b, BV11b, BV11a, Vai11, BGV12, GHS12a, GHS12b, LTV12, Bra12, GSW13], many of which improved the performance of the original scheme drastically. There is also an open-source implementation of FHE available, HElib [Hal13, HS14].

Nevertheless, as of now, FHE remains too slow for running arbitrary functions or for enabling the complex systems we have today. For example, an evaluation (performed in 2012) of the AES circuit reported a performance of 40 minutes per AES block on a machine with very large memory, being more than six orders of magnitude slower than unencrypted AES evaluation.

There are at least three factors that make FHE slow: the cryptographic overhead, the model of computation, and the strong security definition.

The cryptographic overhead consists of the time to perform operations for each gate of the circuit as well as other maintenance operations. Unfortunately, it is hard to characterize simply the cryptographic overhead of FHE because there are a lot of parameters that affect its performance, such as the multiplicative depth of the circuit to evaluate (the largest number of multiplication gates in the circuit on any path from input to output), the security parameter (which indicates the security level desired), the plaintext size, the exact FHE scheme used, the performance of various operations in the finite fields used, etc. Lepoint and Naehrig [LN14] and Halevi [Hal13, GHS12b] provide performance measurements for various settings of these parameters. From our experience with HElib, whenever the multiplicative depth became more than about 3 or whenever we were encrypting non-bit values, performance was becoming too slow (on the order of seconds as opposed to milliseconds) on a commodity machine for the applications we had in mind.

Regarding the model of computation, in order to evaluate a program with FHE, the program must be compiled into a boolean circuit. This could result in a large circuit with a nontrivial multiplication depth.

Hence, if one uses FHE, they should choose a computation that can be easily expressed in additions and multiplications over a finite field, whose multiplicative depth is small, and whose values encrypted are bits or otherwise live in a small plaintext domain.

The final factor, the security guarantees, brings about inherent impracticality. There are settings in which FHE is prohibitively too slow *even if its cryptographic overhead were zero*. The reason is that the security guarantee is so strong that it prevents certain needed optimizations.

Consider the example of a database table containing data about employees in a company. One column represents their salaries and the other column contains further personal information. Now consider a query that requests all the rows where the salary column equals 100. A regular database would execute this query using an index: a fast search tree that enables the database server to quickly locate the items equal to 100 (in a logarithmic number of steps in the number of entries in the database) and return each row in the database having this salary. However, using FHE, the database server has to scan the whole database! The reason is that FHE prohibits the server from learning anything about the data, even learning whether some data is worth returning or not. If the server does not include a salary item in its computation, the server learns that the item did not match the query, which is prohibited by FHE's security. Such linear scans are prohibitively slow in many real databases today. Moreover, with FHE, the database server cannot learn the number of rows that should be in the result. Hence, the server has to return the worst-case number of rows possible. If some rare queries need to return the whole database, the server has to return the whole database *every time*, which can be prohibitively slow.

In Sec. 2.2.1, we will see how functional encryption addresses exactly this problem: it weakens the security guarantee slightly, but just enough for the server to know what rows to return and how to find them efficiently (without the server learning the contents of the rows) and avoid these linear costs.

◇ 2.1.2 Partially homomorphic encryption (PHE)

Functionality. PHE has a similar definition with FHE with the difference that the function f is a specific function. Here are some examples.

- the Goldwasser-Micali [GM82] cryptosystem computes XOR of encrypted bits,
- Paillier [Pai99] computes addition,
- El Gamal [ElG84] computes multiplication,
- the BGN cryptosystem [BGN05] performs any number of additions, one multiplication, followed by any number of additions.

Security. These offer the same strong security as FHE, namely semantic security.

Performance. These schemes are more efficient and closer to practice than FHE, due to their specialized nature. For example, on a commodity machine, Paillier takes 0.005 ms to evaluate addition on two encrypted values, and 9.7 ms to encrypt a value.

□ 2.2 Tools with controlled leakage

The tools in this section do not provide semantic security: they enable the server to learn a function of the data, but nothing else. One can choose the function of the data that the server learns: ideally, this information is small and does not affect privacy, yet it can be exploited to increase performance drastically. We've seen many cases in which this approach increased performance by orders of magnitude as compared to a homomorphic solution, and brought a significantly slow solution into the practical realm.

◇ 2.2.1 Functional encryption

Functionality. In *functional encryption* (FE) [SW05, GPSW06, KSW08, LOS+10, OT10, O'N10, BSW11], anyone can encrypt an input x with a master public key mpk and obtain $\mathsf{Enc}(\mathsf{mpk}, x)$. The holder of the master secret key can provide keys for functions, for example sk_f for function f. (Note how each key is tied to a function!) Anyone with access to a key sk_f and the ciphertext $\mathsf{Enc}(\mathsf{mpk}, x)$ can obtain the result of the computation in plaintext form, $f(x)$, by running $\mathsf{Dec}(\mathsf{sk}_f, \mathsf{Enc}(\mathsf{mpk}, x))$.

As with homomorphic encryption, there are both specialized FE schemes (that support one fixed function) and generic schemes (that support any function).

Regarding specialized FE schemes, the works of Boneh and Waters [BW07], Katz, Sahai and Waters [KSW08], Agrawal, Freeman and Vaikuntanathan [AFV11], and Shen, Shi and Waters [SSW09] show functional encryption schemes for the inner product function.

Regarding general FE schemes, our work in this thesis (Chapter 6) constructs an FE scheme for any general function (where the size of the ciphertext is short as described in Sec. 6.1). Followup work [GGH+13] also constructs general FE in a stronger security model.

Security. Intuitively, the security of FE requires that the adversary learns nothing about x, other than the computation result $f(x)$. Some schemes hide the function f as well (but make some changes to the interface of the scheme or to the precise security definition of the scheme).

Note the difference from homomorphic encryption, in which the server obtains an encrypted computation result $\mathsf{Enc}(f(x))$ and does not know $f(x)$. Hence, this is a weaker security than semantic security; nevertheless, by choosing f in a careful way, one can ensure that little information leaks about x.

Performance and use cases. As with FHE, the practicality of a scheme depends on its cryptographic overhead, model and security. In terms of cryptographic overhead, the general FE schemes are currently prohibitively slow. In fact, so far, we are not even aware of the existence of an implementation of such schemes. On the other hand, the specialized functional encryption schemes could be practical in some settings.

In terms of model and security, FE is closer to practice than FHE. If the cryptographic overhead were engineered to be very small, unlike FHE, the model and security of FE no longer preclude the practical applications discussed in Sec. 2.1.1 as follows. Recall the database example from Sec. 2.1.1. With functional encryption, the client (owner of the database and master secret key), gives the server keys for computing certain functions. For example, the client can provide the server with a key for the function $f_{100}(x) = (x \stackrel{?}{=} 100)$ that compares each salary x to 100. Hence, for each row of the database, the server learns one bit of information: whether the row matches the predicate or not. Nevertheless, the server does not learn the content of those rows or the value 100. This small amount of information drastically improves performance: the server no longer has to return the whole database. (To enable the server to locate the value 100 efficiently, the client can also give keys to the server for functions that help the server navigate the database index.)

◇ 2.2.2 Garbled circuits

Functionality. With a garbling scheme, someone can compute an "obfuscation" of a function represented as a circuit C, called the garbled circuit, and an encoding of an input x, such that anyone can compute $C(x)$ using only the garbled circuit and the encoded input, without learning anything else about x or C. More concretely, a garbling scheme [Yao82, LP09, BHR12] consists of three algorithms ($\mathsf{Gb.Garble}, \mathsf{Gb.Enc}, \mathsf{Gb.Eval}$) as follows. $\mathsf{Gb.Garble}$ takes as input a circuit C, and outputs the garbled circuit Γ and a secret key sk. In many constructions, the circuit C consists of binary gates such as XOR and AND. $\mathsf{Gb.Enc}(\mathsf{sk}, x)$ takes an input x and outputs an encoding c. Finally, $\mathsf{Gb.Eval}(\Gamma, c)$ takes as input a garbled circuit Γ and an encoding c, and outputs a value y that must be equal to $C(x)$.

Many garbling schemes have the property that the secret key is of the form $\mathsf{sk} = \{L_i^0, L_i^1\}_{i=1}^n$, a set of pairs of labels, one for each bit of the input x, where n is the number of bits of the input. The encoding of an input x is of the form $c = (L_1^{x_1}, \ldots, L_n^{x_n})$ where x_i is the i-th bit of x. Namely, it is a selection of a label from each pair in sk based on the value of each bit in x.

Security. Some garbling schemes can provide both input privacy (the input to the garbled circuit does not leak to the adversary) and circuit privacy (the circuit does not leak to the adversary).

Most garbling schemes are secure *only for a one-time evaluation of the garbled circuit*: namely, the adversary can receive at most one encoding of an input to use with a garbled circuit; obtaining more than one encoding breaks the security guarantee. This means that, to execute a circuit C on a list of encrypted inputs, one has to garble the circuit C every time, for every input. The work to garble C is, in most cases, at least as large as evaluating C.

Our work in Chapter 6 provides garbling schemes that are *reusable*. However, our construction is prohibitively impractical, and thus it represents only a proof of concept.

Performance and use cases. There are a lot of implementations of garbled circuits. The state-of-the-art implementation is JustGarble [BHKR13].

There are at least two factors that affect the efficiency of a garbling scheme: the cost of converting from the desired program to a garbled circuit representation, and the cryptographic cost of garbling and evaluating the garbled circuit.

Recent work [BHKR13] managed to bring the cryptographic overhead of the garbled circuits notably low. For example, JustGarble evaluates moderate-sized garbled-circuits at an amortized cost of 23.2 cycles per gate, which corresponds to 7.25 nsec on a commodity hardware.

The conversion from a desired program to a circuit of AND and XOR gates is perhaps the most expensive because it tends to create large circuits. There are generic tools that convert a C program to a circuit [BDNP08], as well more efficient methods due to recent work [ZE13]. The most practical solution, though, is to design the circuit for a certain program from scratch, by taking into account some special structure of the program to compute, if that is possible. For example, Kolesnikov et al. [KSS09] shows how to create a simple circuit for comparing two large integers.

Garbling schemes are often used along with oblivious transfer schemes; in such a scheme, party A having an input x, can obtain the labels corresponding to x from party B without B knowing what x is. An efficient state-of-the-art oblivious transfer scheme is due to Asharov et al. [ALSZ13].

◇ 2.2.3 Secure multi-party computation (MPC)

Functionality. With MPC [Yao82, GMW87], n parties each having a private input (x_i being the private input of party i) can compute jointly a function f such that each party learns the result of the computation $f(x_1, \ldots, x_n)$, but no one else learns more about the inputs to the other parties.

There exist secure multi-party computation protocols for generic functions [Yao82, GMW87, LP07, IPS08, LP09], as well as implementations for generic functions [HKoS$^+$10], [MNPS04], [BDNP08], [BHKR13]. MPC protocols are often constructed using garbled circuits at their basis.

Besides these generic constructions, there are tens of specialized MPC algorithms, supporting all kinds of functions.

Security. Intuitively, each party learns $f(x_1, \ldots, x_n)$ and nothing else about any other party's input. There are a number of variations on the security definition. For example, the security can be defined with respect to honest-but-curious adversaries (adversaries that follow the protocol but try to learn private data), or malicious adversaries (adversaries that can cheat in a variety of ways).

Performance and uses cases. The state-of-the-art tools for generic computation [HKoS$^+$10], [MNPS04], [BDNP08], [BHKR13] are too inefficient to run general-purpose programs. Hence, they can be practical when they run simple programs.

To run slightly more complex computation, the hope is again specialization. The trick here is to observe some structural properties specific to the computation of interest; then, hopefully, one can exploit these properties to design an efficient MPC protocol. For example, the VPriv [PBB09] protocol we designed uses MPC to compute certain aggregation functions on a driver's path, such as a toll cost. The common operations to these aggregation functions were summation and set intersection. We designed a specialized protocol which was three orders of magnitude faster than the then state-of-the-art generic tool. Besides VPriv, the literature contains tens of other specialized and efficient MPC protocols [KSS13] for all kinds of computations. For guidelines on designing efficient MPC protocols, Kolesnikov et al. [KSS13] provide a systematic guide.

One caveat to remember is that these protocols are interactive: they require communication between the various parties involved and many times require that all parties be online at the same time.

◇ 2.2.4 Specialized tools

There are a lot of schemes that can compute all kinds of specialized functions or fit in various setups, while revealing only a well-defined amount of information to the server. Many of these are very efficient because of their specialized nature. When designing a practical system, these provide a gold mine of resources because of the efficiency they provide. For example, in CryptDB, 4 out of the 5 encryptions that can compute fall in this category.

Here are two notable examples:

- *Searchable encryption.* There are all kinds of searchable encryption schemes such as [SWP00, Goh, BCOP04, CM05, BBO07, CGKO06, BW07, BDDY08, ZNS11, YLW11, Raj12, KPR12, CJJ$^+$13, CGKO06, YLW11, ZNS11, Raj12] and the one presented in Chapter 5; Bosch et al. [BHJP14] provide a recent survey. The common functionality of these schemes is to enable a server to find whether some (encrypted) keyword appears in an encrypted document. Some of the variations come from whether the server can search for conjunctions or disjunctions of keywords, whether the server can use a search data structure to improve search time, whether the server can search by some limited regular expressions, variations in security (whether the server learns the number of the repetitions of a word in a document), and others.

- *Order-preserving encryption (OPE)* [AKSX04, PLZ13] maintains the order of the plaintexts in the cirphertexts. Namely, if $x \leq y$, then $\mathsf{Enc}(x) \leq \mathsf{Enc}(y)$. OPE has the advantage of allowing existing servers (such as database servers) to compute range, order, or sort queries on encrypted data in *the same way* as on unencrypted data. Further benefits are that existing software does not need to change to compute on the encrypted data and fast search data structures still work.

- *Deterministic encryption* [BFO08] enables equality checks because the same value will yield the same ciphertext, considering a fixed encryption key.

Despite all the tools we presented above, many times in practice, we need an encryption scheme that does not exist. (as was the case with the order-preserving scheme for CryptDB and the multi-key search for Mylar). Then, one needs to design a new tool and that tool most likely falls in this category.

□ 2.3 Final lessons

Here are a few simple lessons to take from the above discussion:

- There are a number of ways to compute on encrypted data and they provide different tradeoffs in the space of functionality, security, and efficiency.

- The key to good performance is specialization: use or design a specialized encryption scheme, or find a specialized model of the system setting of interest to plug into an existing generic scheme.

- When homomorphic schemes cannot provide a practical solution, take advantage of controlled-leakage tools. Choose the information revealed to the server in a way that enables crucial system optimizations, while keeping a meaningful privacy guarantee.

CHAPTER 3

Securing databases with CryptDB

□ 3.1 Introduction

This chapter presents CryptDB, a database system that protects the confidentiality of the data in the database by running SQL queries over the encrypted database. CryptDB is useful for database-backed applications whose layout consists of an application server separate from the database server. CryptDB addresses the threat of a curious database administrator (DBA) or database hacker who tries to learn private data (e.g., health records, financial statements, personal information) by snooping on the DBMS server. CryptDB prevents the DBA from learning private data because the server sees the database only in encrypted form and never receives the decryption key. Figure 3-1 shows CryptDB's architecture.

The challenge with executing SQL queries on an encrypted database lies in the tension between minimizing the amount of confidential information revealed to the DBMS server and the ability to efficiently execute a variety of queries. As we discussed in the introduction to this thesis, Chapter 1, current approaches for computing over encrypted data are either too slow or do not provide adequate confidentiality. On the other hand, encrypting data with a strong and efficient cryptosystem, such as AES, would prevent the DBMS server from executing many SQL queries, such as queries that ask for the number of employees in the "sales" department or for the names of employees whose salary is greater than $60,000. In this case, the only practical solution would be to give the DBMS server access to the decryption key, but that would allow an adversary to also gain access to all data.

The key insight that enables executing queries over encrypted data is that SQL uses a well-defined set of operators, each of which we are able to support efficiently over encrypted data. CryptDB uses two key ideas:

- The first is to execute SQL queries over encrypted data. CryptDB implements this idea using a *SQL-aware encryption strategy*, which leverages the fact that all SQL queries are made up of a well-defined set of primitive operators, such as equality checks, order comparisons, aggregates (sums), and joins. By adapting known encryption schemes (for equality, additions, and order checks) and using a new privacy-preserving cryptographic method for joins, CryptDB encrypts each data item in a way that allows the DBMS to execute on the transformed data.

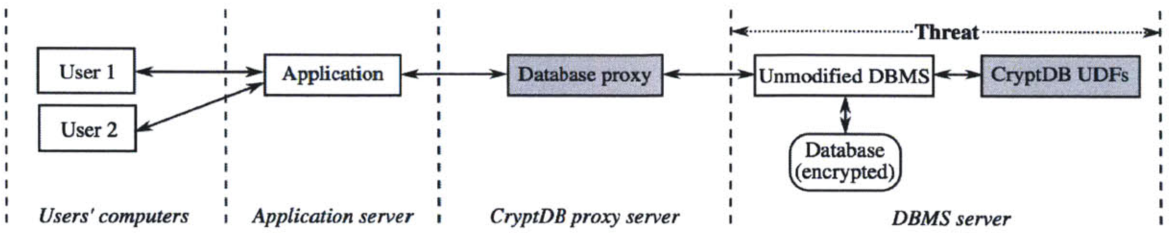

Figure 3-1: CryptDB's architecture consisting of two parts: a *database proxy* and an unmodified *DBMS*. CryptDB uses user-defined functions (UDFs) to perform cryptographic operations in the DBMS. Rectangular and rounded boxes represent processes and data, respectively. Shading indicates components added by CryptDB. Dashed lines indicate separation between users' computers, the application server, a server running CryptDB's database proxy (which is usually the same as the application server), and the DBMS server. CryptDB addresses the threat of a curious database administrator with complete access to the DBMS server snooping on private data.

CryptDB is efficient because it mostly uses symmetric-key encryption, avoids fully homomorphic encryption, and runs on unmodified DBMS software (by using user-defined functions).

- The second technique is *adjustable query-based encryption*. Some encryption schemes leak more information than others about the data to the DBMS server, but are required to process certain queries. To avoid revealing all possible encryptions of data to the DBMS *a priori*, CryptDB carefully *adjusts* the SQL-aware encryption scheme for any given data item, depending on the queries observed at run-time. To implement these adjustments efficiently, CryptDB uses *onions of encryption*. Onions are a novel way to compactly store multiple ciphertexts within each other in the database and avoid expensive re-encryptions.

We have implemented CryptDB on both MySQL and Postgres; our design and most of our implementation should be applicable to most standard SQL DBMSes. An analysis of a 10-day trace of 126 million SQL queries from many applications at MIT suggests that CryptDB can support operations over encrypted data for 99.5% of the 128,840 columns seen in the trace. Our evaluation shows that CryptDB has low overhead, reducing throughput by 26% for queries from TPC-C, compared to unmodified MySQL. We evaluated the security of CryptDB on six real applications (including phpBB, the HotCRP conference management software [Koh08], and the OpenEMR medical records application); the results show that CryptDB protects most sensitive fields with highly secure encryption schemes.

The rest of this chapter is structured as follows. In §4.2, we discuss the threats that CryptDB defends against in more detail. Then, we describe CryptDB's design for encrypted query processing in §3.3.In §3.4, we discuss limitations of our design, and ways in which it can be extended. Next, we describe our prototype implementation in §4.6, and evaluate the performance and security of CryptDB in §3.6 and conclude in §3.8. We compare CryptDB to related work in Chapter 8.

□ 3.2 Security Overview

Figure 4-1 shows CryptDB's architecture and threat model. CryptDB works by intercepting all SQL queries in a *database proxy*, which rewrites queries to execute on encrypted data (CryptDB assumes that all queries go through the proxy). The proxy encrypts and decrypts all data, and changes some query operators, while preserving the semantics of the query. The DBMS server never receives decryption keys to the plaintext so it never sees sensitive data, ensuring that a curious DBA cannot gain access to private information.

Although CryptDB protects data confidentiality, it does not ensure the integrity, freshness, or completeness of results returned to the application. An adversary that compromises the application, proxy, or DBMS server, or a malicious DBA, can delete any or all of the data stored in the database. Similarly, attacks on user machines, such as cross-site scripting, are outside of the scope of CryptDB.

CryptDB guards against a curious DBA or other external attacker with full access to the data stored in the DBMS server. Our goal is confidentiality (data secrecy), not integrity or availability. The attacker is assumed to be *passive*: she wants to learn confidential data, but does not change queries issued by the application, query results, or the data in the DBMS. This threat includes DBMS software compromises, root access to DBMS machines, and even access to the RAM of physical machines. With the rise in database consolidation inside enterprise data centers, outsourcing of databases to public cloud computing infrastructures, and the use of third-party DBAs, this threat is increasingly important.

Approach. CryptDB aims to protect data confidentiality against this threat by executing SQL queries over encrypted data on the DBMS server. The proxy uses secret keys to encrypt all data inserted or included in queries issued to the DBMS. Our approach is to allow the DBMS server to perform query processing on encrypted data as it would on an unencrypted database, by enabling it to compute certain functions over the data items based on encrypted data. For example, if the DBMS needs to perform a `GROUP BY` on column c, the DBMS server should be able to determine which items in that column are equal to each other, but not the actual content of each item. Therefore, the proxy needs to enable the DBMS server to determine relationships among data necessary to process a query. By using SQL-aware encryption that adjusts dynamically to the queries presented, CryptDB is careful about what relations it reveals between tuples to the server. For instance, if the DBMS needs to perform only a `GROUP BY` on a column c, the DBMS server should not know the order of the items in column c, nor should it know any other information about other columns. If the DBMS is required to perform an `ORDER BY`, or to find the `MAX` or `MIN`, CryptDB reveals the order of items in that column, but not otherwise.

Guarantees. CryptDB provides confidentiality for data content and for names of columns and tables; CryptDB does not hide the overall table structure, the number of rows, the types of columns, or the approximate size of data in bytes.

In Sec. 3.7, we describe formally the guarantees CryptDB provides. Here, we explain them at an intuitive level.

There are two types of security guarantees in CryptDB. CryptDB enables a data owner to annotate certain data columns with the "sensitive" annotation. For these columns, CryptDB provides strong security guarantees (semantic security [Gol04] or a similar guarantee) that essentially leak nothing about the data in those columns other than its length. One can imagine this security guarantee as having random values stored in a column with no correlation to the values encrypted other than having the same length.

The other type of security is "best-effort": for each column, CryptDB chooses the most secure encryption scheme that can enable the desired set of queries. In this case, the security of CryptDB is *not perfect:* CryptDB reveals to the DBMS server relationships among data items that correspond to the *classes of computation* that queries perform on the database, such as comparing items for equality, sorting, or performing word search. The granularity at which CryptDB allows the DBMS to perform a class of computations is an entire column (or a group of joined columns, for joins), which means that even if a query requires equality checks for a few rows, executing that query on the server would require revealing that class of computation for an entire column. §3.3.1 describes how these classes of computation map to CryptDB's encryption schemes, and the information they reveal.

More intuitively, CryptDB provides the following properties:

- Sensitive data is never available in plaintext at the DBMS server.
- The information revealed to the DBMS server depends on the classes of computation required by the application's queries, subject to constraints specified by the application developer in the schema (§3.3.5):

  1. If the application requests no relational predicate filtering on a column, nothing about the data content leaks (other than its size in bytes).
  2. If the application requests equality checks on a column, CryptDB's proxy reveals which items repeat in that column (the histogram), but not the actual values.
  3. If the application requests order checks on a column, the proxy reveals the order of the elements in the column.

- The DBMS server cannot compute the (encrypted) results for queries that involve computation classes not requested by the application.

How close is CryptDB to "optimal" security? Fundamentally, optimal security is achieved by recent work in theoretical cryptography enabling any computation over encrypted data [GGP10]; however, such proposals are prohibitively impractical. In contrast, CryptDB is practical, and in §3.6.2, we demonstrate that it also provides significant security in practice. Specifically, we show that all or almost all of the most sensitive fields in the tested applications remain encrypted with highly secure encryption schemes. For such fields, CryptDB provides optimal security, assuming their value is independent of the pattern in which they are accessed (which is the case for medical information, social security numbers, etc). CryptDB is not optimal for fields requiring more revealing encryption schemes, but we find that most such fields are semi-sensitive (such as timestamps).

Finally, we believe that a passive attack model is realistic because malicious DBAs are more likely to read the data, which may be hard to detect, than to change the data or query results, which is more likely to be discovered. In §8.2.1, we cite related work on data integrity that could be used in complement with our work. An active adversary that can insert or update data may be able to indirectly compromise confidentiality. For example, an adversary that modifies an email field in the database may be able to trick the application into sending a user's data to the wrong email address, when the user asks the application to email her a copy of her own data. Such active attacks on the DBMS fall under the second threat model, which we now discuss.

□ 3.3 Queries over Encrypted Data

This section describes how CryptDB executes SQL queries over encrypted data. The DBMS machines and administrators are not trusted, but the application and the proxy are trusted.

CryptDB enables the DBMS server to execute SQL queries on encrypted data almost as if it were executing the same queries on plaintext data. Existing applications do not need to be changed. The DBMS's query plan for an encrypted query is typically the same as for the original query, except that the operators comprising the query, such as selections, projections, joins, aggregates, and orderings, are performed on ciphertexts, and use modified operators in some cases.

CryptDB's proxy stores a secret master key MK, the database schema, and the current encryption layers of all columns. The DBMS server sees an anonymized schema (in which table and column names are replaced by opaque identifiers), encrypted user data, and some auxiliary tables used by CryptDB. CryptDB also equips the server with CryptDB-specific user-defined functions (UDFs) that enable the server to compute on ciphertexts for certain operations.

Processing a query in CryptDB involves four steps:

1. The application issues a query, which the proxy intercepts and rewrites: it anonymizes each table and column name, and, using the master key MK, encrypts each constant in the query with an encryption scheme best suited for the desired operation (§3.3.1).

2. The proxy checks if the DBMS server should be given keys to adjust encryption layers before executing the query, and if so, issues an `UPDATE` query at the DBMS server that invokes a UDF to adjust the encryption layer of the appropriate columns (§3.3.2).

3. The proxy forwards the encrypted query to the DBMS server, which executes it using standard SQL (occasionally invoking UDFs for aggregation or keyword search).

4. The DBMS server returns the (encrypted) query result, which the proxy decrypts and returns to the application.

◇ 3.3.1 SQL-aware Encryption

We now describe the encryption types that CryptDB uses, including a number of existing cryptosystems, an optimization of a recent scheme, and a new cryptographic primitive for joins. For each encryption type, we explain the security property that CryptDB requires from it, its functionality, and how it is implemented.

Random (RND). RND provides the maximum security in CryptDB: indistinguishability under an adaptive chosen-plaintext attack (IND-CPA); the scheme is probabilistic, meaning that two equal values are mapped to different ciphertexts with overwhelming probability. On the other hand, RND does not allow any computation to be performed efficiently on the ciphertext. An efficient construction of RND is to use a block cipher like AES or Blowfish in CBC mode together with a random initialization vector (IV). (We mostly use AES, except for integer values, where we use Blowfish for its 64-bit block size because the 128-bit block size of AES would cause the ciphertext to be significantly longer).

Since, in this threat model, CryptDB assumes the server does not change results, CryptDB does not require a stronger IND-CCA2 construction (which would be secure under a chosen-ciphertext attack). However, it would be straightforward to use an IND-CCA2-secure implementation of RND instead, such as a block cipher in UFE mode [Des00], if needed.

Deterministic (DET). DET has a slightly weaker guarantee, yet it still provides strong security: it leaks only which encrypted values correspond to the same data value, by deterministically generating the same ciphertext for the same plaintext. This encryption layer allows the server to perform equality checks, which means it can perform selects with equality predicates, equality joins, `GROUP BY`, `COUNT`, `DISTINCT`, etc.

In cryptographic terms, DET should be a pseudo-random permutation (PRP) [Gol01]. For 64-bit and 128-bit values, we use a block cipher with a matching block size (Blowfish and AES respectively); we make the usual assumption that the AES and Blowfish block ciphers are PRPs. We pad smaller values out to 64 bits, but for data that is longer than a single 128-bit AES block, the standard CBC mode of operation leaks prefix equality (e.g., if two data items have an identical prefix that is at least 128 bits long). To avoid this problem, we use AES with a variant of the CMC mode [HR03], which can be approximately thought of as one round of CBC, followed by another round of CBC with the blocks in the reverse order. Since the goal of DET is to reveal equality, we use a zero IV (or "tweak" [HR03]) for our AES-CMC implementation of DET.

Order-preserving encryption (OPE). OPE allows order relations between data items to be established based on their encrypted values, without revealing the data itself. If $x < y$, then $\text{OPE}_K(x) < \text{OPE}_K(y)$, for any secret key K. Therefore, if a column is encrypted with OPE, the server can perform range queries when given encrypted constants $\text{OPE}_K(c_1)$ and $\text{OPE}_K(c_2)$ corresponding to the range $[c_1, c_2]$. The server can also perform `ORDER BY`, `MIN`, `MAX`, `SORT`, etc.

OPE is a weaker encryption scheme than DET because it reveals order. Thus, the CryptDB proxy will only reveal OPE-encrypted columns to the server if users request order queries on those columns. OPE has provable security guarantees [BCLO09]: the encryption is equivalent to a random mapping that preserves order.

The scheme we use [BCLO09] is the first provably secure such scheme. Until CryptDB, there was no implementation nor any measure of the practicality of the scheme. The direct implementation of the scheme took 25 ms per encryption of a 32-bit integer on an Intel 2.8 GHz Q9550 processor. We improved the algorithm by using AVL binary search trees for batch encryption (e.g., database loads), reducing the cost of OPE encryption to 7 ms per encryption without affecting its security. We also implemented a hypergeometric sampler that lies at the core of OPE, porting a Fortran implementation from 1988 [KS88].

We remark that, after publishing CryptDB, we designed a new order-preserving encryption scheme, called mOPE [PLZ13]. The ideal security property from an OPE scheme is that it reveals no information to the server about the plaintext data other than its order. The scheme above [BCLO09] reveals some information beyond order; in constrast, mOPE is the first scheme to reveal only order. Our implementation and evaluation results in this chapter are based on [BCLO09]. Nevertheless, in our mOPE paper [PLZ13], we compare mOPE with the scheme of [BCLO09] and show that mOPE is significantly faster in many cases.

Homomorphic encryption (HOM). HOM is a secure probabilistic encryption scheme (IND-CPA secure), allowing the server to perform computations on encrypted data with the final result decrypted at the proxy. While fully homomorphic encryption is prohibitively slow [Coo09], homomorphic encryption for specific operations is efficient. To support summation, we implemented the Paillier cryptosystem [Pai99]. With Paillier, multiplying the encryptions of two values results in an encryption of the sum of the values, i.e., $\text{HOM}_K(x) \cdot \text{HOM}_K(y) = \text{HOM}_K(x+y)$, where the multiplication is performed modulo some public-key value. To compute `SUM` aggregates, the proxy replaces `SUM` with calls to a UDF that performs Paillier multiplication on a column encrypted with HOM. HOM can also be used for computing averages by having the DBMS server return the sum and the count separately, and for incrementing values (e.g., `SET` *`id`*`=`*`id`*`+1`), on which we elaborate shortly.

With HOM, the ciphertext is 2048 bits. In theory, it should be possible to pack multiple values from a single row into one HOM ciphertext for that row, using the scheme of Ge and Zdonik [GZ07], which would result in an amortized space overhead of $2\times$ (e.g., a 32-bit value occupies 64 bits) for a table with many HOM-encrypted columns. However, we have not implemented this optimization in our prototype. This optimization would also complicate partial-row `UPDATE` operations that reset some—but not all—of the values packed into a HOM ciphertext.

Join (JOIN and OPE-JOIN). A separate encryption scheme is necessary to allow equality joins between two columns, because we use different keys for DET to prevent cross-column correlations. JOIN also supports all operations allowed by DET, and also enables the server to determine repeating values between two columns. OPE-JOIN enables joins by order relations. We provide a new cryptographic scheme for JOIN and we discuss it in §3.3.4.

Word search (SEARCH). SEARCH is used to perform searches on encrypted text to support operations such as MySQL's `LIKE` operator. We implemented the cryptographic protocol of Song et al. [SWP00], which was not previously implemented by the authors; we also use their protocol in a different way, which results in better security guarantees. For each column needing SEARCH, we split the text into keywords using standard delimiters (or using a special keyword extraction function specified by the schema developer). We then remove repetitions in these words, randomly permute the positions of the words, and then encrypt each of the words using Song et al.'s scheme, padding each word to the same size. SEARCH is nearly as secure as RND: the encryption does not reveal to the DBMS server whether a certain word repeats in multiple rows, but it leaks the number of keywords encrypted with SEARCH; an adversary may be able to estimate the number of distinct or duplicate words (e.g., by comparing the size of the SEARCH and RND ciphertexts for the same data).

When the user performs a query such as `SELECT * FROM` *`messages`* `WHERE` *`msg`* `LIKE "% alice %"`, the proxy gives the DBMS server a token, which is an encryption of `alice`. The server cannot decrypt the token to figure out the underlying word. Using a user-defined function, the DBMS server checks if any of the word encryptions in any message match the token. In our approach, all the server learns from searching is whether a token matched a message or not, and this happens only for the tokens requested by the user. The server would learn the same information when returning the result set to the users, so the overall search scheme reveals the minimum amount of additional information needed to return the result.

Note that SEARCH allows CryptDB to only perform full-word keyword searches; it cannot support arbitrary regular expressions. For applications that require searching for multiple adjacent words, CryptDB allows the application developer to disable duplicate removal and re-ordering by

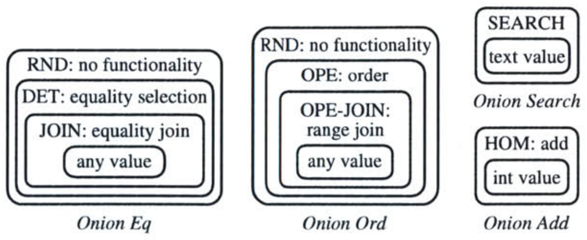

Figure 3-2: Onion encryption layers and the classes of computation they allow. Onion names stand for the operations they allow at some of their layers (Equality, Order, Search, and Addition). In practice, some onions or onion layers may be omitted, depending on column types or schema annotations provided by application developers (§3.3.5). DET and JOIN are often merged into a single onion layer, since JOIN is a concatenation of DET and JOIN-ADJ (§3.3.4). A random IV for RND (§3.3.1), shared by the RND layers in *Eq* and *Ord*, is also stored for each data item.

annotating the schema, even though this is not the default. Based on our trace evaluation, we find that most uses of `LIKE` can be supported by SEARCH with such schema annotations. Of course, one can still combine multiple `LIKE` operators with `AND` and `OR` to check whether multiple independent words are in the text.

◇ 3.3.2 Adjustable Query-based Encryption

A key part of CryptDB's design is *adjustable query-based encryption*, which dynamically adjusts the layer of encryption on the DBMS server. Our goal is to use the most secure encryption schemes that enable running the requested queries. For example, if the application issues no queries that compare data items in a column, or that sort a column, the column should be encrypted with RND. For columns that require equality checks but not inequality checks, DET suffices. However, the query set is not always known in advance. Thus, we need an adaptive scheme that dynamically adjusts encryption strategies.

Our idea is to encrypt each data item in one or more *onions*: that is, each value is dressed in layers of increasingly stronger encryption, as illustrated in Figures 3-2 and 3-3. Each layer of each onion enables certain kinds of functionality as explained in the previous subsection. For example, outermost layers such as RND and HOM provide maximum security, whereas inner layers such as OPE provide more functionality.

Multiple onions are needed in practice, both because the computations supported by different encryption schemes are not always strictly ordered, and because of performance considerations (size of ciphertext and encryption time for nested onion layers). Depending on the type of the data (and any annotations provided by the application developer on the database schema, as discussed in §3.3.5), CryptDB may not maintain all onions for each column. For instance, the *Search* onion does not make sense for integers, and the *Add* onion does not make sense for strings.

For each layer of each onion, the proxy uses the same key for encrypting values in the same column, and different keys across tables, columns, onions, and onion layers. Using the same key for all values in a column allows the proxy to perform operations on a column without having to compute separate keys for each row that will be manipulated. Using different keys across columns prevents the server from learning any additional relations. All of these keys are derived from the master key MK. For example, for table t, column c, onion o, and encryption layer l, the proxy uses

Employees

ID	*Name*
23	Alice

Table1

C1-IV	*C1-Eq*	*C1-Ord*	*C1-Add*	*C2-IV*	*C2-Eq*	*C2-Ord*	*C2-Search*
x27c3	x2b82	xcb94	xc2e4	x8a13	xd1e3	x7eb1	x29b0

Figure 3-3: Data layout at the server. When the application creates the table shown on the left, the table created at the DBMS server is the one shown on the right. Ciphertexts shown are not full-length.

the key

$$K_{t,c,o,l} = \text{PRP}_{MK}(\text{table } t,\ \text{column } c,\ \text{onion } o,\ \text{layer } l), \tag{3.1}$$

where PRP is a pseudorandom permutation (e.g., AES).

Each onion starts out encrypted with the most secure encryption scheme (RND for onions *Eq* and *Ord*, HOM for onion *Add*, and SEARCH for onion *Search*). As the proxy receives SQL queries from the application, it determines whether layers of encryption need to be removed. Given a predicate P on column c needed to execute a query on the server, the proxy first establishes what onion layer is needed to compute P on c. If the encryption of c is not already at an onion layer that allows P, the proxy strips off the onion layers to allow P on c, by sending the corresponding onion key to the server. The proxy never decrypts the data past the least-secure encryption onion layer (or past some other threshold layer, if specified by the application developer in the schema, §3.3.5).

CryptDB implements onion layer decryption using UDFs that run on the DBMS server. For example, in Figure 3-3, to decrypt onion *Ord* of column 2 in table 1 to layer OPE, the proxy issues the following query to the server using the `DECRYPT_RND` UDF:

```
UPDATE Table1 SET
            C2-Ord = DECRYPT_RND(K, C2-Ord, C2-IV)
```

where K is the appropriate key computed from Equation (3.1). At the same time, the proxy updates its own internal state to remember that column *C2-Ord* in *Table1* is now at layer OPE in the DBMS. Each column decryption should be included in a transaction to avoid consistency problems with clients accessing columns being adjusted.

Note that onion decryption is performed entirely by the DBMS server. In the steady state, no server-side decryptions are needed, because onion decryption happens only when a new class of computation is requested on a column. For example, after an equality check is requested on a column and the server brings the column to layer DET, the column remains in that state, and future queries with equality checks require no decryption. This property is the insight into why CryptDB's overhead is modest in the steady state (see §3.6): the server mostly performs typical SQL processing.

◇ 3.3.3 Executing over Encrypted Data

Once the onion layers in the DBMS are at the layer necessary to execute a query, the proxy transforms the query to operate on these onions. In particular, the proxy replaces column names in a query with corresponding onion names, based on the class of computation performed on that column. For example, for the schema shown in Figure 3-3, a reference to the *Name* column for an equality comparison will be replaced with a reference to the *C2-Eq* column.

The proxy also replaces each constant in the query with a corresponding onion encryption of that constant, based on the computation in which it is used. For instance, if a query contains `WHERE Name`

= `'Alice'`, the proxy encrypts 'Alice' by successively applying all encryption layers corresponding to onion *Eq* that have not yet been removed from *C2-Eq*.

Finally, the server replaces certain operators with UDF-based counterparts. For instance, the `SUM` aggregate operator and the + column-addition operator must be replaced with an invocation of a UDF that performs HOM addition of ciphertexts. Equality and order operators (such as = and <) do not need such replacement and can be applied directly to the DET and OPE ciphertexts.

Once the proxy has transformed the query, it sends the query to the DBMS server, receives query results (consisting of encrypted data), decrypts the results using the corresponding onion keys, and sends the decrypted result to the application.

Read query execution. To understand query execution over ciphertexts, consider the example schema shown in Figure 3-3. Initially, each column in the table is dressed in all onions of encryption, with RND, HOM, and SEARCH as outermost layers, as shown in Figure 3-2. At this point, the server can learn nothing about the data other than the number of columns, rows, and data size.

To illustrate when onion layers are removed, consider the query:

`SELECT ID FROM Employees WHERE Name = 'Alice'`,

which requires lowering the encryption of *Name* to layer DET. To execute this query, the proxy first issues the query

`UPDATE Table1 SET C2-Eq = DECRYPT_RND(`$K_{T1,C2,Eq,\mathrm{RND}}$`, C2-Eq, C2-IV)`,

where column *C2* corresponds to *Name*. The proxy then issues

`SELECT C1-Eq, C1-IV FROM Table1 WHERE C2-Eq = x7..d`,

where column *C1* corresponds to *ID*, and where x7..d is the *Eq* onion encryption of "Alice" with keys $K_{T1,C2,Eq,\mathrm{JOIN}}$ and $K_{T1,C2,Eq,\mathrm{DET}}$ (see Figure 3-2). Note that the proxy must request the random IV from column `C1-IV` in order to decrypt the RND ciphertext from `C1-Eq`. Finally, the proxy decrypts the results from the server using keys $K_{T1,C1,Eq,\mathrm{RND}}$, $K_{T1,C1,Eq,\mathrm{DET}}$, and $K_{T1,C1,Eq,\mathrm{JOIN}}$, obtains the result 23, and returns it to the application.

If the next query is `SELECT COUNT(*) FROM Employees WHERE Name = 'Bob'`, no server-side decryptions are necessary, and the proxy directly issues the query `SELECT COUNT(*) FROM Table1 WHERE C2-Eq = xbb..4a`, where xbb..4a is the *Eq* onion encryption of "Bob" using $K_{T1,C2,Eq,\mathrm{JOIN}}$ and $K_{T1,C2,Eq,\mathrm{DET}}$.

Write query execution. To support `INSERT`, `DELETE`, and `UPDATE` queries, the proxy applies the same processing to the predicates (i.e., the `WHERE` clause) as for read queries. `DELETE` queries require no additional processing. For all `INSERT` and `UPDATE` queries that set the value of a column to a constant, the proxy encrypts each inserted column's value with each onion layer that has not yet been stripped off in that column.

The remaining case is an `UPDATE` that sets a column value based on an existing column value, such as `salary=salary+1`. Such an update would have to be performed using HOM, to handle additions. However, in doing so, the values in the OPE and DET onions would become stale. In fact, any hypothetical encryption scheme that simultaneously allows addition and direct comparison on the ciphertext is insecure: if a malicious server can compute the order of the items, and can increment the value by one, the server can repeatedly add one to each field homomorphically until it becomes equal to some other value in the same column. This would allow the server to compute the difference between any two values in the database, which is almost equivalent to knowing their values.

There are two approaches to allow updates based on existing column values. If a column is incremented and then only projected (no comparisons are performed on it), the solution is simple: when a query requests the value of this field, the proxy should request the HOM ciphertext from the *Add* onion, instead of ciphertexts from other onions, because the HOM value is up-to-date. For instance, this approach applies to increment queries in TPC-C. If a column is used in comparisons after it is incremented, the solution is to replace the update query with two queries: a `SELECT` of the old values to be updated, which the proxy increments and encrypts accordingly, followed by an `UPDATE` setting the new values. This strategy would work well for updates that affect a small number of rows.

Other DBMS features. Most other DBMS mechanisms, such as transactions and indexing, work the same way with CryptDB over encrypted data as they do over plaintext, with no modifications. For transactions, the proxy passes along any `BEGIN`, `COMMIT`, and `ABORT` queries to the DBMS. Since many SQL operators behave differently on NULLs than on non-NULL values, CryptDB exposes NULL values to the DBMS without encryption. CryptDB does not currently support stored procedures, although certain stored procedures could be supported by rewriting their code in the same way that CryptDB's proxy rewrites SQL statements.

The DBMS builds indexes for encrypted data in the same way as for plaintext. Currently, if the application requests an index on a column, the proxy asks the DBMS server to build indexes on that column's DET, JOIN, OPE, or OPE-JOIN onion layers (if they are exposed), but not for RND, HOM, or SEARCH. More efficient index selection algorithms could be investigated.

◇ 3.3.4 Computing Joins

There are two kinds of joins supported by CryptDB: *equi-joins*, in which the join predicate is based on equality, and *range joins*, which involve order checks. To perform an equi-join of two encrypted columns, the columns should be encrypted with the same key so that the server can see matching values between the two columns. At the same time, to provide better privacy, the DBMS server should not be able to join columns for which the application did not request a join, so columns that are never joined should not be encrypted with the same keys.

If the queries that can be issued, or the pairs of columns that can be joined, are known *a priori*, equi-join is easy to support: CryptDB can use the DET encryption scheme with the same key for each group of columns that are joined together. §3.3.5 describes how the proxy learns the columns to be joined in this case. However, the challenging case is when the proxy does not know the set of columns to be joined *a priori*, and hence does not know which columns should be encrypted with matching keys.

To solve this problem, we introduce a new cryptographic primitive, JOIN-ADJ (*adjustable join*), which allows the DBMS server to adjust the key of each column at runtime. Intuitively, JOIN-ADJ can be thought of as a keyed cryptographic hash with the additional property that hashes can be adjusted to change their key *without access to the plaintext*. JOIN-ADJ is a deterministic function of its input, which means that if two plaintexts are equal, the corresponding JOIN-ADJ values are also equal. JOIN-ADJ is collision-resistant, and has a sufficiently long output length (192 bits) to allow us to assume that collisions never happen in practice.

JOIN-ADJ is non-invertible, so we define the JOIN encryption scheme as

$$\text{JOIN}(v) = \text{JOIN-ADJ}(v) \,\|\, \text{DET}(v),$$

where $\|$ denotes concatenation. This construction allows the proxy to decrypt a JOIN(v) column to obtain v by decrypting the DET component, and allows the DBMS server to check two JOIN values for equality by comparing the JOIN-ADJ components.

Each column is initially encrypted at the JOIN layer using a different key, thus preventing any joins between columns. When a query requests a join, the proxy gives the DBMS server an onion key to adjust the JOIN-ADJ values in one of the two columns, so that it matches the JOIN-ADJ key of the other column (denoted the *join-base* column). After the adjustment, the columns share the same JOIN-ADJ key, allowing the DBMS server to join them for equality. The DET components of JOIN remain encrypted with different keys.

Note that our adjustable join is transitive: if the user joins columns A and B and then joins columns B and C, the server can join A and C. However, the server cannot join columns in different "transitivity groups". For instance, if columns D and E were joined together, the DBMS server would not be able to join columns A and D on its own.

After an initial join query, the JOIN-ADJ values remain transformed with the same key, so no re-adjustments are needed for subsequent join queries between the same two columns. One exception is if the application issues another query, joining one of the adjusted columns with a third column, which causes the proxy to re-adjust the column to another join-base. To avoid oscillations and to converge to a state where all columns in a transitivity group share the same join-base, CryptDB chooses the first column in lexicographic order on table and column name as the join-base. For n columns, the overall maximum number of join transitions is $n(n-1)/2$.

For range joins, a similar dynamic re-adjustment scheme is difficult to construct due to lack of structure in OPE schemes. Instead, CryptDB requires that pairs of columns that will be involved in such joins be declared by the application ahead of time, so that matching keys are used for layer OPE-JOIN of those columns; otherwise, the same key will be used for all columns at layer OPE-JOIN. Fortunately, range joins are rare; they are not used in any of our example applications, and are used in only 50 out of 128,840 columns in a large SQL query trace we describe in §3.6, corresponding to just three distinct applications.

JOIN-ADJ construction. We now present an overview of our JOIN-ADJ construction. For a formal cryptographic exposition and proof of security, see [PZ12]. Our algorithm uses elliptic-curve cryptography (ECC). JOIN-ADJ$_K(v)$ is computed as

$$\text{JOIN-ADJ}_K(v) := P^{K \cdot \text{PRF}_{K_0}(v)}, \tag{3.2}$$

where K is the initial key for that table, column, onion, and layer, P is a point on an elliptic curve (being a public parameter), and PRF$_{K_0}$ is a pseudo-random function [Gol01] mapping values to a pseudorandom number, such as AES$_{K_0}$(SHA(v)), with K_0 being a key that is the same for all columns and derived from MK. The "exponentiation" is in fact repeated geometric addition of elliptic curve points; it is considerably faster than RSA exponentiation.

When a query joins columns c and c', each having keys K and K' at the join layer, the proxy computes $\Delta K = K/K'$ (in an appropriate group) and sends it to the server. Then, given JOIN-ADJ$_{K'}(v)$ (the JOIN-ADJ values from column c') and ΔK, the DBMS server uses a UDF to adjust the key in c' by computing:

$$\begin{aligned}(\text{JOIN-ADJ}_{K'}(v))^{\Delta K} &= P^{K' \cdot \text{PRF}_{K_0}(v) \cdot (K/K')} \\ &= P^{K \cdot \text{PRF}_{K_0}(v)} = \text{JOIN-ADJ}_K(v).\end{aligned}$$

Now columns c and c' share the same JOIN-ADJ key, and the DBMS server can perform an equi-join on c and c' by taking the JOIN-ADJ component of the JOIN onion ciphertext.

At a high level, the security of this scheme is that the server cannot infer join relations among groups of columns that were not requested by legitimate join queries, and that the scheme does not reveal the plaintext. We proved the security of this scheme based on the standard Elliptic-Curve Decisional Diffie-Hellman hardness assumption (see [PZ12]), and implemented it using a NIST-approved elliptic curve.

◇ 3.3.5 Improving Security and Performance

Although CryptDB can operate with an unmodified and unannotated schema, as described above, its security and performance can be improved through several optional optimizations, as described below.

▷ Security Improvements:

The "sensitive" annotation. Data owners can specify that some data columns are sensitive. In this case, CryptDB's proxy will ensure that the data in those columns is encrypted only with strong encryption schemes, providing semantic security [Gol04] or a similarly strong security guarantee. This strategy means that virtually no information leaks about the data items in those columns, other than their lengths. The encryption schemes that can be used with such columns are RND, HOM and, if the fields in that column are unique, DET as well. This means that a number of operations can still run on top of these columns, although the set of these operations is more limited than CryptDB's full set. The operations supported are select/insert/update/delete data items, addition, and, if the data items in the column are unique, equality.

Minimum onion layers. Application developers can specify the lowest onion encryption layer that may be revealed to the server for a specific column. In this way, the developer can ensure that the proxy will not execute queries exposing sensitive relations to the server. For example, the developer could specify that credit card numbers should always remain at RND or DET.

In-proxy processing. Although CryptDB can evaluate a number of predicates on the server, evaluating them in the proxy can improve security by not revealing additional information to the server. One common use case is a `SELECT` query that sorts on one of the selected columns, without a `LIMIT` on the number of returned columns. Since the proxy receives the entire result set from the server, sorting these results in the proxy does not require a significant amount of computation, and does not increase the bandwidth requirements. Doing so avoids revealing the OPE encryption of that column to the server.

Training mode. CryptDB provides a training mode, which allows a developer to provide a trace of queries and get the resulting onion encryption layers for each field, along with a warning in case some query is not supported. The developer can then examine the resulting encryption levels to understand what each encryption scheme leaks, as described in §4.2. If some onion level is too low for a sensitive field, she should arrange to have the query processed in the proxy (as described above), or to process the data in some other fashion, such as by using a local instance of SQLite.

Onion re-encryption. In cases when an application performs infrequent queries requiring a low onion layer (e.g., OPE), CryptDB could be extended to re-encrypt onions back to a higher layer after the infrequent query finishes executing. This approach reduces leakage to attacks happening in the time window when the data is at the higher onion layer.

▷ **Performance Optimizations:**

Developer annotations. By default, CryptDB encrypts all fields and creates all applicable onions for each data item based on its type. If many columns are not sensitive, the developer can instead provide explicit annotations indicating the sensitive fields, and leave the remaining fields in plaintext.

Known query set. If the developer knows some of the queries ahead of time, as is the case for many web applications, the developer can use the training mode described above to adjust onions to the correct layer *a priori*, avoiding the overhead of runtime onion adjustments. If the developer provides the exact query set, or annotations that certain functionality is not needed on some columns, CryptDB can also discard onions that are not needed (e.g., discard the *Ord* onion for columns that are not used in range queries, or discard the *Search* onion for columns where keyword search is not performed), discard onion layers that are not needed (e.g., the adjustable JOIN layer, if joins are known *a priori*), or discard the random IV needed for RND for some columns.

Ciphertext pre-computing and caching. The proxy spends a significant amount of time encrypting values used in queries with OPE and HOM. To reduce this cost, the proxy pre-computes (for HOM) and caches (for OPE) encryptions of frequently used constants under different keys. Since HOM is probabilistic, ciphertexts cannot be reused. Therefore, in addition, the proxy pre-computes HOM's Paillier r^n randomness values for future encryptions of any data. This optimization reduces the amount of CPU time spent by the proxy on OPE encryption, and assuming the proxy is occasionally idle to perform HOM pre-computation, it removes HOM encryption from the critical path.

□ 3.4 Discussion

CryptDB's design supports most relational queries and aggregates on standard data types, such as integers and text/varchar types. Additional operations can be added to CryptDB by extending its existing onions, or adding new onions for specific data types (e.g., spatial and multi-dimensional range queries [SBC+07]). Alternatively, in some cases, it may be possible to map complex unsupported

	Databases	Tables	Columns
Complete schema	8,548	177,154	1,244,216
Used in query	1,193	18,162	128,840

Figure 3-4: Number of databases, tables, and columns on the `sql.mit.edu` MySQL server, used for trace analysis, indicating the total size of the schema, and the part of the schema seen in queries during the trace period.

operation to simpler ones (e.g., extracting the month out of an encrypted date is easier if the date's day, month, and year fields are encrypted separately).

There are certain computations CryptDB cannot support on encrypted data. For example, it does not support both computation and comparison on the same column, such as `WHERE` *`salary`* `>` *`age`*`*2+10`. CryptDB can process a part of this query, but it would also require some processing on the proxy. In CryptDB, such a query should be (1) rewritten into a sub-query that selects a whole column, `SELECT` *`age`*`*2+10 FROM` ..., which CryptDB computes using HOM, and (2) re-encrypted in the proxy, creating a new column (call it *aux*) on the DBMS server consisting of the newly encrypted values. Finally, the original query with the predicate `WHERE` *`salary`* `>` ***`aux`*** should be run. We have not been affected by this limitation in our test applications (TPC-C, phpBB, HotCRP, and grad-apply).

□ 3.5 Implementation

The CryptDB proxy consists of a C++ library and a Lua module. The C++ library consists of a query parser; a query encryptor/rewriter, which encrypts fields or includes UDFs in the query; and a result decryption module. To allow applications to transparently use CryptDB, we used MySQL proxy [Tay] and implemented a Lua module that passes queries and results to and from our C++ module. We implemented our new cryptographic protocols using NTL [Sho09]. Our CryptDB implementation consists of ~18,000 lines of C++ code and ~150 lines of Lua code, with another ~10,000 lines of test code.

CryptDB is portable and we have implemented versions for both Postgres 9.0 and MySQL 5.1. Porting CryptDB from Postgres to MySQL required changing only 86 lines of code, mostly in the code for connecting to the MySQL server and declaring UDFs. As mentioned earlier, CryptDB does not change the DBMS; we implement all server-side functionality with UDFs and server-side tables. CryptDB's design, and to a large extent our implementation, should work on top of any SQL DBMS that supports UDFs.

□ 3.6 Experimental Evaluation

In this section, we evaluate three aspects of CryptDB: the types of queries and applications CryptDB is able to support, the level of security CryptDB provides, and the performance impact of using CryptDB. For this analysis, we use seven applications as well as a large trace of SQL queries.

We note that CryptDB does not require any changes to existing applications because the CryptDB proxy exports a SQL interface. This means that one can take an SQL-backed application that exists today and run it on top of CryptDB with no changes. Similarly, we made no changes to MySQL,

Application	Total cols.	Consider for enc.	Needs plaintext	Needs HOM	Needs SEARCH	Non-plaintext cols. with MinEnc: RND	SEARCH	DET	OPE	Most sensitive cols. at HIGH
phpBB	563	23	0	1	0	21	0	1	1	6 / 6
HotCRP	204	22	0	2	1	18	1	1	2	18 / 18
grad-apply	706	103	0	0	2	95	0	6	2	94 / 94
OpenEMR	1, 297	566	7	0	3	526	2	12	19	525 / 540
MIT 6.02	15	13	0	0	0	7	0	4	2	1 / 1
PHP-calendar	25	12	2	0	2	3	2	4	1	3 / 4
TPC-C	92	92	0	8	0	65	0	19	8	—
`sql.mit.edu`	128, 840	128, 840	1, 094	1, 019	1, 125	80, 053	350	34, 212	13, 131	—
. . . with in-proxy processing	128, 840	128, 840	571	1, 016	1, 135	84, 008	398	35, 350	8, 513	—
. . . contains *pass*	2, 029	2, 029	2	0	0	1, 936	0	91	0	—
. . . contains *content*	2, 521	2, 521	0	0	52	2, 215	52	251	3	—
. . . contains *priv*	173	173	0	4	0	159	0	12	2	—

Figure 3-5: Steady-state onion levels for database columns required by a range of applications and traces. "Needs plaintext" indicates that CryptDB cannot execute the application's queries over encrypted data for that column. For the applications in the top group of rows, sensitive columns were determined manually, and only these columns were considered for encryption. For the bottom group of rows, all database columns were automatically considered for encryption. For sql.mit.edu, we also show the results for columns containing a keyword (e.g., *priv*) which could indicate a sensitive field. The rightmost column considers the application's most sensitive database columns, and reports the number of them that have MinEnc in HIGH (both terms are defined in §3.6.2).

making it thus easier to port CryptDB to another DBMS.

We analyze the functionality and security of CryptDB on three more applications, on TPC-C, and on a large trace of SQL queries. The additional three applications are OpenEMR, an electronic medical records application storing private medical data of patients; the web application of an MIT class (6.02), storing students' grades; and PHP-calendar, storing people's schedules. The large trace of SQL queries comes from a popular MySQL server at MIT, `sql.mit.edu`. This server is used primarily by web applications running on `scripts.mit.edu`, a shared web application hosting service operated by MIT's Student Information Processing Board (SIPB). In addition, this SQL server is used by a number of applications that run on other machines and use `sql.mit.edu` only to store their data. Our query trace spans about ten days, and includes approximately 126 million queries. Figure 3-4 summarizes the schema statistics for `sql.mit.edu`; each database is likely to be a separate instance of some application.

Finally, we evaluate the overall performance of CryptDB on a query mix from TPC-C, and perform a detailed analysis through microbenchmarks.

In the six applications (not counting TPC-C), we only encrypt sensitive columns, according to a manual inspection. Some fields were clearly sensitive (e.g., grades, private message, medical information), but others were only marginally so (e.g., the time when a message was posted). There was no clear threshold between sensitive or not, but it was clear to us which fields were definitely sensitive. In the case of TPC-C, we encrypt all the columns in the database in single-principal mode so that we can study the performance and functionality of a fully encrypted DBMS. All fields are considered for encryption in the large query trace as well.

◇ 3.6.1 Functional Evaluation

To evaluate what columns, operations, and queries CryptDB can support, we analyzed the queries issued by six web applications, the TPC-C queries, and the SQL queries from `sql.mit.edu`. The results are shown in the left half of Figure 3-5.

CryptDB supports most queries; the number of columns in the "needs plaintext" column, which counts columns that cannot be processed in encrypted form by CryptDB, is small relative to the total number of columns. For PHP-calendar and OpenEMR, CryptDB does not support queries on certain sensitive fields that perform string manipulation (e.g., substring and lowercase conversions) or date manipulation (e.g., obtaining the day, month, or year of an encrypted date). However, if these functions were precomputed with the result added as standalone columns (e.g., each of the three parts of a date were encrypted separately), CryptDB would support these queries.

The next two columns, "needs HOM" and "needs SEARCH", reflect the number of columns for which that encryption scheme is needed to process some queries. The numbers suggest that these encryption schemes are important; without these schemes, CryptDB would be unable to support those queries.

Based on an analysis of the larger `sql.mit.edu` trace, we found that CryptDB should be able to support operations over all but 1,094 of the 128,840 columns observed in the trace. The "in-proxy processing" shows analysis results where we assumed the proxy can perform some lightweight operations on the results returned from the DBMS server. Specifically, this included any operations that are not needed to compute the set of resulting rows or to aggregate rows (that is, expressions that do not appear in a `WHERE`, `HAVING`, or `GROUP BY` clause, or in an `ORDER BY` clause with a `LIMIT`, and are not aggregate operators). With in-proxy processing, CryptDB should be able to process queries over encrypted data over all but 571 of the 128,840 columns, thus supporting 99.5% of the columns.

Of those 571 columns, 222 use a bitwise operator in a `WHERE` clause or perform bitwise aggregation, such as the Gallery2 application, which uses a bitmask of permission fields and consults them in `WHERE` clauses. Rewriting the application to store the permissions in a different way would allow CryptDB to support such operations. Another 205 columns perform string processing in the `WHERE` clause, such as comparing whether lowercase versions of two strings match. Storing a keyed hash of the lowercase version of each string for such columns, similar to the JOIN-ADJ scheme, could support case-insensitive equality checks for ciphertexts. 76 columns are involved in mathematical transformations in the `WHERE` clause, such as manipulating dates, times, scores, and geometric coordinates. 41 columns invoke the `LIKE` operator with a column reference for the pattern; this is typically used to check a particular value against a table storing a list of banned IP addresses, usernames, URLs, etc. Such a query can also be rewritten if the data items are sensitive.

◇ 3.6.2 Security Evaluation

To understand the amount of information that would be revealed to the adversary in practice, we examine the steady-state onion levels of different columns for a range of applications and queries. To quantify the level of security, we define the MinEnc of a column to be the weakest onion encryption scheme exposed on any of the onions of a column when onions reach a steady state (i.e., after the application generates all query types, or after running the whole trace). We consider RND and HOM to be the strongest schemes, followed by SEARCH, followed by DET and JOIN, and finishing with the weakest scheme which is OPE. For example, if a column has onion *Eq* at RND, onion *Ord* at OPE and onion *Add* at HOM, the MinEnc of this column is OPE.

The right side of Figure 3-5 shows the MinEnc onion level for a range of applications and query traces. We see that most fields remain at RND, which is the most secure scheme. For example, OpenEMR has hundreds of sensitive fields describing the medical conditions and history of patients, but these fields are mostly just inserted and fetched, and are not used in any computation. A number of fields also remain at DET, typically to perform key lookups and joins. OPE, which leaks order, is used the least frequently, and mostly for fields that are marginally sensitive (e.g., timestamps and counts of messages). Thus, CryptDB's adjustable security provides a significant improvement in confidentiality over revealing all encryption schemes to the server.

To analyze CryptDB's security for specific columns that are particularly sensitive, we define a new security level, HIGH, which includes the RND and HOM encryption schemes, as well as DET for columns having no repetitions (in which case DET is logically equivalent to RND). These are highly secure encryption schemes leaking virtually nothing about the data. DET for columns with repeats and OPE are not part of HIGH as they reveal relations to the DBMS server. The rightmost column in Figure 3-5 shows that most of the particularly sensitive columns (again, according to manual inspection) are at HIGH.

For the `sql.mit.edu` trace queries, approximately 6.6% of columns were at OPE even with in-proxy processing; other encrypted columns (93%) remain at DET or above. Out of the columns that were at OPE, 3.9% are used in an `ORDER BY` clause with a `LIMIT`, 3.7% are used in an inequality comparison in a `WHERE` clause, and 0.25% are used in a `MIN` or `MAX` aggregate operator (some of the columns are counted in more than one of these groups). It would be difficult to perform these computations in the proxy without substantially increasing the amount of data sent to it.

Although we could not examine the schemas of applications using `sql.mit.edu` to determine what fields are sensitive—mostly due to its large scale—we measured the same statistics as above for columns whose names are indicative of sensitive data. In particular, the last three rows of Figure 3-5 show columns whose name contains the word "pass" (which are almost all some type of password), "content" (which are typically bulk data managed by an application), and "priv" (which are typically some type of private message). CryptDB reveals much less information about these columns than an average column, almost all of them are supported, and almost all are at RND or DET.

Finally, we empirically validated CryptDB's confidentiality guarantees by trying real attacks on phpBB that have been listed in the CVE database [Nat11], including two SQL injection attacks (CVE-2009-3052 & CVE-2008-6314), bugs in permission checks (CVE-2010-1627 & CVE-2008-7143), and a bug in remote PHP file inclusion (CVE-2008-6377). We found that, for users not currently logged in, the answers returned from the DBMS were encrypted; even with root access to the application server, proxy, and DBMS, the answers were not decryptable.

◇ 3.6.3 Performance Evaluation

To evaluate the performance of CryptDB, we used a machine with two 2.4 GHz Intel Xeon E5620 4-core processors and 12 GB of RAM to run the MySQL 5.1.54 server, and a machine with eight 2.4 GHz AMD Opteron 8431 6-core processors and 64 GB of RAM to run the CryptDB proxy and the clients. The two machines were connected over a shared Gigabit Ethernet network. The higher-provisioned client machine ensures that the clients are not the bottleneck in any experiment. All workloads fit in the server's RAM.

We compare the performance of a TPC-C query mix when running on an unmodified MySQL server versus on a CryptDB proxy in front of the MySQL server. We trained CryptDB on the query

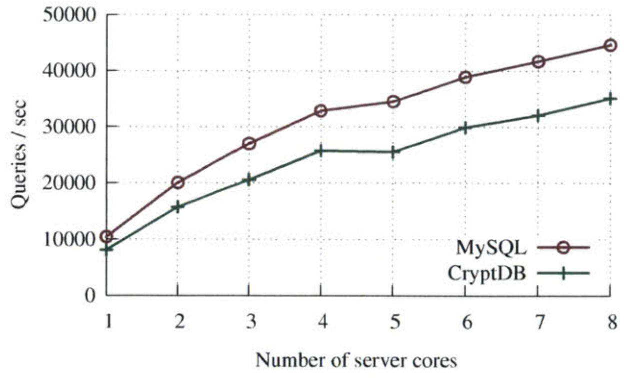

Figure 3-6: Throughput for TPC-C queries, for a varying number of cores on the underlying MySQL DBMS server.

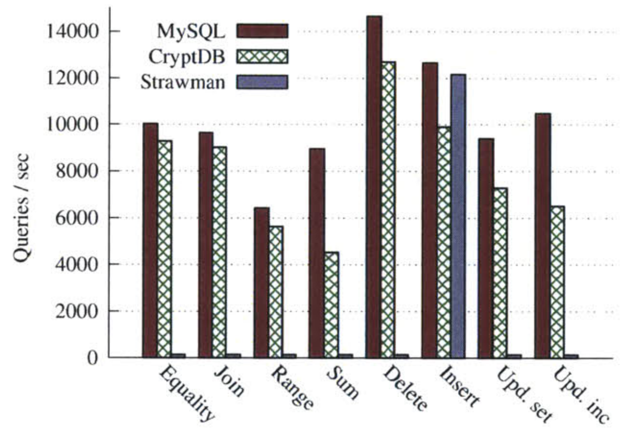

Figure 3-7: Throughput of different types of SQL queries from the TPC-C query mix running under MySQL, CryptDB, and the strawman design. "Upd. inc" stands for `UPDATE` that increments a column, and "Upd. set" stands for `UPDATE` which sets columns to a constant.

set (§3.3.5) so there are no onion adjustments during the TPC-C experiments. Figure 3-6 shows the throughput of TPC-C queries as the number of cores on the server varies from one to eight. In all cases, the server spends 100% of its CPU time processing queries. Both MySQL and CryptDB scale well initially, but start to level off due to internal lock contention in the MySQL server, as reported by `SHOW STATUS LIKE 'Table%'`. The overall throughput with CryptDB is 21–26% lower than MySQL, depending on the exact number of cores.

To understand the sources of CryptDB's overhead, we measure the server throughput for different types of SQL queries seen in TPC-C, on the same server, but running with only one core enabled. Figure 3-7 shows the results for MySQL, CryptDB, and a *strawman* design; the strawman performs each query over data encrypted with RND by decrypting the relevant data using a UDF, performing the query over the plaintext, and re-encrypting the result (if updating rows). The results show that CryptDB's throughput penalty is greatest for queries that involve a `SUM` (2.0× less throughput) and for incrementing `UPDATE` statements (1.6× less throughput); these are the queries that involve HOM additions at the server. For the other types of queries, which form a larger part of the TPC-C mix, the throughput overhead is modest. The strawman design performs poorly for almost all queries because the DBMS's indexes on the RND-encrypted data are useless for operations on the underlying

Query (& scheme)	MySQL Server	CryptDB Server	CryptDB Proxy	CryptDB Proxy$\star$
Select by = (DET)	0.10 ms	0.11 ms	0.86 ms	0.86 ms
Select join (JOIN)	0.10 ms	0.11 ms	0.75 ms	0.75 ms
Select range (OPE)	0.16 ms	0.22 ms	**0.78** ms	28.7 ms
Select sum (HOM)	0.11 ms	0.46 ms	0.99 ms	0.99 ms
Delete	0.07 ms	0.08 ms	0.28 ms	0.28 ms
Insert (all)	0.08 ms	0.10 ms	**0.37** ms	16.3 ms
Update set (all)	0.11 ms	0.14 ms	**0.36** ms	3.80 ms
Update inc (HOM)	0.10 ms	0.17 ms	**0.30** ms	25.1 ms
Overall	0.10 ms	0.12 ms	**0.60** ms	10.7 ms

Figure 3-8: Server and proxy latency for different types of SQL queries from TPC-C. For each query type, we show the predominant encryption scheme used at the server. Due to details of the TPC-C workload, each query type affects a different number of rows, and involves a different number of cryptographic operations. The left two columns correspond to server throughput, which is also shown in Figure 3-7. "Proxy" shows the latency added by CryptDB's proxy; "Proxy$\star$" shows the proxy latency without the ciphertext pre-computing and caching optimization (§3.3.5). Bold numbers show where pre-computing and caching ciphertexts helps. The "Overall" row is the average latency over the mix of TPC-C queries. "Update set" is an `UPDATE` where the fields are set to a constant, and "Update inc" is an `UPDATE` where some fields are incremented.

Scheme	Encrypt	Decrypt	Special operation
Blowfish (1 int.)	0.0001 ms	0.0001 ms	—
AES-CBC (1 KB)	0.008 ms	0.007 ms	—
AES-CMC (1 KB)	0.016 ms	0.015 ms	—
OPE (1 int.)	9.0 ms	9.0 ms	Compare: 0 ms
SEARCH (1 word)	0.01 ms	0.004 ms	Match: 0.001 ms
HOM (1 int.)	9.7 ms	0.7 ms	Add: 0.005 ms
JOIN-ADJ (1 int.)	0.52 ms	—	Adjust: 0.56 ms

Figure 3-9: Microbenchmarks of cryptographic schemes, per unit of data encrypted (one 32-bit integer, 1 KB, or one 15-byte word of text), measured by taking the average time over many iterations.

plaintext data. It is pleasantly surprising that the higher security of CryptDB over the strawman also brings better performance.

To understand the latency introduced by CryptDB's proxy, we measure the server and proxy processing times for the same types of SQL queries as above. Figure 3-8 shows the results. We can see that there is an overall server latency increase of 20% with CryptDB, which we consider modest. The proxy adds an average of 0.60 ms to a query; of that time, 24% is spent in MySQL proxy, 23% is spent in encryption and decryption, and the remaining 53% is spent parsing and processing queries. The cryptographic overhead is relatively small because most of our encryption schemes are efficient; Figure 3-9 shows their performance. OPE and HOM are the slowest, but the ciphertext pre-computing and caching optimization (§3.3.5) masks the high latency of queries requiring OPE and HOM. Proxy$\star$ in Figure 3-8 shows the latency without these optimizations, which is significantly higher for the corresponding query types. `SELECT` queries that involve a `SUM` use HOM but do not benefit from this optimization, because the proxy performs decryption, rather than encryption.

In all TPC-C experiments, the proxy used less than 20 MB of memory. Caching ciphertexts for the $30,000$ most common values for OPE accounts for about 3 MB, and pre-computing ciphertexts and randomness for 30,000 values at HOM required 10 MB.

Storage. CryptDB increases the amount of the data stored in the DBMS, because it stores multiple onions for the same field, and because ciphertexts are larger than plaintexts for some encryption schemes. For TPC-C, CryptDB increased the database size by $3.76\times$, mostly due to cryptographic expansion of integer fields encrypted with HOM (which expand from 32 bits to 2048 bits); strings and binary data remains roughly the same size.

Adjustable encryption. Adjustable query-based encryption involves decrypting columns to lower-security onion levels. Fortunately, decryption for the more-secure onion layers, such as RND, is fast, and needs to be performed only once per column for the lifetime of the system.[1] Removing a layer of RND requires AES decryption, which our experimental machine can perform at ~200 MB/s per core. Thus, removing an onion layer is bottlenecked by the speed at which the DBMS server can copy a column from disk for disk-bound databases.

□ 3.7 Security

In this section, we present formally CryptDB's security guarantees.

◇ 3.7.1 Main Theorems

Recall that data owners can mark certain columns as "sensitive". For ease of exposition, we assume that every column not marked is marked with "best-effort encryption". In our analysis, we assume that the order-preserving encryption scheme used is the scheme we designed, mOPE [PLZ13], because it is the only such scheme that does not reveal to the server more than order.

We present our theorems here and later provide formal definitions and proofs.

"Sensitive". CryptDB provides strong security guarantees for a column marked as sensitive: semantic security [Gol04] or a similar guarantee. Such security means that the encryption leaks nothing about the plaintext values to *any polynomial-time adversary* (other than their lengths), even when the adversary *has any side information* about the data. As mentioned, CryptDB can still compute on such sensitive data: selection, insertion, deletion, update, summation, and in some cases, equality checks.

Theorem 1 (Main). *CryptDB provides (distinct-) semantic security (Def. 3) for every column marked as "sensitive", under standard cryptographic assumptions.*

Def. 3 defines (distinct-)semantic security, Def. 10 defines what it means to provide a security guarantee for a column, and the proof of this theorem is in Sec. 3.7.5.

[1] Unless the administrator periodically re-encrypts data/columns.

"Best-effort encryption". For columns marked as "best-effort", CryptDB chooses the most secure encryption scheme from the ones it has available while still supporting the queries issued by the application. Intuitively, CryptDB aims to gives the DB server the *least information* enabling the server to execute the desired queries using a practical model of computation. The practical model requires that processing on encrypted data happens in the same way as on unencrypted data; for example, database indexes (which are crucial for performance) should work as before: this implies the server should be able to perform equality checks on the column indexed. This model makes CryptDB orders of magnitude more efficient than theoretical approaches.

Theorem 2. *Consider any database schema S (Def. 7). CryptDB achieves least-knowledge security (Def. 13) for every column in S (Def. 5) that is marked as "best-effort".*

The proof is in Sec. 3.7.5.

Denote by a *private* column, a column that is annotated with "sensitive" or "best-effort".

CryptDB never decrypts data to plaintext and even the weakest encryption scheme used by CryptDB still provides significant security when the adversary does not have side information about the data in the database:

Corollary 1 (of Th. 1, 2 – Informal: see formal statement in Cor. 1). *The probability that any adversary identifies any one data item in any private column with randomly-chosen values is less than $\frac{n}{D-n} + \text{negl}(\kappa)$, where n is the number of values in the column and D is the size of the domain of values.*

The proof of this Corollary is in Sec. 3.7.5.

For example, if values are 64-bit long and there are a billion data items in the database, the chance that an adversary guesses any given data item is less than $1/2^{33}$, a small number.

We now present two useful facts on columns annotated with "best-effort":

Fact 1. *CryptDB does not decrypt to plaintext at the server any private column.*

Fact 2. *For any query set Q, for any private column c in any schema S, if no query in Q performs comparison or search operations on column c, CryptDB provides semantic security [Gol04] for column c.*

◇ 3.7.2 Preliminaries

In the rest of this document, we assume the reader has basic cryptographic knowledge. We introduce the notation we use.

Let κ denote the security parameter throughout this paper. For a distribution $\mathcal{D}$, we say $x \leftarrow \mathcal{D}$ when x is sampled from the distribution $\mathcal{D}$. If S is a finite set, by $x \leftarrow S$ we mean x is sampled from the uniform distribution over the set S.

We use $p(\cdot)$ to denote that p is a function that takes one input. Similarly, $p(\cdot, \cdot)$ denotes a function p that takes two inputs.

We say that a function f is negligible in an input parameter κ, if for all $d > 0$, there exists K such that for all $\kappa > K$, $f(\kappa) < k^{-d}$. For brevity, we write: for all sufficiently large κ, $f(\kappa) = \text{negl}(\kappa)$. We say that a function f is polynomial in an input parameter κ, if there exists a polynomial p such that for all κ, $f(\kappa) \leq p(\kappa)$. We write $f(\kappa) = \text{poly}(\kappa)$.

Let $[n]$ denote the set $\{1, \ldots, n\}$ for $n \in \mathbb{N}^*$. When saying that a Turing machine A is p.p.t. we mean that A is a non-uniform probabilistic polynomial-time machine.

Two ensembles, $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$, are said to be *computationally indistinguishable* (and denoted $\{X_\kappa\}_{\kappa \in \mathbb{N}} \overset{c}{\approx} \{Y_\kappa\}_{\kappa \in \mathbb{N}}$) if for every probabilistic polynomial-time algorithm D,

$$|\Pr[D(X_\kappa, 1^\kappa) = 1] - \Pr[D(Y_\kappa, 1^\kappa) = 1]| = \mathrm{negl}(\kappa).$$

We use the notation $\mathsf{Adv}^{O(\cdot)}$ to mean that some algorithm Adv runs with oracle access to O; the argument "$\cdot$" to O indicates that, for each request, Adv provides one input to oracle O.

◇ 3.7.3 Cryptographic notions

Let us recall the security definition of a pseudorandom function ensemble, adapted from [KL07].

Definition 1 (Pseudorandom Function Ensembles). *Let* $\mathsf{PRF} : \{0,1\}^* \times \{0,1\}^* \leftarrow \{0,1\}^*$ *be an efficient, length-preserving, keyed function. We say* PRF *is a pseudorandom function if for all* PPT *distinguishers* D*, for every sufficiently large* n*,*

$$Pr[k \leftarrow \{0,1\}^n, D^{F(k,\cdot)}(1^n) = 1] - Pr[\textit{draw a random function } H_n; D^{H_n(\cdot)}(1^n) = 1] = \mathrm{negl}(n),$$

where $H_n : \{0,1\}^n \to \{0,1\}^n$ *maps each input to a random value.*

We refer the reader to [Gol04] for definitions of semantic security for both private- and public-key.

◇ 3.7.4 Formal definitions and statements

Security definitions

We define the notion of distinct-semantic security. Intuitively, this notion is similar to semantic security as long as one only encrypts distinct values. We refer the reader to [Gol04] for definitions of semantic security for both private- and public-key.

Definition 2 ($\mathsf{DSemExp}$ security experiment). *We now describe the distinct-semantic security experiment* $\mathsf{DSemExp}^{\mathsf{Adv}}(\kappa)$*, where* κ *is the security parameter and* Adv *an adversary.*

1. *Let* $k \leftarrow \mathsf{KeyGen}(1^\kappa)$
2. $(m_0, m_1) \leftarrow \mathsf{Adv}^{\mathsf{enc}_k(\cdot)}(1^\kappa)$ *with* $|m_0| = |m_1|$.
3. *Let* $b \leftarrow \{0,1\}$ *and the ciphertext* $\mathsf{ct} \leftarrow \mathsf{enc}_k(m_b)$.
4. $b' \leftarrow \mathsf{Adv}^{\mathsf{enc}_k(\cdot)}(\mathsf{ct})$.
5. *Output* 1 *if* $b' = b$ *and* Adv *never requested* m_0 *or* m_1 *to* $\mathsf{enc}_k(\cdot)$ *oracle, else output* 0.

Definition 3 (Distinct semantic security). *Consider the* $\mathsf{DSemExp}$ *security experiment in Def. 2. A private-key encryption scheme (*KeyGen, enc, Dec*) is distinct-semantically secure if for all* PPT Adv*, for all sufficiently large security parameters* κ*,*

$$\Pr[\mathsf{DSemExp}^{\mathsf{Adv}}(1^\kappa) = 1] \leq 1/2 + \mathrm{negl}(\kappa),$$

where the probability is taken over the random coins in the experiment and of Adv.

System model

To be able to prove statements about CryptDB's security, we need a formal model of the system. Since systems are complex, it is customary to define a simplified system model that captures the core functionality of the system. The expectation is that the rest of the operations in the system map closely to one of the operations in the model. Providing a model for the real system (namely, capturing the hundreds of operators in SQL) in this thesis would be infeasible and not even instructive/meaningful to the reader.

We consider a database system that only allows queries `CREATE`, `SELECT`, `INSERT`, `UPDATE`, `DELETE`. The only operations and filters allowed are equality and order comparison as well as summation. We only consider queries to one database, and the extension to more databases is straightforward.

The database is formed of tables which have columns.

Definition 4 (Annotation). *An annotation is one of the following: "sensitive", "best-effort encryption", and "unencrypted".*

Definition 5 (Column information). *A column information consists of a column name (a string), a domain $\mathcal{D}$ (for the values that can be in the column), and an annotation.*

Definition 6 (Table information). *A table information consists of a table name (a string), a positive integer n, and a list of n column information-s:* $\mathsf{colinfo}_1, \ldots, \mathsf{colinfo}_n$.

Definition 7 (Schema). *A schema is a list of table information.*

Definition 8 (Queries). *Only the following types of queries are allowed:*

- "`CREATE table` ($\mathsf{colinfo}_1$, ..., $\mathsf{colinfo}_n$)" *for some positive integer n, where* $\mathsf{colinfo}_i$ *is a column information and* `table` *is a string.*
- "`INSERT INTO table VALUES` $(v_1, \ldots, v_n)$", *where v_i is in the domain $\mathcal{D}$ of* $\mathsf{colinfo}_i$.
- "`UPDATE table SET name = const WHERE [filters(table)]`", *where* `name` *is the name of a column in* `table`, `const` *is a constant, and* `[filters]` *is defined below.*
- "`DELETE FROM table WHERE [filters(table)]`".
- "`SELECT A, sum(B) FROM table1, table2 WHERE name1 = name2 AND/OR [filters(table1)] AND/OR [filters(table2)]`", *where* `A`, `B`, *and* `name1` *are columns in* `table1` *and* `name2` *is a column in* `table2`. *This query makes uses of projection, sum, join, and various filters below.*
- `filters(table)` *can be a series of any of the following terms connected with* `AND` *or* `OR`:
  - `name` $=$ `const`
  - `name` $\geq$ `const`
  - `name` $\leq$ `const`

operation	onion level
update, delete, insert	RND
equality between a value from column c and a constant	DET
order between a value from column c and a constant	OPE
join by equality on column a and b	JOIN
project on column c	RND
sum on column c	RND
no operation	RND

Table 3.1: Operations and the onion levels they require.

with `name` *being the name of a column in* `table` *and* `const` *a constant in the domain of* `name`.

Definition 9 (Column of a constant in a query q). *The column of a constant in a query q is the column to which the constant is being assigned if q is an* `UPDATE` *query in the* `SET` *clause or if q is an* `INSERT` *query, or it is the column to which it is compared in a filter.*

For example, in the `UPDATE` query in Def. 8, the column of the constant `const` is `name`.

Definition 10 (Providing security level $\mathcal{L}$ for a column c - Informal). *CryptDB provides a security level $\mathcal{L}$ for a column c if all the onions of the column are encrypted with encryption schemes that achieve $\mathcal{L}$ with a key stored at the client, and all the constants for that column (Def. 9) in any query are encrypted with the same scheme and key.*

Definition 11. *There are four types of security levels RND, DET, OPE, and JOIN.*

The HOM onion level in CryptDB is considered the same as RND in this document because it also has semantic security. In this document, we do not model SEARCH, although one can view it as no less secure than DET.

If a column is encrypted with an onion level, it has a certain security guarantee:

onion level	security guarantee
RND	semantic security, defined in [Gol04]
DET	PRF, defined in Def. 1
OPE	IND-OCPA, defined in [PLZ13]
JOIN	as defined in [PZ12]

The least-knowledge security definition says that no column will have an onion level that is not needed by some query (we only consider queries that are authorized by the user to run on the database).

Definition 12 (Exposed onion levels for column c.). *The exposed onion levels for column c at a given time is the set of onion levels such that each onion level is the topmost level on some onion of column c.*

Definition 13 (Least-knowledge security for column c from schema S.). *Let Q be any set of queries as in Def. 8. Consider running CryptDB only on queries from Q.*

CryptDB achieves least-knowledge security for column c, if : for each onion level exposed for column c at any point during the execution, there is an operation in Q that requires that onion level based on Table 3.1.

Formal Corollary 1

We now present the formal corollary of Cor. 1. The weakest encryption scheme in CryptDB is OPE [PLZ13] (all other encryption schemes provide at least the security of OPE). Let OPE.Keygen be the keygeneration algorithm of the OPE scheme of [PLZ13] and let OPE.Enc be the encryption algorithm. Recall that OPE.Enc is interactive so we denote by $\mathsf{OPE.Enc}^{\mathsf{Adv}}$, the encryption algorithm where Adv is the server as defined in [PLZ13].

Corollary 1 (of Th. 1, 2). *Consider any schema S with at least one column and any column c in the schema. Let $\mathcal{D}$ be the domain of each item in column c. Let OPE be a scheme achieving IND-OCPA as defined in [PLZ13]. Then, for any stateful* PPT *adversary* Adv, *we have:*

$$\Pr\left[\begin{array}{l} k \leftarrow \mathsf{OPE.Keygen}(1^\kappa); \\ n \leftarrow \mathsf{Adv}(1^\kappa); \\ x_1, \ldots, x_n \leftarrow \mathcal{D}, \textit{ uniformly and distinct}; \\ \{c_i \leftarrow \mathsf{OPE.Enc}^{\mathsf{Adv}}(k, x_i)\}_{i=1}^n; \\ \mathsf{Adv}(c_1, \ldots, c_n) = (i, x_i') \\ \textit{s.t. } x_i = x_i' \textit{ and } 2n < |\mathcal{D}| \end{array}\right] < \frac{n}{|\mathcal{D}| - n} + \operatorname{negl}(\kappa).$$

For simplicity, the values $x_1, \ldots, x_n$ were chosen to be distinct, but one can easily prove a similar statement for when the values are sampled with replacement.

◇ 3.7.5 Proofs

Proof of Theorem 1

Proof. CryptDB encrypts data in columns marked as sensitive with the following encryption schemes:

- RND which is AES in UFE mode, an IND-CCA2 secure scheme; it thus provides an even stronger security definition than semantic security.
- HOM which is Paillier encryption, proven to be semantically secure in [Pai99].
- DET is a pseudorandom function (PRF) and Lemma 1 proves that PRFs achieve distinct-semantic security.

Lemma 1. *Any pseudorandom function (PRF), Def. 1, is distinct-semantically secure.*

Proof. For contradiction, assume there is a PPT adversary Adv that breaks distinct semantic-security of a PRF; let us show how to construct a PPT reduction B that breaks the security of the PRF. Since Adv breaks distinct-semantic security, we have

$$\Pr[\mathsf{DSemExp}^{\mathsf{Adv}}(1^\kappa) = 1] > 1/2 + 1/\alpha(\kappa), \tag{3.3}$$

for some polynomial α.

B receives access to an oracle $\mathcal{O}$ which can either be the PRF function on a key k or a random oracle. To distinguish which is the case, B proceeds as follows. B starts Adv.

- For every request x that Adv makes to the $\mathsf{enc}_k(\cdot)$ oracle, B makes a request for x to its oracle $\mathcal{O}$ and returns the result to Adv.

- Adv replies to B with m_0 and m_1 and B picks a bit at random b and returns to Adv the value $\mathcal{O}(m_b)$.
- For all further requests of Adv to $\mathsf{enc}_k(\cdot)$, B proceeds as before.
- Adv returns its guess b' to B. If $b = b'$, B outputs 1 (meaning "PRF"), else B outputs 0 (meaning "random function").

Let win_B be the chance that B outputs the correct result. If $\mathcal{O}$ was a random oracle, Adv has a chance of exactly half of outputting a correct guess because the encryptions of m_0 and m_1 are identically distributed and Adv was not allowed to ask enc_k queries to m_0 or m_1. If $\mathcal{O}$ was the PRF for some key k, B simulated the input distribution to Adv correctly meaning that Adv has the advantage of guessing b as in Eq. 3.3. Therefore,

$$\begin{aligned}\Pr[\mathsf{win}_B] &= 1/2\Pr[\mathsf{win}_B|\mathcal{O} = \mathsf{enc}(k, H_k)] + 1/2\Pr[\mathsf{win}_B|\mathcal{O} = \mathsf{enc}(k, \cdot)] \\ &= 1/2 \cdot 1/2 + 1/2(1/2 + 1/\alpha(\kappa)) \\ &= 1/2 + 1/2\alpha(\kappa).\end{aligned}$$

Hence, B also has a nonneligible advantage of breaking security of the PRF, which concludes our proof.

□

□

Proof of Corollary 1

Proof. For contradiction, assume there is a PPT adversary Adv that breaks the corollary and let us construct a PPT reduction B that breaks the IND-OCPA security of the OPE scheme. To break IND-OCPA, B needs to output two sequences of values $\bar{x} = x_1, \ldots, x_n$ and $\bar{y} = y_1, \ldots, y_n$ such that they have the same order relation (as formalized in [PLZ13]). Then, B receives $\bar{x}$ or $\bar{y}$ encrypted, denoted $c_1, \ldots, c_n$, and must decide which of $\bar{x}$ or $\bar{y}$ were encrypted. B proceeds as follows:

1. B starts Adv and receives n. B chooses a sequences $\bar{x} = x_1, \ldots, x_n$ from $\mathcal{D}$ by sampling uniformly without replacement. Then B samples a sequence $\bar{y} = y_1, \ldots, y_n$ uniformly from all sequences of n distinct elements in $\mathcal{D}$ with the same order as $\bar{x}$ such that no element of $\bar{y}$ appears in $\bar{x}$. (This can be done by sampling n distinct elements from $\mathcal{D}$ at random that are different from the elements of $\bar{x}$ then sorting $\bar{y}$ to have the same order relation as $\bar{x}$. Since $|\mathcal{D}| > 2n$ such sampling is well defined). B outputs $\bar{x}$ and $\bar{y}$.
2. B receives $c_1, \ldots, c_n$ and sends it to Adv which replies with i, v.
3. If $x_i = v$, B outputs "it is $\bar{x}$"; if $y_i = v$, B outputs "it is $\bar{y}$", otherwise B outputs a random guess.

Even though $\bar{x}$ and $\bar{y}$ are restricted to different values, Adv only receives the ciphertext corresponding to one of them, and any one of $\bar{x}$ or $\bar{y}$ in isolation is distributed as Adv expects. Therefore, B simulates the inputs to Adv perfectly.

Let's compute the winning probability of B. For this, we need to introduce notation and quantify some intermediary probabilities.

Let us denote by wrong_κ the chance that Adv outputs y_i instead of x_i when $\bar{c}$ encrypts $\bar{x}$ or that Adv outputs x_i instead of y_i when $\bar{c}$ encrypts $\bar{y}$. Consider the case when $\bar{c}$ encrypts $\bar{x}$. Even if Adv knows the entire vector $\bar{x}$, Adv gets no information about $\bar{y}$ information-theoretically. Moreover, $\bar{y}$ consists of random values. The probability of wrong_κ is at least the chance that the value v equals no value in $\bar{y}$, which is $1 - \binom{D-n-1}{n} / \binom{D-n}{n}$; hence,

$$\Pr[\mathsf{wrong}_\kappa] \leq \frac{n}{D-n} \tag{3.4}$$

Let guess_κ be the event that Adv guesses correctly v. By the hypothesis of the contradiction, we have

$$\Pr[\mathsf{guess}_\kappa] \geq \frac{n}{D-n} + \frac{1}{\alpha(\kappa)}, \tag{3.5}$$

for some polynomial α.

We are now ready to compute the winning probability of$B$; denote by win_κ the event that B wins. Let us assume that $\bar{c}$ encrypts $\bar{x}$ (the case when $\bar{c}$ encrypts $\bar{y}$ results in the same winning probability). We have

$$\begin{aligned}
\Pr[\mathsf{win}_\kappa] &= \Pr[\mathsf{win}_\kappa | x_i = v] \Pr[x_i = v] + \Pr[\mathsf{win}_\kappa | y_i = v] \Pr[y_i = v] \\
&+ \Pr[\mathsf{win}_\kappa | v \notin \{x_i, y_i\}] \Pr[v \notin \{x_i, y_i\}] \\
&= 1 \cdot \Pr[x_i = v] + 0 \cdot \Pr[y_i = v] + 1/2 \cdot \Pr[v \notin \{x_i, y_i\}] \\
&= \Pr[\mathsf{guess}_\kappa] + 1/2(1 - \Pr[\mathsf{guess}_\kappa] - \Pr[\mathsf{wrong}_\kappa]) \\
&= 1/2 + 1/2(\Pr[\mathsf{guess}_\kappa] - \Pr[\mathsf{wrong}_\kappa]) \\
&\geq 1/2 + 1/2\alpha(\kappa).
\end{aligned}$$

□

Proof of Theorem 2

Proof. We prove the theorem by induction on the queries executed by CryptDB.

The base case of the induction is when no queries were executed. In that case, CryptDB's initial onion state for every column has only RND exposed. We can see that this satisfies the base case because no operation requires RND. Let Col be a column marked with "best-effort". At inductive step i, we assume CryptDB satisfied Theorem 2 for the first i queries for column Col, and we show that, after the $i+1$-th query, CryptDB still satisfies the theorem statement.

Let q be the $i+1$-th query with $q \in Q$. The only time that CryptDB changes the onion level exposed for column c is during an onion adjustment. Onion adjustments only happen for three operations:

1. *There is an operator of the form column* Col $=$ const *or* const $=$ Col. This operation triggers an adjustment only if the current onion level is RND on the Equality onion. In this case, the resulting onion level will be DET, which is requires by the "$=$" operation.

2. *There is an operator of the form column* $\mathrm{Col} \geq \mathsf{const}$ *or* $\mathsf{const} \geq \mathrm{Col}$. This operation triggers an adjustment only if the current onion level is RND on the Order onion. In this case, the resulting onion level will be OPE, which is required by the "$>$" operation.

3. *There is an operator of the form column* $\mathrm{Col} = \mathrm{Col}_2$ *for some other column* Col_2. This operation triggers an adjustment only if the current onion level is RND or DET on the Equality onion. In this case, the resulting onion level will be JOIN, which is required by this operation.

Since the hypothesis was correct up to step i and step $i+1$ introduced only exposed onion levels that are required by a query in Q, the hypothesis is also correct for step $i+1$, thus concluding our proof.

□

□ 3.8 Conclusion

In this chapter, we presented CryptDB, a system that provides a practical and strong level of confidentiality in the face of a significant threat: curious DBAs or hackers gaining access to the database server. CryptDB meets its goals using two ideas: running queries efficiently over encrypted data using a novel SQL-aware encryption strategy, and dynamically adjusting the encryption level using onions of encryption to minimize the information revealed to the untrusted DBMS server.

Our evaluation on a large trace of 126 million SQL queries from a production MySQL server shows that CryptDB can support operations over encrypted data for 99.5% of the 128,840 columns seen in the trace. The throughput penalty of CryptDB is modest, resulting in a reduction of 26% on an industry-standard benchmark as compared to unmodified MySQL. Our security analysis shows that CryptDB protects most sensitive fields with highly secure encryption schemes for six applications. CryptDB makes no changes to existing SQL-backed applications or to the database server, making it easier to adopt. The source code for our implementation is available for download at http://css.csail.mit.edu/cryptdb/.

CHAPTER 4

Securing web applications with Mylar

This chapter presents Mylar, a new platform for building web applications that stores only encrypted data on the server.

□ 4.1 Motivation

Using a web application for confidential data requires the user to trust the server to protect the data from unauthorized disclosures. This trust is often misplaced, however, because there are many ways in which confidential data could leak from a server. For example, attackers could exploit a vulnerability in the server software to break in [Tud13], a curious administrator could peek at the data on the server [Che10, Bor13], or the server operator may be compelled to disclose data by law [Goo13]. Is it possible to build web applications that protect data confidentiality against attackers with *full access* to servers?

A promising approach is to give each user their own encryption key, encrypt a user's data with that user's key in the web browser, and store only encrypted data on the server. This model ensures that an adversary would not be able to read any confidential information on the server, because they would lack the necessary decryption keys. In fact, this model has been already adopted by some privacy-conscious web applications [The13, Meg13].

Unfortunately, this approach suffers from three significant security, functionality, and efficiency shortcomings. First, a compromised server could provide malicious client-side code to the browser and extract the user's key and data. Ensuring that the server did not tamper with the application code is difficult because a web application consists of many files, such as HTML pages, Javascript code, and CSS style sheets, and the HTML pages are often dynamically generated.

Second, this approach does not provide data sharing between users, a crucial function of web applications. To address this problem, one might consider encrypting shared documents with separate keys, and distributing each key to all users sharing a document via the server. However, distributing keys via the server is challenging because a compromised server can supply arbitrary keys to users, and thus trick a user into using incorrect keys.

Third, this approach requires that all of the application logic runs in a user's web browser because it can decrypt the user's encrypted data. But this is often impractical: for instance, doing a keyword search would require downloading all the documents to the browser.

This chapter presents Mylar, a new platform for building web applications that stores only encrypted data on the server. Mylar makes it practical for many classes of applications to protect confidential data from compromised servers. It leverages the recent shift in web application frameworks towards implementing logic in client-side Javascript code, and sending data, rather than HTML, over the network [Met13]; such a framework provides a clean foundation for security. Mylar addresses the challenges mentioned above with a combination of systems techniques and novel cryptographic primitives, as follows.

Data sharing. To enable sharing, each sensitive data item is encrypted with a key available to users who share the item. To prevent the server from cheating during key distribution, Mylar provides a mechanism for establishing the correctness of keys obtained from the server: Mylar forms certificate paths to attest to public keys, and allows the application to specify what certificate paths can be trusted in each use context. In combination with a user interface that displays the appropriate certificate components to the user, this technique ensures that even a compromised server cannot trick the application into using the wrong key.

Computing over encrypted data. Keyword search is a common operation in web applications, but it is often impractical to run on the client because it would require downloading large amounts of data to the user's machine. While there exist practical cryptographic schemes for keyword search, they require that data be encrypted with a single key. This restriction makes it difficult to apply these schemes to web applications that have many users and hence have data encrypted with many different keys.

Mylar provides the first cryptographic scheme that can perform keyword search efficiently over data encrypted with *different* keys. The client provides an encrypted word to the server and the server can return all documents that contain this word, without learning the word or the contents of the documents.

Verifying application code. With Mylar, code running in a web browser has access to the user's decrypted data and keys, but the code itself comes from the untrusted server. To ensure that this code has not been tampered with, Mylar checks that the code is properly signed by the web site owner. This checking is possible because application code and data are separate in Mylar, so the code is static. Mylar uses two origins to simplify code verification for a web application. The primary origin hosts only the top-level HTML page of the application, whose signature is verified using a public key found in the server's X.509 certificate. All other files come from a secondary origin, so that if they are loaded as a top-level page, they do not have access to the primary origin. Mylar verifies the hash of these files against an expected hash contained in the top-level page.

To evaluate Mylar's design, we built a prototype on top of the Meteor web application framework [Met13]. We ported 6 applications to protect confidential data using Mylar: a medical application for endometriosis patients, a web site for managing homework and grades, a chat application called kChat, a forum, a calendar, and a photo sharing application. The endometriosis application is used to collect data from patients with that medical condition, and was designed under the aegis

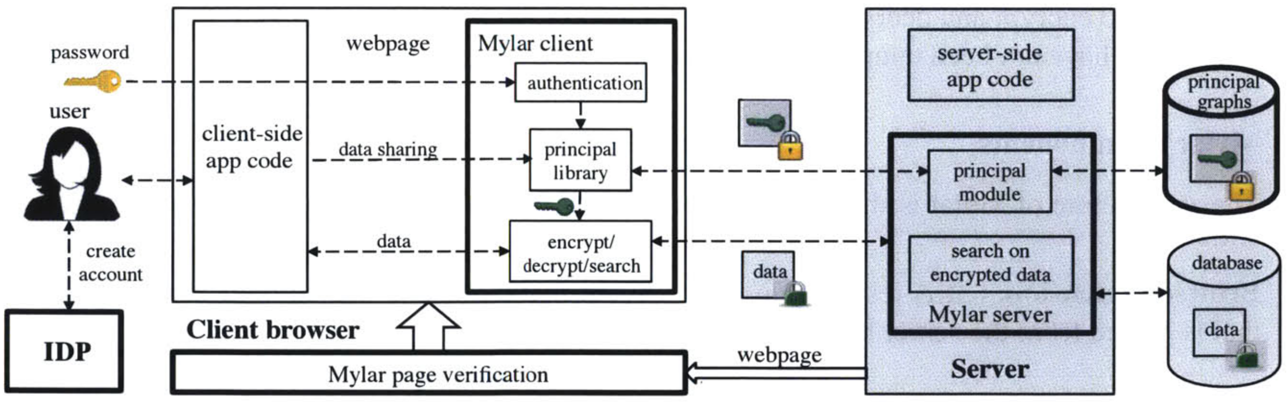

Figure 4-1: System overview. Shaded components have access only to encrypted data. Thick borders indicate components introduced by Mylar.

of the MIT Center for Gynepathology Research by surgeons at the Newton-Wellesley hospital (affiliated with the Harvard Medical School) in collaboration with biological engineers at MIT; the Mylar-secured version is currently being tested by patients and is undergoing IRB approval before deployment.

Our results show that Mylar requires little developer effort: we had to modify an average of just 36 lines of code per application. We also evaluated the performance of Mylar on three of the applications above. For example, for kChat, our results show that Mylar incurs modest overheads: a 17% throughput reduction and a 50 msec latency increase for the most common operation (sending a message). These results suggest that Mylar is a good fit for multi-user web applications with data sharing.

□ 4.2 Mylar's architecture

There are three different parties in Mylar: the users, the web site owner, and the server operator. Mylar's goal is to help the site owner protect the confidential data of users in the face of a malicious or compromised server operator.

◇ 4.2.1 System overview

Mylar embraces the trend towards client-side web applications; Mylar's design is suitable for platforms that:

1. Enable client-side computation on data received from the server.

2. Allow the client to intercept data going to the server and data coming from the server.

3. Separate application code from data, so that the HTML pages supplied by the server are static.

AJAX web applications with a unified interface for sending data over the network, such as Meteor [Met13], fit this model. Such frameworks provide a clean foundation for security, because they send data separately from the HTML page that presents the data. In contrast, traditional server-side frameworks incorporate dynamic data into the application's HTML page in arbitrary ways, making it difficult to encrypt and decrypt the dynamic data on each page while checking that the fixed parts of the page have not been tampered with [Sta13].

Mylar's components

The architecture of Mylar is shown in Figure 4-1. Mylar consists of the four following components:

Browser extension. It is responsible for verifying that the client-side code of a web application that is loaded from the server has not been tampered with.

Client-side library. It intercepts data sent to and from the server, and encrypts or decrypts that data. Each user has a private-public key pair. The client-side library stores the private key of the user at the server, encrypted with the user's password.[1] When the user logs in, the client-side library fetches and decrypts the user's private key. For shared data, Mylar's client creates separate keys that are also stored at the server in encrypted form.

Server-side library. It performs computation over encrypted data at the server. Specifically, Mylar supports keyword search over encrypted data, because we have found that many applications use keyword search.

Identity provider (IDP). For some applications, Mylar needs a trusted identity provider service (IDP) to verify that a given public key belongs to a particular username. An application needs the IDP if the application has no trusted way of verifying the users who create accounts, and the application allows users to choose whom to share data with. For example, if Alice wants to share a sensitive document with Bob, Mylar's client needs the public key of Bob to encrypt the document. A compromised server could provide the public key of an attacker, so Mylar needs a way to verify the public key. The IDP helps Mylar perform this verification by signing the user's public key and username. An application does not need the IDP if the site owner wants to protect against only passive attacks (§4.2.4), or if the application has a limited sharing pattern for which it can use a static root of trust (see §4.3.2).

An IDP can be shared by many applications, similar to an OpenID provider [Ope13]. The IDP does not store per-application state, and Mylar contacts the IDP only when a user first creates an account in an application; afterwards, the application server stores the certificate from the IDP.

◇ 4.2.2 Mylar for developers

The developer starts with a regular (non-encrypted) web application implemented in Mylar's underlying web platform (Meteor in our prototype). To secure this application with Mylar, a developer uses Mylar's API (Figure 4-2), as we explain in the rest of this chapter. First, the developer uses Mylar's authentication library for user login and account creation. If the application allows a user to choose what other users to share data with, the developer should also specify the URL and public key of a trusted IDP.

[1]The private key can also be stored at a trusted third-party server, to better protect it from offline password guessing attacks and to recover from forgotten passwords without re-generating keys.

Function	Semantics
idp_config(*url, pubkey*)	Declares the *url* and *pubkey* of the IDP and returns the principal corresponding to the IDP.
create_user(*uname, password, auth_princ*)	Creates an account for user *uname* which is certified by principal *auth_princ*.
login(*uname, password*)	Logs in user *uname*.
logout()	Logs out the currently logged-in user.
collection.**encrypted**({*field*: *princ_field*}, ...)	Specify that *field* in *collection* should be encrypted for the principal in *princ_field*.
collection.**auth_set**([*princ_field, fields*], ...)	Authenticate the set of *fields* with principal in *princ_field*.
collection.**searchable**(*field*)	Mark *field* in *collection* as searchable.
collection.**search**(*word, field, princ, filter, proj*)	Search for *word* in *field* of *collection*, filter results by *filter* and project only the fields in *proj* from the results. Use *princ*'s key to generate the search token.
princ_create(*name, creator_princ*)	Create principal named *name*, sign the principal with *creator_princ*, and give *creator_princ* access to it.
princ_create_static(*name, password*)	Create a static principal called *name*, hardcode it in the application, and wrap its secret keys with *password*.
princ_static(*name, password*)	Return the static principal *name*; if a correct password is specified, also load the secret keys for this principal.
princ_current()	Return the principal of currently logged in user.
princ_lookup($name_1, \ldots, name_k$, *root*)	Look up principal named $name_1$ as certified by a chain of principals named $name_i$ rooted in *root* (e.g., the IDP).
granter.**add_access**(*grantee*)	Give the *grantee* principal access to the *granter* principal.
grantee.**allow_search**(*granter*)	Allow matching keywords from *grantee* on *granter*'s data.

Figure 4-2: Mylar API for application developers split in three sections: authentication, encryption/integrity annotations, and access control. All of the functions except `princ_create_static` and `searchable` run in the client browser. This API assumes a MongoDB storage model where data is organized as collections of documents, and each document consists of fieldname-and-value pairs. Mylar also preserves the generic functionality for unencrypted data of the underlying web framework.

Second, the developer specifies which data in the application should be encrypted, and who should have access to it. Mylar uses principals for access control; a principal corresponds to a public/private key pair, and represents an application-level access control entity, such as a user, a group, or a shared document. In our prototype, all data is stored in MongoDB collections, and the developer annotates each collection with the set of fields that contain confidential data and the name of the principal that should have access to that data (i.e., whose key should be used).

Third, the developer specifies which principals in the application have access to which other principals. For example, if Alice wants to invite Bob to a confidential chat, the application must invoke the Mylar client to grant Bob's principal access to the chat room principal.

Fourth, the developer changes their server-side code to invoke the Mylar server-side library when performing keyword search. Our prototype's client-side library provides functions for common operations such as keyword search over a specific field in a MongoDB collection.

Finally, as part of installing the web application, the site owner generates a public/private key pair, and signs the application's files with the private key using Mylar's bundling tool. The web application must be hosted using `https`, and the site owner's public key must be stored in the web server's X.509 certificate. This ensures that even if the server is compromised, Mylar's browser extension will know the site owner's public key, and will refuse to load client-side code if it has been tampered with.

◇ 4.2.3 Mylar for users

To obtain the full security guarantees of Mylar, a user must install the Mylar browser extension, which detects tampered code. However, if a site owner wants to protect against only passive attacks (§4.2.4), users don't have to install the extension and their browsing experience is entirely unchanged.

◇ 4.2.4 Threat model

Threats. Both the application and the database servers can be *fully* controlled by an adversary: the adversary may obtain all data from the server, cause the server to send arbitrary responses to web browsers, etc. This model subsumes a wide range of real-world security problems, from bugs in server software to insider attacks.

Mylar also allows some user machines to be controlled by the adversary, and to collude with the server. This may be either because the adversary is a user of the application, or because the adversary broke into a user's machine.

We call this adversary *active*, in contrast to a *passive* adversary that eavesdrops on all information at the server, but does not make any changes, so that the server responds to all client requests as if it were not compromised.

Guarantees. Mylar protects a data item's confidentiality in the face of arbitrary server compromises, as long as none of the users with access to that data item use a compromised machine. Mylar does not hide data access patterns, or communication and timing patterns in an application. Mylar provides data authentication guarantees, but does not guarantee the freshness or correctness of results from the computation at the server.

Assumptions. To provide the above guarantees, Mylar makes the following assumptions. Mylar assumes that the web application as written by the developer will not send user data or keys to untrustworthy recipients, and cannot be tricked into doing so by exploiting bugs (e.g., cross-site scripting). Our prototype of Mylar is built on top of Meteor, a framework that helps programmers avoid many common classes of bugs in practice.

Mylar also assumes that the IDP correctly verifies each user's identity (e.g., email address) when signing certificates. To simplify the job of building a trustworthy IDP, Mylar does not store any application state at the IDP, contacts the IDP only when a user first registers, and allows the IDP to be shared across applications.

Finally, Mylar assumes that the user checks the web browser's security indicator (e.g., the `https` shield icon) and the URL of the web application they are using, before entering any sensitive data. This assumption is identical to what users must already do to safely interact with a *trusted* server. If the user falls for a phishing attack, neither Mylar nor a trusted server can prevent the user from entering confidential data into the adversary's web application.

◇ 4.2.5 Security overview

At a high level, Mylar achieves its goal as follows. First, it verifies the application code running in the browser (§4.5), so that it is safe to give client-side code access to keys and plaintext data. Then, the client code encrypts the data marked sensitive before sending it to the server. Since users need to share data, Mylar provides a mechanism to securely share and look up keys among users (§4.3). Finally, to perform server-side processing, Mylar introduces a new cryptographic scheme that can perform keyword search over documents encrypted with many different keys, without revealing the content of the encrypted documents or the word being searched for (§4.4).

□ 4.3 Sharing data between users

Many web applications share data between users according to some policy. A simple example is a chat application, where messages are shared between the sender and the recipients. In Mylar's threat model, an application cannot trust the server to enforce the sharing policy, because the server is assumed to be compromised. As a result, the application must encrypt shared data using a key that will be accessible to just the right set of users.

Mylar allows an application to specify its security policy in terms of application-defined principals. In particular, each principal has an application-chosen *name*, a *public key* used to encrypt data for that principal, and a *private key* used to decrypt that principal's data.

In addition to allowing the application to create principals, and to use the principals' keys to encrypt and decrypt data, Mylar provides two critical operations to the application for managing principals:

- Find a principal so that the application can use the corresponding private key to decrypt data. The goal is to ensure that only authorized users can get access to the appropriate private key.

- Find a principal so that the application can use the corresponding public key to encrypt or share data with other users. The goal is to ensure that a malicious server cannot trick Mylar into returning the wrong public key, which could lead the application to share confidential data with the adversary.

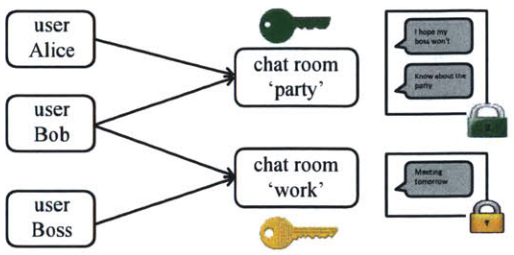

Figure 4-3: Example access graph for a chat application. Rounded rectangles represent principals, and arrows represent access relationships. Alice and Bob share the chat room "party" so they both have access to the principal for this room. Messages in each chat room are encrypted with the key of the room's principal.

Mylar cryptographically enforces the above goals by forming two graphs on top of principals: an *access graph*, which uses key chains to distribute the private keys of shared principals to users, and a *certification graph*, which uses certificate chains to attest to the mapping between a principal name and its public key.

◇ 4.3.1 Access graph

To ensure that only authorized users can access the private key of a principal, Mylar requires the application to express its access control policy in terms of *access* relationships between principals. Namely, if principal A can access principal B's private key, then we say *A has access to B*. The *has access to* relation is transitive: if B in turn has access to C, then A can access C's private key as well. To express the application's policy in the access graph, the application must create appropriate *has access to* relationships between principals. The application can also create intermediate principals to represent, say, groups of users that all should have access to the same private keys.

As an example, consider a chat application where messages in each chat room should be available only to that room's participants. Figure 4-3 shows the access graph for this scenario. Both Alice and Bob have access to the key encrypting the "party" room, but the boss does not.

Key chaining. To enforce the access graph cryptographically, Mylar uses key chaining, as in CryptDB [PRZB11]. When an application asks to add a new *has access to* edge from principal A to principal B, Mylar creates a *wrapped key*: an encryption of B's private keys under the public key of principal A. This ensures that a user with access to A's private key can decrypt the wrapped key and obtain B's private key. For example, in Figure 4-3, the private key of the "party" chat room is encrypted under the public key of Alice, and separately under the public key of Bob as well. The server stores these wrapped keys, which is safe since the keys are encrypted.

In practice, *has access to* relationships are rooted in user principals, so that a user can gain access to all of their data when they initially log in and have just the private key of their own user principal. When Mylar needs to decrypt a particular data item, it first looks up that data item's principal, as specified by the **encrypted** annotation (Figure 4-2). Mylar then searches for a chain of wrapped keys, starting from the principal of the currently logged in user, and leading to the data item's principal.

◇ 4.3.2 Certification graph

Mylar applications must look up public keys of principals when sharing data, for two broad purposes: either to encrypt data with that key, or to give some principal access to that key. In both cases, if a compromised server tricks the client application into using the public key of the adversary, the adversary will gain access to confidential data. For example, in the chat example, suppose Bob wants to send a confidential message to the "work" chat room. If the server supplies the adversary's public key for the chat room principal and the application client uses it, the adversary will be able to decrypt the message. Preventing such attacks is difficult because all of the wrapped keys are stored at the server, and the server may be malicious.

To prevent such attacks, Mylar relies on a certification graph, which allows one principal to vouch for the name and the public key of another principal. The nodes of this graph are principals from the access graph together with some *authority* principals, which are principals providing the root of trust (described in §4.3.3). Applications create certificate chains for principals, rooted in an authority principal. For instance, in the chat example, the application can sign the "chatroom:work" principal with the key of the "user:boss" principal that created the chat room. Using the certification graph, applications can look up the public key of a principal by specifying the name of the principal they are looking for, along with a chain of certifications they expect to find.

Since the server is not trusted, there is no single authority to decide on the public key for a given principal name: in our chat example, both the real boss and a malicious server may have created chat rooms named "work." To prevent such naming ambiguity, one approach is to display the names in a certification chain to the user, similar to how web browsers display the hostname from an X.509 certificate for `https` web sites. As we describe later in §4.7, if the chat application displays the email address of the chat room creator (who signed the chat room principal), in addition to the name of the chat room, the user could distinguish a correct "work" chat room, created by the boss, from an impostor created by an attacker. This requires Mylar applications to unambiguously map human-meaningful names, such as the "work" chat room and the identity of the Boss user, onto principal names, such as "chatroom:work" and "user:boss."

Mylar's certificate chains are similar to X.509; the difference is that X.509 typically has fixed roots of trust and fixed rules for what certificate chains are allowed, whereas Mylar allows the application to specify different roots of trust and acceptable chains for each lookup.

◇ 4.3.3 Principals providing the root of trust

The authority principals can be either the IDP or *static principals*. Static principals are access control entities fixed in the application's logic. For example, the endometriosis medical application has a group called "surgeons" representing the surgeons that have access to all patient data. Similarly, the homework submission application has a group called "staff" representing staff members with access to all student homework submissions and grades. In these applications, static principals can altogether remove the need for an IDP.

A developer can create a static principal by running **princ_create_static**(*name*, *password*) with the help of a command-line tool. This generates fresh keys for a principal, and encrypts the secret keys with *password*, so they can be retrieved only by providing *password* to **princ_static**. The resulting public key and encrypted secret key are hardcoded into the application's source code. This allows the application to refer to the static principal by name without relying on the IDP.

Static principals can also certify other principals. For example, in the endometriosis application, all user accounts are manually created by surgeons. This allows all user principals to be certified by the static "surgeons" principal, avoiding the need for an IDP to do the same.

◇ 4.3.4 User principals

To create an account for a new user, the application must invoke **create_user**, as shown in Figure 4-2. This causes the Mylar client to generate a new principal for the user, encrypt the secret key with the user's password, and store the principal with the encrypted secret key on the server.

To enable the application to later look up this user's public key, in the presence of active adversaries, the principal must be certified. To do this, the application supplies the *auth_princ* argument to **create_user**. This is typically either a static principal or the IDP. For static principals, the certificate is generated directly in the browser that calls **create_user**; the creator must have access to the private key of *auth_princ*. For example, the endometriosis application, where all users are manually created by a surgeon, follows this model. If *auth_princ* is the IDP, the Mylar client interprets *uname* as the user's email address, and contacts the IDP, which verifies the user's email address and signs a certificate containing the user's public key and email address.

Even though multiple applications can share the IDP, a buggy or malicious application will not affect other applications that use the same IDP (unless users share passwords across applications). This property is ensured by never sending passwords or secret keys to the IDP, and explicitly including the application's origin in the certificate generated by the IDP.

◇ 4.3.5 Data integrity

To prevent an attacker from tampering with the data, Mylar provides two ways to authenticate data, as follows.

First, all encrypted data is authenticated with a MAC (message authentication code),[2] which means that clients will detect any tampering with the ciphertext. However, an adversary can still replace the ciphertext of one field in a document with any other ciphertext that was encrypted using the same key.

To protect against such attacks, developers can specify an *authentication set* of fields whose values must be consistent with one other, using the **auth_set** annotation. This annotation guarantees that if a client receives some document, then all fields in each authentication set were consistent at some point, according to the corresponding principal. Mylar enforces authentication sets by computing a MAC over the values of all fields in each set.

For example, in a chat room application, each message has several fields, including the message body and the (client-generated) timestamp. By putting these two fields into an authentication set, the developer ensures that an adversary cannot splice together the body of one message with the timestamp from another message.

Mylar does not guarantee data freshness, or correctness of query results. An adversary can roll back the entire authentication set to an earlier version without detection, but cannot roll back a *subset* of an authentication set.

[2]For efficiency, Mylar uses authenticated encryption, which conceptually computes both the ciphertext and the MAC tag in one pass.

□ 4.4 Computing on encrypted data

The challenge facing Mylar in computing over encrypted data is that web applications often have many users, resulting in data encrypted with many different keys. Existing efficient encryption schemes for computation over encrypted data, such as keyword search, assume that all data is encrypted with a single key [SWP00, KPR12]. Using such a scheme in Mylar would require computation over one key at a time, which is inefficient.

For example, consider a user with access to N documents, where each document is encrypted with a different key (since it can be shared with a different set of users). Searching for a keyword in all of these documents would require the user to generate N distinct cryptographic search tokens, and to send all of them to the server. Even for modest values of N, such as 1000, this can result in noticeable computation and network costs for the user's machine. Moreover, if the N keys are not readily available in the client browser, fetching these keys may bring further overhead.

To address this limitation, Mylar introduces a *multi-key search* scheme, as described in the rest of this section.

◇ 4.4.1 Multi-key search

Mylar's multi-key search scheme provides a simple abstraction. If a user wants to search for a word in a set of documents on a server, each encrypted with a different key, the user's machine needs to provide only a single search token for that word to the server. The server, in turn, returns each encrypted document that contains the user's keyword, as long as *the user has access* to that document's key.

The intuition for our scheme is as follows. Say that the documents that a user has access to are encrypted under keys $k_1, \ldots, k_n$ and the user's own key is uk. The user's machine computes a search token for a word w using key uk, denoted $\mathsf{tk}_{\mathsf{uk}}^{w}$. If the server had $\mathsf{tk}_{k_1}^{w}, \ldots, \mathsf{tk}_{k_n}^{w}$ instead of $\mathsf{tk}_{\mathsf{uk}}^{w}$, the server could match the search token against the encrypted documents using an existing searchable encryption scheme.

Our idea is to enable the server *to compute these tokens by itself*; that is, to adjust the initial $\mathsf{tk}_{\mathsf{uk}}^{w}$ to $\mathsf{tk}_{k_i}^{w}$ for each i. To allow the server to perform the adjustment, the user's machine must initially compute *deltas*, which are cryptographic values that enable a server to adjust a token from one key to another key. We use $\Delta_{\mathsf{uk} \to k_i}$ to denote the delta that allows a server to adjust $\mathsf{tk}_{\mathsf{uk}}^{w}$ to $\mathsf{tk}_{k_i}^{w}$. These deltas represent the user's access to the documents, and crucially, these deltas can be reused for every search, so the user's machine needs to generate the deltas only once. For example, if Alice has access to Bob's data, she needs to provide one delta to the server, and the server will be able to adjust all future tokens from Alice to Bob's key.

In terms of security, our scheme guarantees that the server does not learn the word being searched for, and does not learn the content of the documents. All that the server learns is whether the word in the search token matched some word in a document, and in the case of repeated searches, whether two searches were for the same word. Knowing which documents contain the word being searched for is desirable in practice, to avoid the overhead of returning unnecessary documents.

This chapter presents the multi-key search scheme at a high level, with emphasis on its interface and security properties as needed in our system. We provide a rigorous description and a cryptographic treatment of the scheme (including formal security definitions and proofs) in a technical report [PZ13]. Readers that are not interested in cryptographic details can skip to §4.4.3.

```
*Client-side operations:*
procedure KEYGEN()                                  ▷ Generate a fresh key
    $key \leftarrow$ random value from $\mathbb{Z}_p$
    return $key$
procedure ENC($key$, $word$)
    $r \leftarrow$ random value from $\mathbb{G}_T$
    $c \leftarrow \langle r, H_2(r, e(H(word), g)^{key}) \rangle$
    return $c$
procedure TOKEN($key$, $word$)
                              ▷ Generate search token for matching $word$
    $tk \leftarrow H(word)^{key}$ in $\mathbb{G}_1$
    return $tk$
procedure DELTA($key_1$, $key_2$)
                         ▷ Allow adjusting search token from $key_1$ to $key_2$
    $\Delta_{key_1 \to key_2} \leftarrow g^{key_2/key_1}$ in $\mathbb{G}_2$
    return $\Delta_{key_1 \to key_2}$

*Server-side operations:*
procedure ADJUST($tk$, $\Delta_{k_1 \to k_2}$)
                                  ▷ Adjust search token $tk$ from $k_1$ to $k_2$
    $atk \leftarrow e(\mathsf{tk}, \Delta_{k_1 \to k_2})$ in $\mathbb{G}_T$
    return $atk$
procedure MATCH($atk$, $c = \langle r, h \rangle$)
                            ▷ Return whether $c$ and $atk$ refer to same word
    $h' \leftarrow H_2(r, atk)$
    return $h' \overset{?}{=} h$
```

Figure 4-4: Pseudo-code for Mylar's multi-key search scheme.

◇ 4.4.2 Cryptographic construction

We construct the multi-key search scheme using bilinear maps on elliptic curves, which, at a high level, are functions $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are special groups of prime order p on elliptic curves. Let g be a generator of $\mathbb{G}_2$. Let H and H_2 be certain hash functions on the elliptic curves. e has the property that $e(H(w)^a, g^b) = e(H(w), g)^{ab}$. Figure 4-4 shows pseudo-code for our multi-key search scheme.

◇ 4.4.3 Indexing search

One efficiency issue with this algorithm is that the server has to scan through every word of every document to identify a match. This can be slow if the documents are large, but is unavoidable if the encryption of each word is randomized with a different r, as in Figure 4-4.

To enable the construction of an efficient index over the words in a searchable document, Mylar supports an indexable version of this multi-key search scheme. The idea is to remove randomness without compromising security. Intuitively, randomness is needed to hide whether two words encrypted under the same key are equal. But for words within one document, Mylar can remove the duplicates at the time the document is encrypted, so per-word randomness is not needed within a document.

Therefore, to encrypt a document consisting of words $w_1, \dots, w_n$, the client removes duplicates, chooses one random value r, and then uses the same r when encrypting each of the words using ENC().

When searching for word w in a document, the server performs the adjustment as before and obtains atk. It then computes $v \leftarrow \text{COMBINE}(r, atk) = \langle r, H_2(r, atk)\rangle$ using the document's randomness r. If one of the words in the document is w, its encryption will be equal to v, because they use the same randomness r. Therefore, the server can perform direct equality checks on encrypted words. This means that it can build an index over the encrypted words in the document (e.g., a hash table), and then use that index and v to figure out in constant time if there is a match without scanning the document.

A limitation is that the server has to use an index per unique key rather than one holistic index.

◇ 4.4.4 Integrating search with the principal graph

Mylar integrates the multi-key search scheme with the principal graph as follows. When a principal P is created, Mylar generates a key k_P using KEYGEN (Figure 4-4). Whenever P receives access to some new principal A, Mylar includes k_A in the wrapped key for P. The first time a user with access to P comes online, the Mylar client in that user's browser retrieves k_A from the wrapped key, computes $\Delta_{k_P \to k_A} \leftarrow$DELTA($k_P$, k_A), and stores it at the server. This delta computation happens just once for a pair of principals.

To encrypt a document for some principal A, the user's browser encrypts each word w in the document separately using ENC(k_A, w). Since the multi-key search scheme does not support decryption, Mylar encrypts all searchable documents twice: once with the multi-key search scheme, for searching, and once with a traditional encryption scheme like AES, for decryption.

To search for a word w with principal P, the user's client uses TOKEN(k_P, w) to compute a token tk, and sends it to the server. To search over data encrypted for principal A, the server obtains $\Delta_{k_P \to k_A}$, and uses ADJUST(tk, $\Delta_{k_P \to k_A}$) to adjust the token from k_P to k_A, obtaining the adjusted token atk_A. Then, for each document encrypted under k_A with randomness r, the server computes $v \leftarrow \text{COMBINE}(r, atk_A)$ and checks if v exists in the document using an index. The server repeats the same process for all other principals that P has access to.

Integrating the access graph with keyword search brings up two challenges. The first comes from the fact that our multi-key search scheme allows adjusting tokens just once. In the common case of an access graph where all paths from a user to the data's encryption key consist of one edge (such as the graph in Figure 4-3), Mylar associates the search delta with the edge, and stores it along with the wrapped key. In our chat example, this allows a user's browser to search over all chat rooms that the user has access to, by sending just one search token.

Some applications can have a more complex access graph. For example, in the endometriosis application, all doctors have access to the *staff* principal, which in turn has access to all patient principals. Here, the optimal approach is to use the ADJUST() function on the server between principals with the largest number of edges, so as to maximize the benefit of multi-key search. For instance, if a doctor wanted to search over patient records, the doctor's browser should fetch the *staff* principal it has access to, and produce a search token using the *staff* principal's private key. The server would then use ADJUST() to look for matches in documents encrypted with each patient's key. Because most of our applications have simple access graphs, our prototype does not automate this step, and a developer must choose the principal with which to search.

The second challenge comes from the fact that searching over data supplied by an adversary can leak the word being searched for. For example, suppose an adversary creates a document containing all the words in a dictionary, and gives the user access to that document. If the user searches for a word w in all of the documents he has access to, including the one from the adversary, the server will see which of the words in the adversary's document matches the user's token, and hence will know which dictionary word the user searched for. To prevent this, users must explicitly *accept* access to a shared document, and developers must invoke the **allow_search** function, provided by Mylar for this purpose, as appropriate.

□ 4.5 Verifying client-side code

Although Mylar uses encryption to protect confidential data stored on the untrusted server, the cryptographic keys and the plaintext data are both available to code executing in the user's web browser. The same-origin policy [Zal12] ensures that applications from *other* origins running in the browser do not access the data in the Mylar application. However, Mylar must also ensure that code running in the application's origin has not been tampered with.

Since the code in a web page is static in Mylar, a strawman solution is to sign this code and verify the signature in the browser. The strawman does not suffice because of a combination of two factors. On the one hand, most web applications (including those using Mylar) consist of multiple files served by the web server. On the other hand, the only practical way to control what is loaded in a browser is to interpose on individual HTTP requests.

The problem arises because at the level of individual HTTP requests, it is difficult to reason about what code the browser will execute. For example, if an image is loaded in the context of an `<IMG SRC=...>` tag, it will not execute Javascript code. But if the same image is loaded as a top-level page, the browser's content-sniffing algorithm may decide the file is actually HTML, and potentially execute Javascript code embedded in the image [BCS09]. Thus, a well-meaning developer must be exceedingly careful when including any content, such as images, in their web application. If the developer inadvertently includes a malicious image file in the application, an adversary can cause the browser to load that file as a top-level page [BJM08] and trigger this attack. Similar problems can arise with other content types, including CSS style sheets, PDF files, etc.

Two-origin signing. To address this problem, Mylar uses two origins to host an application. The *primary* origin hosts exactly one file: the application's top-level HTML page. Consequently, this is the only page that can gain access to the application's encryption keys and plaintext data in the browser. All other files, such as images, CSS style sheets, and Javascript code, are loaded from the *secondary* origin. Mylar verifies the authenticity of these files to prevent tampering, but if an adversary tries to load one of these files as a top-level page, it will run with the privileges of the secondary origin, and would not be able to access the application's keys and data.

To verify that the application code has not been tampered with, Mylar requires the site owner to create a public/private key pair, and to sign the application's top-level HTML page (along with the corresponding HTTP headers) with the private key. Any references to other content must refer to the

```
procedure PROCESSRESPONSE(url, cert, response)
                                    ▷ url is the requested URL
                                    ▷ cert is server's X.509 certificate
    if cert contains attribute mylar_pubkey then
        $pk \leftarrow$ cert.mylar_pubkey
        $sig \leftarrow$ response.header["Mylar-Signature"]
        if not VERIFYSIG(pk, response, sig) then
            return ABORT
    if url contains parameter "mylar_hash=h" then
        if hash(response) $\neq$ h then return ABORT
    return PASS
```

Figure 4-5: Pseudo-code for Mylar.'s code verification extension.

secondary origin, and must be augmented to include a mylar_hash=h parameter in the query string, specifying the expected hash of the response. The hash prevents an adversary from tampering with that content or rolling it back to an earlier version. Rollback attacks are possible on the top-level HTML page (because signatures do not guarantee freshness), but in that case, the entire application is rolled back: hashes prevent the adversary from rolling back some but not all of the files, which could confuse the application.

This signing mechanism can verify only the parts of an application that are static and supplied by the web site owner ahead of time. It is up to the application code to safely handle any content dynamically generated by the server at runtime (§4.2.4). This model is a good fit for AJAX web applications, in which the dynamic content is only data, rather than HTML or code.

Browser extension. Each user of Mylar applications should install the Mylar browser extension in their web browser, which verifies that Mylar applications are properly signed before running them. Figure 4-5 shows the pseudo-code for the Mylar browser extension. The site owner's public key is embedded in the X.509 certificate of the web server hosting the web application. Mylar assumes that certificate authorities will sign certificates for the web application's hostname only on behalf of the proper owner of the web application's domain (i.e., the site owner). Thus, as long as the site owner includes the public key in all such certificates, then users visiting the correct web site via `https` will obtain the owner's public key, and will verify that the page was signed by the owner.

□ 4.6 Implementation

We implemented a prototype of Mylar by building on top of the Meteor web application framework [Met13]. Meteor allows client-side code to read and update data via MongoDB operations, and also to issue RPCs to the server. Mylar intercepts and encrypts/decrypts data accessed via the MongoDB interface, but requires developers to explicitly handle data passed via RPCs. We have not found this to be necessary in our experience.

We use the SJCL library [SHB09] to perform much of our cryptography in Javascript, and use elliptic curves for most public-key operations, owing to shorter ciphertexts and higher performance. As in previous systems, Mylar uses faster symmetric-key encryption when possible [PRZB11]. For bilinear pairings, we use the PBC C++ library to improve performance, which runs either as a Native

```
// On both the client and the server:
idp = idp_config(url, pubkey);
Messages.encrypted({"message": "roomprinc"});
Messages.auth_set(["roomprinc", ["id", "message", "room", "date"]]);
Messages.searchable("message");

// On the client:
function create_user(uname, password):
    create_user(uname, password, idp);
function create_room(roomtitle):
    princ_create(roomtitle, princ_current());
function invite_user(username):
    global room_princ;
    room_princ.add_access(princ_lookup(username, idp));
function join_room(room):
    global cur_room, room_princ;
    cur_room = room;
    room_princ = princ_lookup(room.name, room.creator, idp);
function send_message(msg):
    global cur_room, room_princ;
    Messages.insert({message: msg, room: cur_room.id,
                     date: new Date().toString(),
                     roomprinc: room_princ});
function search(word):
    return Messages.search(word, "message", princ_current(), all, all);
```

Figure 4-6: Pseudo-code for changes to the kChat application to encrypt messages. Not shown is unchanged code for managing rooms, receiving and displaying messages, and login/logout (Mylar provides wrappers for Meteor's user accounts API).

Client module (for Chrome), as a plugin (for Firefox), or as an NDK-based application (for Android phones). To verify code in the user's browser, we developed a Firefox extension. Mylar comprises $\sim$9,000 lines of code in total.

When looking up paths in the principal graphs, Mylar performs breadth-first search. We have not found this to be a bottleneck in our experience so far, but more efficient algorithms, such as meet-in-the-middle, are possible.

□ 4.7 Building a Mylar application

To demonstrate how a developer can build a Mylar application, we show the changes that we made to the kChat application to encrypt messages. In kChat, users can create chat rooms, and existing members of a chat room can invite new users to join. Only invited users have access to the messages from the room. A user can search over data from the rooms he has access to. Figure 4-6 shows the changes we made to kChat, using Mylar's API (Figure 4-2).

The call to *Messages*.**encrypted** specifies that data in the "message" field of that collection should be encrypted. This data will be encrypted with the public key of the principal specified in the

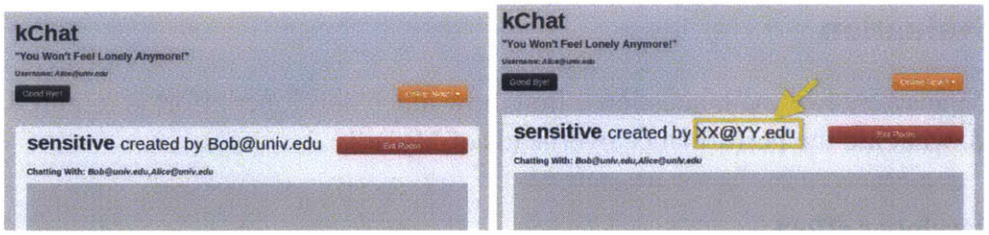
kChat
"You Won't Feel Lonely Anymore!"
sensitive created by Bob@univ.edu
Chatting With: Bob@univ.edu,Alice@univ.edu
kChat
"You Won't Feel Lonely Anymore!"
sensitive created by XX@YY.edu
Chatting With: Bob@univ.edu,Alice@univ.edu

Figure 4-7: Two screenshots from kChat. On the top, Alice is chatting with Bob as intended. On the bottom, the server provided a fake "sensitive" chat room created by the adversary; Alice can detect this by checking the creator's email address.

Application	LoC before	LoC added for Mylar	Number and types of fields secured	Existed before?	Keyword search on
kChat [Kiq]	793	45	1 field: chat messages	Yes	messages
endometriosis	3659	28	tens of medical fields: mood, pain, surgery, ...	Yes	N/A
submit	8410	40	3 fields: grades, homework, feedback	Yes	homework
photo sharing	610	32	5 fields: photos, thumbnails, captions, ...	Yes	N/A
forum	912	39	9 fields: posts body, title, creator, user info, ...	No	posts
calendar	798	30	8 fields: event body, title, date, user info, ...	No	events
WebAthena [Ben13]	4800	0	N/A: used for code authentication only	Yes	N/A

Figure 4-8: Applications ported to Mylar. "LoC before" reports the number of lines of code in the unmodified application, not including images or Meteor packages. "Existed before" indicates whether the application was originally built independent of Mylar.

"roomprinc" field. All future accesses to the *Messages* collection will be transparently encrypted and decrypted by Mylar from this point. The call to *Messages*.**searchable** specifies that clients will need to search over the "message" field; consequently, Mylar will store a searchable encryption of each message in addition to a standard ciphertext.

When a user creates a new room (create_room), the application in turn creates a new principal, named after the room title and signed by the creator's principal. To invite a user to a room, the application needs to give the new user access to the room principal, which it does by invoking **add_access** in invite_user.

When joining a room (join_room), the application must look up the room's public key, so that it can encrypt messages sent to that room. The application specifies both the expected room title as well as the room creator as arguments to **princ_lookup**, to distinguish between rooms with the same title. By displaying both the room title and the creator email address, as in Figure 4-7, the application helps the user distinguish the correct room from an identically named room that an adversary created.

To send a message to a chat room, kChat needs to specify a principal in the **roomprinc** field of the newly inserted document. In this case, the application keeps the current room's principal in the *room_princ* global variable. Similarly, when searching for messages containing a word, the application supplies the principal whose key should be used to generate the search token. In this case, kChat uses the current user principal, **princ_current**().

□ 4.8 Evaluation

This section answers two main questions: first, how much developer effort is required to use Mylar, and second, what are the performance overheads of Mylar?

◇ 4.8.1 Developer effort

To measure the amount of developer effort needed to use Mylar, we ported 6 applications to Mylar. Two of these applications plan to start using Mylar in production in the near future: a medical application in which endometriosis patients record their symptoms, and a web site for managing homework and grades for a class at MIT. We also ported an existing chat application called kChat, in which users share chat rooms by invitation and exchange private messages, and a photo sharing application. We also built a Meteor-based forum and calendar, which we then ported to Mylar. Finally, to demonstrate the generality of Mylar's code verification, we used it to verify the code for WebAthena [Ben13], an in-browser Javascript Kerberos client.

Figure 4-8 summarizes the fields we secured with Mylar in the above applications, along with how much code the developer had to change. In the case of the endometriosis application, fields were stored in the database as field name and field value pairs, so encrypting the generic "value" field secured tens of different kinds of data. In the other apps, a field corresponded to one kind of sensitive data. The results show that Mylar requires little developer effort to protect a wide range of confidential data, averaging 36 lines of code per application.

◇ 4.8.2 Performance

Mylar's performance goal is to avoid significantly affecting the user experience with the web application. To evaluate whether Mylar meets this goal, we answer the following questions:

- How much latency does Mylar add to the web application's overall user interface?
- How much throughput overhead does Mylar impose on a server?
- Is Mylar's multi-key search important to achieve good performance?
- How much storage overhead does Mylar impose?

To answer these questions, we measured the performance of kChat, the homework submission application ("submit"), and the endometriosis application. Although kChat has only one encrypted field, every message sent exercises this field. We used two machines running recent versions of Debian Linux to perform our experiments. The server had an Intel Xeon 2.8 GHz processor and 4 GB of RAM; the client had eight 10-core Intel Xeon E7-8870 2.4 GHz processors with 256 GB of RAM. The client machine is significantly more powerful to allow us to run enough browsers to saturate the server. For browser latency experiments, we simulate a 5 Mbit/s client-server network with 20 msec round-trip latency. All experiments were done over `https`, using nginx as an `https` reverse proxy on the server. We used Selenium to drive a web browser for all experiments. We also evaluated Mylar on Android phones and found that performance remained acceptable, but we omit these results for brevity.

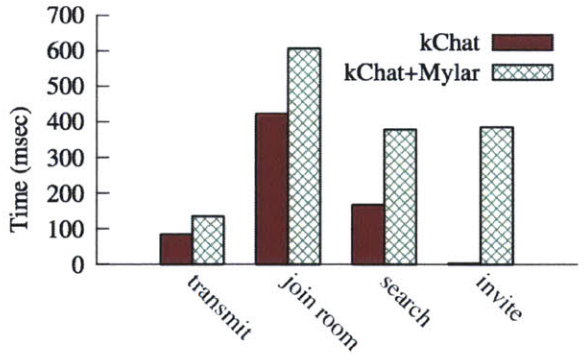

Figure 4-9: End-to-end latency of four operations in kChat. Transmit includes the time from when one user sends a message to when another user receives it.

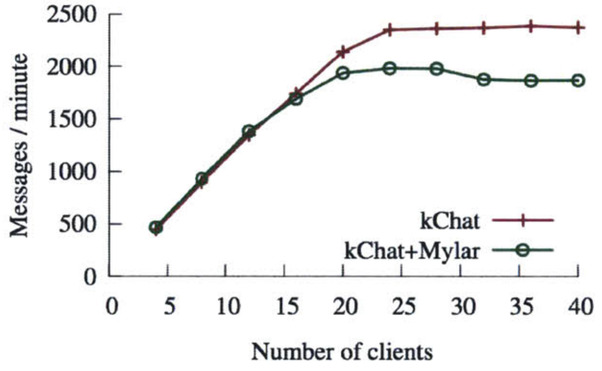

Figure 4-10: Server throughput for kChat.

End-to-end latency. Figure 4-9 shows the end-to-end latency Mylar introduces for four main operations in kChat: transmitting a message, joining a room, searching for a word in all rooms, and inviting a user to a room. For message transmission, we measured the time from the sender clicking "send" until the message renders in the recipient's browser. This is the most frequent operation in kChat, and Mylar adds only 50 msec of latency to it. This difference is mostly due to searchable encryption, which takes 43 msec. The highest overhead is for inviting a user, due to principal operations: looking up and verifying a user principal (218 msec) and wrapping the key (167 msec). Overall, we believe the resulting latency is acceptable for many applications, and subjectively the application still feels responsive.

We also measured the latency of initially loading a page. The original kChat application loads in 291 msec. The Mylar version of kChat, without the code verification extension, loads in 356 msec, owing to Mylar's additional code. Enabling the code verification extension increases the load time to 1109 msec, owing to slow signature verification in the Javascript-based extension. Using native code for signature verification, as we did for bilinear pairings, would reduce this overhead. Note that users experience the page load latency only when first navigating to the application; subsequent clicks are handled by the application without reloading the page.

We also measured the end-to-end latency of the most common operations in the endometriosis application (completing a medical survey and reading such a survey), and the submit application (a

Application	Operation for latency	Latency w/o Mylar	Latency with Mylar	Tput w/o Mylar	Tput with Mylar	Tput units
submit	send and read a submission	65 msec	606 msec	723	394	submissions/min
submit w/o search			70 msec		595	
endometriosis	fill in/read survey	1516 msec	1582 msec	6993	6130	field updates/min

Figure 4-11: Latency and throughput (abbreviated tput) of different applications with and without Mylar. The latency is the end-to-end time to perform the most common operation in that application. For submit, the latency is the time from one client submitting an assignment until another client obtains that submission. For endometriosis, the latency is the time from one client filling out a survey until another client obtains the survey.

student uploading an assignment, and a staff member reading such a submission); the results are shown in Figure 4-11. For the submit application, we used real data from 122 students who used this application during the fall of 2013 in MIT's 6.858 class. Submit's latency is higher than that of other applications because the amount of data (student submissions) is larger, so encryption with search takes longer. For comparison, we also show the latency of submit when search is turned off. The search encryption can happen asynchronously so the user does not have to wait for it.

Throughput. To measure Mylar's impact on server throughput, we used kChat, and we set up many pairs of browsers—a sender and a receiver—where the sender continuously sends new messages. Receivers count the total number of messages received during a fixed interval. Figure 4-10 shows the results, as a function of the total number of clients (each pair of browsers counts as 2 clients). Mylar decreases the maximum server throughput by 17%. Since the server does not perform any cryptographic operations, Mylar's overhead is due to the increase in message size caused by encryption, and the encrypted search index that is added to every message to make it searchable.

Figure 4-11 also shows the server throughput of the endometriosis and class submit application when clients perform representative operations.

Search. To evaluate the importance of Mylar's multi-key search, we compare it to two alternative approaches for secure search. The first alternative is single-key server-side search, in which the client generates a token for every key by directly computing the adjusted token from our multi-key search. This alternative is similar to prior work on encrypted keyword search. In this case, the client looks up the principal for every room, computes a token for each, and the server uses one token per room. The second alternative is to perform the search entirely at the client, by downloading all messages. In this case, the client still needs to look up the principal for each room so that it can decrypt the data.

Figure 4-12 shows the time taken to search for a word in kChat for a fixed number of total messages spread over a varying number of rooms, using multi-key search and the two alternatives described above. We can see that multi-key search is much faster than either of the two alternatives, even with a small number of rooms. The performance of the two alternatives is dominated by the cost of looking up the principal for each room and obtaining its private key. Multi-key search does not need to do this, because the server directly uses the deltas, and it achieves good performance because both ADJUST and MATCH are fast, as shown in Figure 4-13.

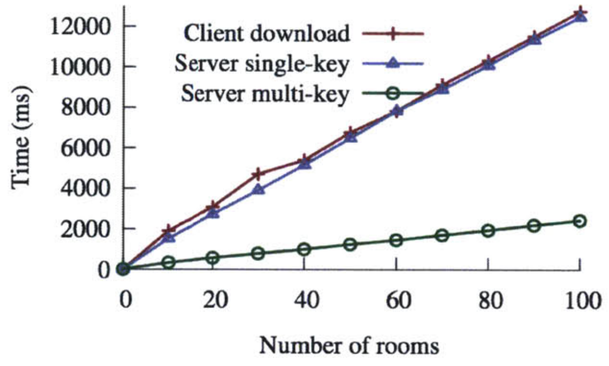

Figure 4-12: End-to-end latency of keyword search in kChat, searching over 100 6-word messages, spread over a varying number of rooms.

Encrypt	Delta	Token	Adjust	Match
6.5 ms	7.1 ms	0.9 ms	5.6 ms	0.007 ms

Figure 4-13: Time taken to run each multi-key search operation.

Storage overhead. For kChat, the server storage overhead after inserting 1,000 messages with Mylar was $4\times$ that of unmodified kChat. This is due to three factors: principal graphs (storing certificates and wrapped keys), symmetric key encryption, and searchable encryption. Our prototype stores ciphertexts in base-64 encoding; using a binary encoding would reduce storage overheads.

□ 4.9 Discussion

Mylar focuses on protecting confidential data in web applications. However, Mylar's techniques for searching over encrypted data and for verifying keys are equally applicable to desktop and mobile phone applications; the primary difference is that code verification becomes simpler, since applications are explicitly installed by the user, instead of being downloaded at application start time.

Mylar relies on X.509 certificates to supply the web site owner's public key for code verification. Alternative schemes could avoid the need for fully trusted certificate authorities [Tho11, WAP08], and the Mylar extension could allow users to manually specify site owner public keys for especially sensitive web sites.

Revoking access to shared data is difficult, because Mylar cannot trust the server to forget a wrapped key. Complete revocation requires re-encrypting shared data under a new key, and giving legitimate users access to the new key. In less sensitive situations, it may suffice to try deleting the key from the server, which would work if the server is not compromised at the time of the deletion.

□ 4.10 Conclusion

In this chapter, we presented Mylar, a novel web application framework that enables developers to protect confidential data in the face of arbitrary server compromises. Mylar leverages the recent shift to exchanging data, rather than HTML, between the browser and server, to encrypt all data stored on the server, and decrypt it only in users' browsers. Mylar provides a principal abstraction to securely share data between users, and uses a browser extension to verify code downloaded from the server that runs in the browser. For keyword search, which is not practical to run in the browser, Mylar introduces a cryptographic scheme to perform keyword search at the server over data encrypted with different keys. Experimental results show that using Mylar requires few changes to an application, and that the performance overheads of Mylar are modest.

Mylar and the applications discussed in this chapter are available at http://css.csail.mit.edu/mylar/.

CHAPTER 5

Multi-key searchable encryption

In this section, we present the multi-key searchable encryption scheme we designed for Mylar, and prove its security formally. This scheme is an example of a situation when there did not exist an encryption scheme that fit Mylar's scenario, and we had to design a new scheme. In a nutshell, this scheme enables keyword search over data encrypted with *different* keys. We present the scheme standalone, independent from Mylar, because we think it could be useful in other settings as well.

□ 5.1 Introduction

As discussed in this thesis, a promising approach to protecting data confidentiality against adversaries who compromise servers is to store encrypted data on servers, and to encrypt and decrypt documents only on client machines. In the case of a multi-user application, each user may have access to a different set of documents stored on the server; this can be achieved by ensuring that each document is encrypted with a separate per-document key, and arranging for each user's client machine to have access to the keys of the documents that the corresponding user has access to.

One challenge with this approach lies in supporting applications that allow users to search for documents that contain a given word. Many applications, such as document sharing, chat, forums, and calendars, support search over documents shared by different users. Prior work on searchable encryption schemes would require the client to provide the server with a search token under each key that a matching document might be encrypted with, and thus the number of tokens scales with the number of documents to search. This can be slow when there is a large number of documents.

We present a cryptographic scheme that allows a client to provide a single search token to the server, but still allows the server to search for that token's word in documents *encrypted with different keys*. We call such a scheme *multi-key search*. Intuitively, the scheme hides the content of the document and the words one searches for, and the only information the server learns is whether some word being searched for matches a word in a document. We formalize the security guarantees with cryptographic security definitions and prove the security of our scheme under variants of the Bilinear Decisional Diffie-Hellman and External Diffie-Hellman assumptions, as well as in the random oracle model. The scheme is practical and was designed to be included in a new system for protecting data

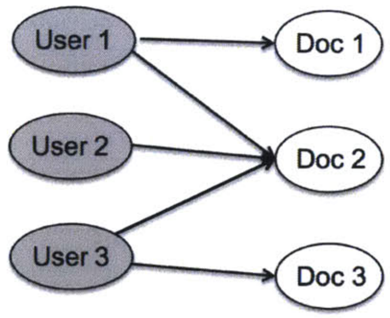

Figure 5-1: Access graph example.

confidentiality against attacks on the server.

The most challenging aspect when coming up with such a scheme is that there is no single trusted user; for example, in many web applications, anyone, including an adversary, can create an account and become a user. As a result, users cannot agree on a secret, and each document must be encrypted under different keys that are generated independently, rather than generated from a common secret key. Another challenge is that the scheme must be practical because our goal is to use it in a real system.

In the rest of the chapter, we explain the problem setting in Sec. 5.2, we provide syntax and security definitions in Sec. 5.4, we present our construction together with a performance measurement in Sec. 5.5 and 5.6, respectively, and finally, we prove the security of our scheme in Sec. 5.8 under the assumptions in Sec. 5.7. We compare our scheme to related work in Sec. 8.2.3.

□ 5.2 Problem setting

In our model, there is a set of users, a server, and a set of documents. The server stores encrypted documents. Each user has access to a subset of the documents. A user can create a document and then give access to other users to the document by giving them the decryption key of the document. We call the graph of user accesses to documents, an *access graph*, defined below. Fig. 5-1 shows an example of an access graph.

Definition 14 (Access graph)**.** *An access graph $G = (U, D, E)$ consists of a set of users U, a set of documents D, as well as a set of edges E, where an edge e is a pair (i, j) for $i \in U$ and $j \in D$ denoting user i has access to document j. We write $e \in G$ to mean that $e \in E$.*

At a high level, the following security guarantees are desirable. If some user was not given access to a document, the user should not be able to read the contents of that document or search over that document, *even if the user colludes with the server.* The setting is entirely distributed. Each user generates his key and there is no trusted party for choosing keys, and no globally trusted user. Moreover, there is no trusted party to create document keys or to help with providing access to documents.

The functionality goal is to allow a user to search a word over all the documents he can access, say n documents, even if those documents are encrypted under different keys. Note that the user has access to all the keys for these n documents, but the user should only give one search token to the server, instead of n tokens.

Let's now consider a more concrete model for such a multi-key search. We denote the key of user i with uk_i, and the key of document j with k_j. Consider that a user, say Alice, (with key uk_A) has n encrypted documents at the server, and each is encrypted under a key k_j for $j = 1 \dots n$. Alice wants to search for a word w over all the documents she has access to, so she uses uk_A to compute a *token* for a word w. In order to allow the server to match the token against words encrypted with $k_1, \dots, k_n$, Alice gives the server some public information called *delta*. Alice provides one delta per key k_j, denoted $\Delta_{\mathsf{uk}_A, k_j}$. The server can use $\Delta_{\mathsf{uk}_A, k_j}$ to convert a search token under key uk_A to a search token under k_j, a process we call *adjust*. In this way, the server can obtain tokens for word w under $k_1, \dots, k_n$ while only receiving one token from Alice, and then performing a traditional single-key search with the new tokens.

Multi-key search provides efficiency guarantees over single-key search. If T is the total number of words Alice searches, she provides $O(n + T)$ pieces of information to the server: n deltas and T tokens, the size of all of which only depends on the security parameter. In contrast, if Alice uses a single-key searchable encryption as in previous work, she provides $O(nT)$ pieces of information to the sever, because she provides n tokens, one for each key k_j, for each of T words.

□ 5.3 Preliminaries

We denote by κ the security parameter throughout this chapter. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ when x is sampled from the distribution $\mathcal{D}$. If S is a finite set, by $x \leftarrow S$ we mean x is sampled from the uniform distribution over the set S.

We use $p(\cdot)$ to denote that p is a function that takes one input. Similarly, $p(\cdot, \cdot)$ denotes a function p that takes two inputs.

We say that a function f is negligible in an input parameter κ, if for all $d > 0$, there exists K such that for all $\kappa > K$, $f(\kappa) < k^{-d}$. For brevity, we write: for all sufficiently large κ, $f(\kappa) = \text{negl}(\kappa)$. We say that a function f is polynomial in an input parameter κ, if there exists a polynomial p such that for all κ, $f(\kappa) \leq p(\kappa)$. We write $f(\kappa) = \text{poly}(\kappa)$. A similar definition holds for $\text{polylog}(\kappa)$.

Let $[n]$ denote the set $\{1, \dots, n\}$ for $n \in \mathbb{N}^*$.

When saying that a Turing machine A is PPT we mean that A is a probabilistic polynomial-time machine.

Two ensembles, $X = \{X_\kappa\}_{\kappa \in \mathbb{N}}$ and $Y = \{Y_\kappa\}_{\kappa \in \mathbb{N}}$, are said to be *computationally indistinguishable* (denoted $\{X_\kappa\}_{\kappa \in \mathbb{N}} \overset{c}{\approx} \{Y_\kappa\}_{\kappa \in \mathbb{N}}$) if for every probabilistic polynomial-time algorithm D,

$$|\Pr[D(X_\kappa, 1^\kappa) = 1] - \Pr[D(Y_\kappa, 1^\kappa) = 1]| = \text{negl}(\kappa).$$

We use asymmetric bilinear map groups of Type 2 for our construction [GPS08]. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two disjoint cyclic subgroups on an elliptic curve of Type 2, and let e be a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$. Let $\mathsf{params} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T) \leftarrow \mathsf{CSetup}(1^\kappa)$ be the procedure that generates curve parameters, where g_1, g_2, and g_T are generators of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$.

□ 5.4 Syntax and security definitions

We now formalize the syntax and security definitions.

Definition 15 (Multi-key search). *A multi-key search scheme* MK *is a tuple of algorithms (*MK.Setup, MK.KeyGen, MK.Delta, MK.Token, MK.Enc, MK.Adjust, MK.Match*) as follows:*

- params $\leftarrow$ MK.Setup(1^κ)*: Takes as input the security parameter and outputs system wide parameters.*
- $k \leftarrow$ MK.KeyGen(params)*: Takes as input the system parameters and outputs a secret key, which could be a key for a user or for a document.*
- $\Delta \leftarrow$ MK.Delta(k_1, k_2)*: Takes as input two keys and outputs a delta.*
- tk $\leftarrow$ MK.Token(k, w)*: Takes as input a key k and a word w and outputs a search token* tk.
- $c \leftarrow$ MK.Enc(k, w)*: Takes as input a key k and a word w and outputs an encryption of the word c.*
- stk $\leftarrow$ MK.Adjust(tk, Δ)*: Takes as input a token* tk *and a delta Δ and outputs a search token* tk$'$.
- $b \leftarrow$ MK.Match(stk, c)*: Takes as input a search token* stk *and a ciphertext c and outputs a bit* b.

Correctness. *For any polynomial $n(\cdot)$, for every sufficiently large security parameters κ, for all $w \neq w' \in \{0,1\}^{n(\kappa)}$,*

$$\Pr\left[\begin{array}{l}\mathsf{params} \leftarrow \mathsf{MK.Setup}(1^\kappa);\\ \mathsf{uk} \leftarrow \mathsf{MK.KeyGen}(\mathsf{params});\, k \leftarrow \mathsf{MK.KeyGen}(\mathsf{params});\\ \Delta \leftarrow \mathsf{MK.Delta}(\mathsf{uk}, k);\\ \mathsf{stk} \leftarrow \mathsf{MK.Adjust}(\mathsf{MK.Token}(\mathsf{uk}, w), \Delta):\\ \mathsf{MK.Match}(\mathsf{stk}, \mathsf{MK.Enc}(k, w)) = \textit{True and } \mathsf{MK.Match}(\mathsf{stk}, \mathsf{MK.Enc}(k, w')) = \textit{False}\end{array}\right] = 1-\mathrm{negl}(\kappa).$$

Correctness says that when searching for a word w, encryptions of the word w in some document will match (after adjusting the token for w to the key of the document), but encryptions of a different word w' will not match the search.

For simplicity, we do not include a decryption algorithm in the syntax of the scheme, but a multi-key search scheme can be easily augmented with a decryption algorithm by appending to each ciphertext produced in MK.Enc a symmetric-key semantically secure encryption with the same key as the argument to MK.Enc.

Remark 1. *In an alternate syntax, each user has a public key* pk*, and the algorithm* MK.Delta *takes as input the public key of a user instead of his private key. A public-key* MK.Delta *algorithm has the advantage that when a user, say Alice, wants to give access to another user, say Bob, to a document, Alice can just compute the delta to the document for Bob and provide it to the server. (In fact, our construction can be adapted to public-key by setting the public key of a user to* pk $= g_2^{1/\mathsf{uk}}$*, where* uk *is the secret key of the user.)*

However, inherently, such a multi-key scheme cannot hide the word searched for because the functionality of the scheme allows a dictionary attack. Assume that an adversary wants to learn what Alice searches for, and let pk$_A$ *be Alice's public key. An adversary can create a document* *with some key k, and encrypt in this document every word of a dictionary using key k. Then, the adversary can produce a delta for Alice to this document by computing $\Delta_A := \mathsf{MK.Delta}(\mathsf{pk}_A, k)$. Now, for every search token* tk *of Alice, the adversary computes* $\mathsf{stk} := \mathsf{MK.Adjust}(\mathsf{tk}, \Delta_A)$ *and uses* stk *to find a match in the encrypted dictionary. Once a match is found, the adversary knows what word the user searched for.*

Intuitively, we want two security properties from the MK scheme: the ciphertext and the token should not reveal the value of the underlying plaintext, and the only information revealed to the server is whether a search token matches a ciphertext only when the server has a delta for some document or whether one is searching for the same word as before. Moreover, if the key of a document leaks, the key of the user should not leak and the contents of the other documents the user has access to should not leak.

We formalize these properties with two games, *data hiding* and *token hiding*, that express these goals. One holistic security definition would be a stronger guarantee, but that greatly complicates the proofs. Nevertheless, the separate definitions also capture the desired security goals.

◇ 5.4.1 Data hiding

Data hiding requires that the adversary not be able to distinguish between ciphertexts of two values not matched by some token. The case when the token matches a ciphertext is handled by the token hiding game. In the following definition, documents are numbered from 0 onwards and users from 1 onwards. The reason there is document 0 is that this is a special document used in the challenge.

Definition 16 (Data hiding game). *The data hiding game is between a challenger* Ch *and an adversary* Adv *on security parameter κ and public parameters* params.

- Ch *computes* $\mathsf{params} \leftarrow \mathsf{CSetup}(1^\kappa)$ *and provides them to* Adv.
- Adv *provides an access graph G with users numbered from 1 and documents numbered from 0 to* Ch *along with keys k_j for every document with $j > 0$.*
- Ch *generates* $k_0 \leftarrow \mathsf{MK.KeyGen}(1^\kappa, \mathsf{params})$ *for document 0. Then, for every user i, it generates* $\mathsf{uk}_i \leftarrow \mathsf{MK.KeyGen}(1^\kappa)$ *and for every edge $(i, j) \in G$, it provides* $\mathsf{MK.Delta}(\mathsf{uk}_i, k_j)$ *to* Adv.
- *Challenge step:* Adv *chooses* $w_0^*, w_1^* \leftarrow \{0,1\}^{n(\kappa)}$ *and provides* w_0^*, w_1^* *to* Ch. Ch *chooses a random bit* b *and provides* $\mathsf{MK.Enc}(k_0, w_\mathsf{b}^*)$ *to* Adv.
- *Adaptive step:* Adv *makes the following queries to* Ch *adaptively. The ℓ-th query can be:*
  1. *"Encrypt w_ℓ to document 0":* Ch *returns* $\mathsf{MK.Enc}(k_0, w_\ell)$.
  2. *"Token for word w_ℓ for user i":* Ch *returns* $\mathsf{MK.Token}(\mathsf{uk}_i, w_\ell)$.
- Adv *outputs* b$'$, *its guess for* b.

Restriction on Adv: *for all token queries w_ℓ for user i, if $(i, 0) \in G$, it must be that $w_\ell \notin \{w_0^*, w_1^*\}$.*

Adv *wins the game if b$'$ = b. Let* $\mathsf{win}_{\mathsf{Adv}}(\kappa)$ *be the random variable indicating whether* Adv *wins the game for security parameter κ.*

Definition 17. *A multi-key search scheme is data hiding if, for all* PPT *adversaries* Adv, *for all sufficiently large* κ, $\Pr[\mathsf{win}_{\mathsf{Adv}}(\kappa)] < 1/2 + \text{negl}(\kappa)$.

Here is how the definition models our intentions:

- The fact that Adv can provide keys for all documents except for the challenge one models the fact that an adversary could steal keys of document or create documents, but such actions should not allow Adv to learn information about a document he does not own.
- The restriction on the token queries of Adv is required because otherwise Adv could distinguish the ciphertexts based on the functionality of the scheme.
- Note that Adv can ask tokens for words that are part of the challenge (e.g., w_0 or w_1) for users that do not have a delta to document 0. This ensures that any user i that does not have a delta to a document cannot search that document.
- We do not need to allow Adv to ask for encrypt queries to documents i for $i > 0$ because Adv has the corresponding secret keys and can encrypt by itself.

A stronger definition would allow an adaptive step before the challenge step as well. Our scheme can also be proven secure in that setting, but results in a more complicated proof, which we do not provide here.

◇ 5.4.2 Token hiding

Token hiding requires that an adversary cannot learn the word one searches for.

Definition 18. *A u-free document in a particular graph is a document with no edge from user u in that graph. A u-free user in a particular graph is a user that has edges only to u-free documents in that graph.*

User 0 will be the challenge user, for which Adv will have to distinguish tokens. Thus, we will refer to 0-free users and 0-free documents as simply "free users" and "free documents".

Definition 19 (Token hiding game). *The token hiding game is between a challenger* Ch *and an adversary* Adv *on security parameter κ and public parameters* params.

- Ch *computes* params $\leftarrow$ CSetup(1^κ) *and provides them to* Adv.
- Adv *provides an access graph G with users numbered from* 0 *and documents numbered from* 1, *along with keys* uk$_i$ *for every free user i and k_j for every free document j.*
- Ch *generates* uk$_i$ $\leftarrow$ MK.KeyGen(1^κ) *for every non-free user i, $k_j \leftarrow$* MK.KeyGen(1^κ) *for every non-free document j. For every edge $(i, j) \in G$,* Ch *sends* MK.Delta(uk$_i$, k_j) *to* Adv.
- *Adaptive step.* Adv *makes the following queries to* Ch *adaptively. At query ℓ:*

  1. *"Encrypt w_ℓ for document j":* Ch *returns* MK.Enc(k_j, w_ℓ).
  2. *"Token w_ℓ for user i" with $i > 0$: receives* MK.Token(uk$_i$, w_ℓ).

- *Challenge step:* Adv *sends* w_0^* *and* w_1^* *to* Ch *and receives* MK.Token($\mathsf{uk}_0, w_\mathsf{b}^*$) *for a random bit* b.

- Adv *repeats the adaptive step.*

- Adv *outputs* b′, *its guess for* b.

Restriction on Adv*: For every "Token* w_ℓ *for user* i*" query:* $w_\ell \notin \{w_0^*, w_1^*\}$ *or user* i *is free. For every "Encrypt* w_ℓ *for document* j*" query:* $w_\ell \notin \{w_0^*, w_1^*\}$ *or document* j *is free.*

Adv *wins the game if* $b' = b$. *Let* $\mathsf{win}_{\mathsf{Adv}}^{\mathsf{token}}(\kappa)$ *be the random variable indicating whether* Adv *wins the game for security parameter* κ.

Definition 20. *A multi-key search scheme is token-hiding if, for all* PPT *adversaries* Adv, *for all sufficiently large* κ, $\Pr[\mathsf{win}_{\mathsf{Adv}}^{\mathsf{token}}(\kappa)] < 1/2 + \text{negl}(\kappa)$.

As before, the reason Adv can pick keys is to signify that Adv can corrupt certain users or documents, or can even create nodes in the access graph.

The constraints on the game are so that the adversary cannot distinguish the challenge words trivially, because the functionality of the scheme distinguishes them (either because there is a search match or the token is deterministic). Note that the definition (and in fact the scheme as well) allows an adversary to tell if two tokens are equal: in practice, if the same set of documents match a token, it is likely that the token is the same so we did not consider important to hide this equality relation among tokens. A solution for hiding the token is to use composite groups and multiply a random element from the second group to the token, but we do not explore this further here.

□ 5.5 Construction

Let $H : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \mathbb{G}_T \times \mathbb{G}_T \rightarrow \{0,1\}^*$ be hash functions, modeled as random oracles. Our multi-key search scheme is as follows:

- params $\leftarrow$ MK.Setup(1^κ): return $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T) \leftarrow$ CSetup(1^κ).

- $k \leftarrow$ MK.KeyGen(params): return $k \leftarrow \mathbb{Z}_p$.

- $\Delta \leftarrow$ MK.Delta(k_1, k_2): return $\Delta = g_2^{k_2/k_1} \in \mathbb{G}_2$.

- tk $\leftarrow$ MK.Token(k, w): return tk $= H(w)^k \in \mathbb{G}_1$.

- $c \leftarrow$ MK.Enc(k, w): Draw $r \leftarrow \mathbb{G}_T$. Output $c = \big(r, H_2(r, e(H(w), g_2)^k)\big)$.

- tk′ $\leftarrow$ MK.Adjust(tk, Δ): return tk′ $= e(\mathsf{tk}, \Delta) \in \mathbb{G}_T$.

- $b \leftarrow$ MK.Match(tk, c): Let $c = (r, h)$. Return $H_2(r, \mathsf{tk}) \stackrel{?}{=} h$.

Remark 2 (Alternate constructions). *Using asymmetric pairings here is crucial for security. With symmetric pairings ($G_1 = G_2$), there is an attack that can determine the search word: given $H(w)$, $H(w)^k$, and $H(w_2)$, one can distinguish $H(w_2)^k$ from R by computing crossed pairings and thus* *can do a dictionary attack. Asymmetric groups prohibit applying the pairing between elements of* $\mathbb{G}_1$.

Another way to hide the search token would be to employ composite-order groups and multiply a random element $R \in G_h$ *by the token for a word. One can also simulate composite order groups with standard groups using the orthogonal vector space techniques of Freeman and Lewko, which enables faster implementations.*

Remark 3 (Indexed search). *If the encryption scheme were deterministic, it would be easier to search for matches because an index could be constructed over the data. To make the scheme indexable in this way, one can modify* $\mathsf{MK.Enc}$ *to just output* $e(H(w), g_2)^k$. *If a user makes sure that there are no repetitions of words* w *in a document encrypted with the same key, making the encryption deterministic results in roughly the same security guarantees (although the data-hiding definitions need a few changes).*

Theorem 3. *The scheme above is a data- and token-hiding multi-key search scheme, based on the BDHV and XDHV assumptions in the random oracle model for* H *and* H_2.

Proof. We prove correctness of the scheme here, and in Sec. 5.8, we prove that it achieves the security properties.

Consider the setup from the correctness definition: $\mathsf{params} \leftarrow \mathsf{MK.Setup}(1^\kappa)$, $k_1 \leftarrow \mathsf{MK.KeyGen}(\mathsf{params})$, $k_2 \leftarrow \mathsf{MK.KeyGen}(\mathsf{params})$, $\Delta \leftarrow \mathsf{MK.Delta}(k_1, k_2)$, $\mathsf{tk} \leftarrow \mathsf{MK.Adjust}(\mathsf{MK.Token}(k_1, w), \Delta)$.

This means that $\mathsf{tk} = e(H(w)^{k_1}, g_2^{k_2/k_1}) = e(H(w), g_2)^{k_2}$.

Then $H_2(r, \mathsf{tk}) = H_2(r, e(H(w), g_2)^{k_2})$, so $\mathsf{MK.Match}(\mathsf{tk}, \mathsf{MK.Enc}(k_2, w))$ outputs True as desired.

The chance that $H_2(r, e(H(w), g_2)^{k_2}) = H_2(r, e(H(w'), g_2)^{k_2})$ is statistically negligible (in fact, it can be zero if the hash functions' output size is not smaller than the input size). Therefore, $\mathsf{MK.Match}(\mathsf{tk}, \mathsf{MK.Enc}(k_2, w'))$ outputs False. We thus showed correctness of our scheme.

□

□ 5.6 Implementation

We implemented the scheme in C++ and used the PBC library [PBC] for implementation of a Type 2 curve [GPS08], called Type D in the library. Below are evaluation results on an AMD Opteron(tm) Processor 2.4GHz, running on one core, when scheme is encrypting average-sized words, randomly generated. The scheme has a modest overhead.

Algorithm	MK.KeyGen	MK.Delta	MK.Token	MK.Enc	MK.Adjust	MK.Match
Time (ms)	0.35	6.3	0.89	6.3	5.5	0.0021

□ 5.7 Assumptions

Our construction can be proven secure under variants of the Decisional Diffie-Hellman and External Diffie-Hellman assumptions, both of which are standard assumptions and were used in previous constructions, and in the random oracle model. Our assumptions are simple variants of these, and one can verify they hold in the generic group model.

Definition 21 (Bilinear Diffie-Hellman Variant (BDHV) assumption)**.** *For all* PPT *algorithms* Adv, *for every sufficiently large security parameter* κ*,*

$$\begin{aligned}|\Pr[\mathsf{params} \leftarrow \mathsf{CSetup}(1^\kappa);\quad & a, b, c \leftarrow \mathbb{Z}_p : \mathsf{Adv}(\mathsf{params}, g_1^a, g_2^b, g_2^{1/a}, g_1^c, e(g_1, g_2)^{abc}) = 1] -\\ \Pr[\mathsf{params} \leftarrow \mathsf{CSetup}(1^\kappa);\quad & a, b, c \leftarrow \mathbb{Z}_p, R \leftarrow \mathbb{G}_T : \mathsf{Adv}(\mathsf{params}, g_1^a, g_2^b, g_2^{1/a}, g_1^c, R) = 1]| = \mathrm{negl}(\kappa).\end{aligned}$$

Definition 22 (External Diffie-Hellman Variant (XDHV) assumption)**.** *For all* PPT *algorithms* Adv, *for every sufficiently large security parameter* κ*,*

$$\begin{aligned}|\Pr[\mathsf{params} \leftarrow \mathsf{CSetup}(1^\kappa);\quad & a, b, c, m \leftarrow \mathbb{Z}_p : \mathsf{Adv}(\mathsf{params}, g_1^a, g_1^b, g_1^{ab}, g_2^{ca}, g_2^{cd}, g_1^d, g_2^{1/d}) = 1] -\\ \Pr[\mathsf{params} \leftarrow \mathsf{CSetup}(1^\kappa);\quad & a, b, c, m \leftarrow \mathbb{Z}_p, R \leftarrow \mathbb{G}_1 : \mathsf{Adv}(\mathsf{params}, g_1^a, g_1^b, R, g_2^{ca}, g_2^{cd}, g_1^d, g_2^{1/d}) = 1]|\\ = \mathrm{negl}(\kappa).\end{aligned}$$

This assumption consists of the XDH assumption in the first three terms, but with extra information about a in the form of g_2^{ca}, but masked by c, which itself is masked by d.

As mentioned, we also model the hash functions H and H_2 as random oracles.

□ 5.8 Security proof

The proofs are in the random oracle model for H and H_2, and H is a programmable random oracle.

To show that our scheme is secure with either of the security games, we consider a sequence of hybrid games starting from the game in the security definition in consideration, moving through gradually simpler games, and reaching the final game; in the final game, no adversary can guess the challenge bit b correctly with more than negligible chance information-theoretically.

During the sequence of hybrid games, we will sometimes show that Game "target" $\Leftarrow$ Game "new", meaning that if a scheme is secure in Game "new", it will be secure in Game "target", so it suffices to prove that the scheme is secure in Game "new".

Other times, we will show that Game "target" $\Leftrightarrow$ Game "new" meaning that the games are computationally indistinguishable. We will not review here the notion of game indistinguishability. Loosely speaking, any PPT adversary $\mathcal{D}$ playing the role of the adversary in Game "target" and Game "new" cannot tell in which of the two games it is. If two games are computationally indistinguishable and no PPT adversary can win in one game with more than negligible probability, then no PPT adversary can win in the other game either.

For brevity, we do not include in the hybrid games the initial step in which Ch computes params $\leftarrow$ CSetup(1^κ) and provides them to Adv, the fact that Adv always has access to H_1 and H_2, as well as the final step when Adv outputs his guess for b, the challenger's bit. For clarity, we highlight certain parts of a game in blue, to indicate that these are differences from the previous game.

◇ 5.8.1 Data hiding proof

Proof. The sequence of hybrid games in the proof are related as follows:

Data hiding game $\Leftarrow$ Game 1 $\Leftarrow$ Game 2 $\Leftarrow$ Game 3 $\overset{\text{BDHV}}{\Leftrightarrow}$ Game 4 $\Leftarrow$ Game 5.

Games 1– 3 provide gradual simplications of the original game. Game 4 is computationally indistinguishable from Game 3 based on the BDHV assumption. In Game 5, any adversary has chance of guessing statistically close to $1/2$.

Game 1 no longer has the keys k_j for $j > 0$: see the difference in blue. We will also replace the algorithms of the multi-key scheme with the exact quantities returned.

Game 1

- Adv_1 provides an access graph G with one document, labeled 0, and any number of users.
- Ch_1 generates $k_0 \leftarrow \mathsf{MK.KeyGen}(1^\kappa, \mathsf{params})$ for document 0. Then, Ch_1 provides $g_1^{k_0/\mathsf{uk}_i}$ for every edge $(i, 0) \in G$, and g_1^{1/uk_i} for every user i.
- Adv_1 provides w_0^*, w_1^*.
- Ch_1 chooses a random bit b and provides $r^*, H_2(r^*, e(H(w_{\mathsf{b}}^*), g_2)^{k_0})$.
- Adaptive step: Adv_1 makes the following queries to Ch_1 adaptively. The ℓ-th query can be:
  1. "Encrypt w_ℓ": Ch_1 returns $r_\ell, H_2(r_\ell, e(H(w_\ell), g_2)^{k_0})$.
  2. "Token for word w_ℓ for user i": Ch_1 returns $H(w_\ell)^{\mathsf{uk}_i}$.

Restriction on Adv_1: for all token queries w_ℓ for user i, if $(i, 0) \in G$, it must be that $w_\ell \notin \{w_0^*, w_1^*\}$.

If the scheme is secure in this game, then it is secure in the data hiding game. The reason is that if there is there is a PPT adversary Adv that wins in the data hiding game, there is a PPT adversary Adv_1 that wins the Game 1. Adv_1 can use Adv to win Game 1. Adv_1 can simulate the inputs to Adv by simply storing the k_j values from Adv and computing g^{k_j/uk_i} when given g^{1/uk_i}, as in the third step of the data-hiding game that Adv is expecting.

Next, we would like to remove from the game users that do not have access to document 0. The intuition is that whatever information the adversary gets about those users is unrelated to document 0 and hence to the challenge. We create a new game in which the adversary creates only users with access to document 0.

Game 2

- Adv_2 provides an access graph G with one document, labeled 0, and any number of users all with access to document 0.
- Ch_2 generates $k_0 \leftarrow \mathsf{MK.KeyGen}(1^\kappa, \mathsf{params})$ and provides $g_2^{k_0/\mathsf{uk}_i}$ and g_2^{1/uk_i} for every user i.
- Adv_2 provides w_0^*, w_1^*.

- Ch_2 chooses a random bit b and provides $r^*, H_2(r^*, e(H(w_\mathsf{b}^*), g_2)^{k_0})$.
- Adaptive step: Adv_2 makes the following queries to Ch_2 adaptively. The ℓ-th query can be:

    1. "Encrypt w_ℓ": Ch_2 returns $r_\ell, H_2(r_\ell, e(H(w_\ell), g_2)^{k_0})$.
    2. "Token for word w_ℓ for user i": Ch_2 returns $H(w_\ell)^{\mathsf{uk}_i}$.

Restriction on Adv_2: for all w_ℓ in token queries, $w_\ell \notin \{w_0^*, w_1^*\}$.

Claim 1. *If a scheme is secure in Game 2, the scheme is secure in Game 1.*

Proof. For contradiction, let Adv_1 be an adversary that breaks Game 1, and let us construct an adversary Adv_2 that breaks Game 2. Adv_2's strategy is as follows: for users i with access to doc 0, Adv_2 uses its challenger Ch_2 to answer token queries of Adv_1; for other users, Adv_2 generates a random key for each such user i, uk_i, and answers Adv_1's queries using that key.

Let Ch_2 be the challenger of Adv_2 in Game 2. Adv_2 works as follows:

1. Receive a graph G from Adv_1. Construct a graph G' which is G from which we remove the users with no access to doc 0 as well as their edges. Provide G' to Ch_2. Receive $g_2^{k_0/\mathsf{uk}_i}$ and g_2^{1/uk_i} for every user $i \in G'$ from Ch_2. Choose $\mathsf{uk}_i \leftarrow \mathsf{MK.KeyGen}(1^\kappa)$ for all users $i \in G' - G$. For every edge $(i, 0)$, compute g_2^{1/uk_i}. Provide all this information to Adv_1.

2. Adv_2 gets w_0^* and w_1^* from Adv_1, forwards them to Ch_2 and returns Ch_2's answer.

3. Adaptive step: answer Adv_1's queries as follows:

    - Forward any encrypt query to Ch_2 and provide Ch_2's result to Adv_1.
    - Forward any token request for user $i \in G'$ to Ch_2 and return answer to Adv_1. Compute $H(w_\ell)^{\mathsf{uk}_i}$ for every user $i \in G - G'$ using the generated uk_i.

4. Adv_2 outputs Adv_1's decision.

We can see that since Adv_1 makes no token queries containing w_0^*, w_1^* for users with access to doc 0, Adv_2 will also satisfy the restriction in Game 2.

We can see that Adv_2 simulates Adv_1's inputs perfectly and when Adv_1 distinguishes, so does Adv_2; since Adv_1 wins in Game 1 with nonnegligible probability, Adv_2 also wins in Game 2 with the same probability, concluding the proof.

$\square$

We would like to simplify the game by only allowing encryption queries to w_0^* and w_1^*. Note that Adv_2 can compute by himself the result of any encrypt query for a word $w_\ell \notin \{w_0^*, w_1^*\}$ by simply requesting a token for w_ℓ for any user and using the delta information $g_2^{k_0/\mathsf{uk}_i}$. So it suffices to receive encryptions for the w_0^* and w_1^* only, as in the following game.

Game 3

- Adv_3 provides an access graph G with one document, labeled 0, and any number of users all with access to document 0.
- Ch_3 generates $k_0 \leftarrow \mathsf{MK.KeyGen}(1^\kappa, \mathsf{params})$ and provides $g_2^{k_0/\mathsf{uk}_i}$ and g_2^{1/uk_i} for every user i.
- Adv_3 provides w_0^*, w_1^*.
- Ch_3 chooses a random bit b and provides $r^*, H_2(r^*, e(H(w_\mathsf{b}^*), g_2)^{k_0})$.
- Adaptive step: Adv_3 makes the following queries to Ch_3 adaptively. The ℓ-th query can be
  - "Encrypt w_ℓ", for $w_\ell \in \{w_0^*, w_1^*\}$: Ch_3 returns $r_\ell, H_2(r_\ell, e(H(w_\ell), g_2)^{k_0})$
  - "Token for word $w_\ell \notin \{w_0^*, w_1^*\}$ for user i": Ch_3 returns $H(w_\ell)^{\mathsf{uk}_i}$.

Claim 2. *If a scheme is secure in Game 3, the scheme is secure in Game 2.*

Proof. For contradiction, assume there is a PPT adversary Adv_2 that can break Game 2, and let us show how to construct an PPT adversary Adv_3 that can break Game 3.

Let Ch_3 be the challenger of Adv_3 in Game 3. The idea is that Adv_3 will answer encrypt queries for word $w_\ell \notin \{w_0^*, w_1^*\}$ by asking for a token for w_ℓ and then computing the ciphertext, and for words w_0^* or w_1^*, by asking Ch_3 for encryptions. Adv_3 proceeds as follows.

1. Adv_3 receives the graph G from Adv_2. Adv_3 creates an additional user I with edge to document 0 and adds it to G. Adv_3 sends the new graph to Ch_3, records the answers from Ch_3 and returns all answers to Adv_2 except for $g_2^{k_0/\mathsf{uk}_I}$ and g_2^{1/uk_I}.
2. Challenge step: Adv_3 receives w_0^*, w_1^* from Adv_2 and provides them to Ch_3. Adv_3 forwards these to Ch_3 and receives $r^*, H_2(r^*, e(H(w_\mathsf{b}^*), g_2)^{k_0})$. Adv_3 sends all these values to Adv_2.
3. Adv_3 answers the queries of Adv_2 from the adaptive step as follows:
   - "Encrypt w_ℓ" : If $w_\ell \in \{w_0^*, w_1^*\}$, Adv_3 sends this query to Ch_3 and returns Ch_3's result. Else Adv_3 asks Ch_3 for "token w_ℓ user I", receives $H(w_\ell)^{\mathsf{uk}_I}$ and computes $r, H_2(r, e(H(w_\ell), g_2)^{k_0})$ for some $r \leftarrow \mathbb{Z}_p$ by using k_0/uk_I.
   - "Token w_ℓ for user i": forward this query to Ch_3 and send the response to Adv_2.
4. Adv_3 outputs Adv_2 decision.

We can see that Adv_3 plays the game with Ch_3 correctly because it never asks Ch_3 for encryption to words not in $\{w_0^*, w_1^*\}$. Moreover, Adv_3 simulates the inputs to Adv_2 exactly so Adv_3 also has a nonnegligible chance of deciding correctly equal to the one of Adv_2, which concludes the proof.

□

We now use the BDHV assumption to replace $e(H(w_b^*), g_2)$ with a random value R, which is desirable so that the adversary loses the information about b that $e(H(w_b^*), g_2)$ provides.

Game 4

- Adv_4 provides an access graph G with one document, labeled 0, and any number of users all with access to document 0.
- Ch_4 generates $k_0 \leftarrow \mathsf{MK.KeyGen}(1^\kappa, \mathsf{params})$ and provides $g_2^{k_0/\mathsf{uk}_i}$ and g_2^{1/uk_i} for every user i.
- Adv_4 provides w_0^*, w_1^*.
- Ch_4 chooses a random bit b and provides $r^*, H_2(r^*, R)$ for $R \leftarrow \mathbb{G}_T$.
- Adaptive step: Adv_4 makes the following queries to Ch_4 adaptively. The ℓ-th query can be:
  1. "Encrypt w_ℓ" for $w_\ell \in \{w_0^*, w_1^*\}$: If $w_\ell = w_\mathsf{b}^*$, Ch_4 returns r_ℓ and $H_2(r_\ell, R)$, else Ch_4 returns $r_\ell, H_2(r_\ell, R^\alpha)$, where α is such that $g_1^\alpha = H(w_{1-b}^*)/H(w_\mathsf{b}^*)$.
  2. "Token for word w_ℓ for user i" for $w_\ell \notin \{w_0, w_1\}$: Ch_4 returns $H(w_\ell)^{\mathsf{uk}_i}$.

Claim 3. *Assuming BDHV and that H is a programmable random oracle, Game 3 and Game 4 are computationally indistinguishable.*

Proof. For contradiction, we assume that there is a PPT adversary $\mathcal{D}$ that distinguishes the two games, and show how to construct a PPT reduction B that breaks BDHV.

B receives as input params, $g_1^a, g_2^b, g_2^{1/a}, g_1^c$ and T, where T is either $e(g_1, g_2)^{abc}$ or random. To distinguish what is T, B proceeds as follows.

B wants to embed some of the values from its challenge into the random oracle results when $\mathcal{D}$ queries for w_0^* or w_1^*. However, $\mathcal{D}$ could make queries to these values before declaring to B the values in the challenge step.

As a solution, B will guess which of the queries to the random oracle H are for challenge values. Without loss of generality, assume that $\mathcal{D}$ makes unique queries to H. We have three cases:

- B makes no query to the random oracle H including w_0^* or w_1^* before the challenge step.
- B queries exactly one of w_0^* and w_1^* to H before the challenge step.
- B queries both w_0^* and w_1^* to H before the challenge step.

Let i_0 be the guessed index of the query to H in which B requests w_0^*; i_0 could be $\perp$ if B does not request this value before the challenge step. Let p be a polynomial upper-bounding the runtime of $\mathcal{D}$ and hence the number of queries to H that $\mathcal{D}$ makes. B assigns a probability of $1/3$ to each case above and draws i_0, i_1 from $1, \dots, p(\kappa)$.

When $\mathcal{D}$ provides w_0^* and w_1^* to B in the challenge step, B can check whether it guessed i_0 and i_1 correctly. If it did not, B outputs a random guess in its game, and halts.

- *Initialization:* B generates params and sends them to $\mathcal{D}$. B chooses $\alpha \leftarrow \mathbb{Z}_p$.
- *H simulation:* Initialize oracle. For each query w of $\mathcal{D}$ to H, B does:
  - If this is the i_0-th query, return g_1^c.
  - If this is the i_1-th query, return $g_1^{c\alpha}$.
  - Otherwise, choose $q \leftarrow \mathbb{Z}_p$, store $\mathsf{oracle}[w] := q$ and return g_1^q.
- B receives a graph G from $\mathcal{D}$. For each user $i > 1$, let $\Delta_i \leftarrow \mathbb{Z}_p$ and let $\Delta_1 := 1$. Instead of $g_2^{k_0/\mathsf{uk}_i}$, provide g_2^{b/Δ_i}, and instead of g_2^{1/uk_i}, provide $g_2^{1/a}$ to $\mathcal{D}$.
- Challenge step: Receive w_0^* and w_1^* from $\mathcal{D}$. Validate whether i_0 and i_1 were correct guesses. If not, output a bit at random and halt. Else, provide r^*, $H_2(r^*, T)$ to $\mathcal{D}$.
- For each query of $\mathcal{D}$ during adaptive step:
  - For "encrypt w_ℓ": if $w_\ell = w_{\mathsf{b}}^*$, return r_ℓ and $H_2(r_\ell, T)$, else return r_ℓ, $H_2(r_\ell, T^\alpha)$.
  - For "Token w_ℓ user i": return $g_1^{a\Delta_i \mathsf{oracle}[w_\ell]}$.
- Output $\mathcal{D}$'s answer.

Let us argue that B simulates the inputs to $\mathcal{D}$ correctly. All the inputs to the random oracle H are correctly distributed, and the chance that c equals some value q drawn by B is statistically small.

B will have a chance of $1/\text{poly}$ of guessing correctly i_0 and i_1. Therefore, all we have to show is that when B guesses these values correctly, B has a nonnegligible chance of outputting b.

For this purpose, let us show that the inputs B provides to $\mathcal{D}$ are statistically close to the inputs from Game 3. Consider the following change of variables and note it preserves distributions:

$$a \leftrightarrow \mathsf{uk}_1, \quad b \leftrightarrow k_0/\mathsf{uk}_1, \quad g_1^c \leftrightarrow H(w_{\mathsf{b}}^*), \quad g_1^{c\alpha} \leftrightarrow H(w_{1-\mathsf{b}}^*), \quad \Delta_i \leftrightarrow \mathsf{uk}_i/\mathsf{uk}_1$$

B sends $\mathcal{D}$: $g_2^{1/a\Delta_i} = g_2^{1/\mathsf{uk}_i}$, $g_2^{b/\Delta_i} = g_2^{k_0/\mathsf{uk}_i}$.

For "encrypt" and the challenge step, note that if $T = e(g_1, g_2)^{abc}$ then $T = e(H(w_{\mathsf{b}}^*), g_2)^{k_0}$ as in Game 3, else T has the same distribution as R in Game 4.

For "token", $g_1^{a\Delta_i \times \mathsf{oracle}[w_\ell]} = H(w_\ell)^{\mathsf{uk}_i}$, as desired.

Finally, when $\mathcal{D}$ distinguishes Game 3 from Game 4, B also breaks the BDHV assumption, which completes the proof.

□

Note that in Game 4, all the information using uk_i, k_0 is useless to an adversary because the challenge ciphertexts do not depend on these values. Therefore, we can simplify further the game:

Game 5

- Adv_5 provides w_0^*, w_1^*.
- Ch_5 chooses a random bit b and provides r^*, $H_2(r^*, R)$ for $R \leftarrow \mathbb{G}_T$.

- Adv_5 can repeat the following query; query ℓ is "Encrypt w_ℓ" for $w_\ell \in \{w_0^*, w_1^*\}$: Ch_5 draws $r_\ell \leftarrow \mathbb{Z}_p$; if $w_\ell = w_{\mathrm{b}}^*$, Ch_5 returns $r_\ell, H_2(r_\ell, R)$, else Ch_5 returns $r_\ell, H_2(r_\ell, R^\alpha)$, where α is such that $g_1^\alpha = H(w_{1-\mathrm{b}}^*)/H(w_{\mathrm{b}}^*)$.

By the security of the random oracle H_2, no Adv can distinguish in Game 5 with non-negligible probability, concluding our proof.

□

◇ 5.8.2 Token hiding proof

Proof. We will create a set of hybrid games that progressively simplify the game until it becomes easy to show that Adv cannot learn b.

The first game, Game 1 is the same as the token hiding game except that it removes the encrypt queries. The intuition is that the output of the encrypt algorithm in our construction can be deduced from the outputs of the token and delta algorithms.

Game 1

1. Adv_1 provides G along with keys uk_i and k_j for free users and documents.
2. Ch_1 generates a new key for every non-free user i and document j using $\mathsf{MK.KeyGen}(1^\kappa)$. For every edge $(i, j) \in G$, Ch_1 sends $\mathsf{MK.Delta}(\mathsf{uk}_i, k_j)$ to Adv_1.
3. Adaptive step: Adv_1's ℓ-th query is "Token w_ℓ for user i" and Adv_1 receives $\mathsf{MK.Token}(\mathsf{uk}_i, w)$.
4. Adv_1 provides w_0^* and w_1^* to Ch_1. Ch_1 chooses $b \leftarrow \{0, 1\}$ and sends $\mathsf{MK.Token}(\mathsf{uk}_0, w_b^*)$ to Adv_1.
5. Adv_1 runs the adaptive step again.

Restriction on Adv_1: For every "Token w_ℓ for user i" query: $w_\ell \notin \{w_0^*, w_1^*\}$ or user i is free.

Claim 4. *If a scheme is secure with Game 1, the scheme must be token hiding.*

Proof. For contradiction, assume there is a PPT adversary Adv that wins the token hiding game, and let us construct a PPT adversary Adv_1 that wins Game 1. Let Ch_1 be the challenger in Game 1. Adv_1 uses Adv as follows.

- On input a graph G and keys from Adv, Adv_1 simply forwards these to Ch_1. Adv_1 forwards the responses from Ch_1 to Adv and records these as well.
- Adaptive step: For "Token" queries, Adv_1 sends the same queries to Ch_1 and forwards the responses to Adv.

For a query "Encrypt w_ℓ for document j", Adv_1 proceeds as follows. If document j is free, Adv_1 knows k_j from Adv so it simply computes $\mathsf{MK.Enc}(k_j, w_\ell)$. If document j is non-free, Adv_1 must have a delta between user 0 and document j, say $\Delta_{0,j}$. Adv_1 requests "Token w_ℓ for document 0" to Ch_1, which is a valid request because $w_\ell \notin \{w_{0,\alpha}, w_{1,\alpha}\}_\alpha$ because of the constraints on Adv. Upon receiving token Token back, Adv_1 sends $r, H_2(r, e(t, \Delta_{0,j}))$ to Adv.

- Adv_1 forwards the challenges from Adv to Ch_1 and sends Ch_1's answer to Adv_1.
- Adv_1 proceeds as above in the second adaptive step.
- Adv_1 ouputs Adv's answer.

We can see that Adv_1 simulates Adv's inputs perfectly. Since Adv wins in the token hiding game with non-negligible probability, so will Adv_1 win in Game 1.

□

To simplify the game further, we would like to remove the free documents and the free users from the game, and only work with non-free users and documents.

Game 2

1. Adv_2 provides a graph G that has only non-free documents and users.
2. Ch_2 generates a new key for every user i and document j using $\mathsf{MK.KeyGen}(1^\kappa)$. For every edge $(i, j) \in G$, Ch_2 provides $g_2^{k_j/\mathsf{uk}_i}$ to Adv_2. For every user $i > 0$, Ch_2 provides g_2^{1/uk_i}.
3. Adaptive step: Adv_2's ℓ-th query can be "Token w_ℓ for user i", in which case it receives $H(w_\ell)^{\mathsf{uk}_i}$ from Ch_2.
4. Adv_2 provides w_0^* and w_1^* to Ch_2. Ch_2 chooses b at random and provides $H(w_\mathsf{b}^*)^{\mathsf{uk}_0}$ to Adv_2.
5. Adv_2 runs the adaptive step again.

Restriction on Adv_2: $w_\ell \notin \{w_0^*, w_1^*\}$, for all ℓ.

Claim 5. *If a scheme is secure with Game 2, the scheme is secure with Game 1.*

Proof. For contradiction, assuming there is a PPT adversary Adv_1 for Game 1, let us show how to construct a PPT reduction Adv_2 that wins in Game 2. Let Ch_2 be the challenger of Adv_2 in Game 2. Adv_2 works as follows:

- Receive G from Adv_1 along with uk_i and k_j for all free nodes. Remove from G all free nodes and thus obtain a new graph G'. Send G' to Ch_2. Store uk_i, k_j.

- Ch_2 replies with $g_2^{k_j/\mathsf{uk}_i}$ and g_2^{1/uk_i} corresponding to non-free nodes. Adv_2 needs to compute all deltas for G for Adv_1. For an edge between two free nodes, Adv_2 has both keys so it can directly compute the delta. For an edge between two non-free nodes, Adv_2 got $g_2^{k_j/\mathsf{uk}_i}$ from Adv_1. For an edge between a non-free user i and a free document j, Adv_2 knows g_2^{1/uk_i} and k_j so it can compute delta $g_2^{k_j/\mathsf{uk}_i}$. To provide g_2^{1/uk_i} to Adv_1, Adv_2 either uses its knowledge of uk_i for free users or receives this value from Ch_2.

- Adv_2 now answers Adv_1's queries, which are of the form "Token w_ℓ for user i". We have two cases. If i is a free user, Adv_2 can directly compute the token using uk_i. If i is non-free, i can ask Ch_2 for the token and forward it to Adv_2.

We can see that Adv_2 still satisfies the constraints of its game and simulates the inputs to Adv_1 perfectly. Moreover, whenever Adv_1 wins, Adv_2 wins as well.

□

We now write a final hybrid in which $H(w_{b,\ell})^{\mathsf{uk}_0}$ is a random value preserving the equality relations of $w_{b,\ell}$. Claim 6 shows that Game 2 and Game 3 are computationally indistinguishable.

Game 3

1. Adv_3 provides G with only non-free documents and users.
2. Ch_3 generates a new key for every user i and document j using $\mathsf{MK.KeyGen}(1^\kappa)$. For every edge $(i, j) \in G$, Ch_3 provides $g_2^{k_j/\mathsf{uk}_i}$ to Adv_3. For every user $i > 0$, Ch_3 provides g_2^{1/uk_i}.
3. Adaptive step: Adv_3 queries ℓ-th query: "Token w_ℓ for user i" and receives $H(w_\ell)^{\mathsf{uk}_i}$.
4. Adv_3 provides w_0^* and w_1^* to Ch_3. Ch_3 chooses b at random and sends R, for $R \leftarrow \mathbb{G}_T$.
5. Adv_3 runs the adaptive step again.

Restriction on Adv_3: $w_\ell \notin \{w_0^*, w_1^*\}$, for all ℓ.

We can see that in this game Adv_3 receives no information about b information theoretically. The chance Adv_3 has to guess b is exactly $1/2$, which completes our proof.

Claim 6. *Assuming XDHV holds, Game 2 is computationally indistinguishable from Game 3, in the random oracle model for H.*

Proof. For contradiction, assume that there is a PPT adversary $\mathcal{D}$ that can distinguish the two games (i.e., distinguish between Ch_2 and Ch_3). Let us construct a PPT adversary B that can break the XDHV assumption.

Let p be a polynomial in which $\mathcal{D}$ runs. As in the proof of data hiding, B wants to embed a ciphertext from its challenge, g_1^b into the oracle result to $\mathcal{D}$, when $\mathcal{D}$ queries for the challenge ciphertext w_{b}^*. However, $\mathcal{D}$ can query w_{b}^* to H before the challenge step, so before B knows the value of w_{b}^*. Therefore, B will guess which of the queries to the random oracle H are w_{b}^*. If during the challenge step the guess turns out to be incorrect, B outputs a bit at random and halts. Otherwise B proceeds.

B receives as input $g_1^a, g_1^b, T, g_2^{ca}, g_2^{cd}, g_1^d, g_2^{1/d}$ and must decide if $T = g_1^{ab}$ or T is random.

- *H simulation:* B flips a coin, and if the coin is heads, B predicts that $\mathcal{D}$ will never ask for w_b^* to the random oracle; otherwise, B predicts that $\mathcal{D}$ with ask for w_b^* and chooses an index at random $I \in \{0, \ldots, p(\kappa)\}$ to represent the index of the query during which $\mathcal{D}$ will ask for w_b^*. For each query w of $\mathcal{D}$ to H, B does:
  - If this is the I-th query, return g_1^b.
  - Otherwise, choose $q \leftarrow \mathbb{Z}_p$, store $\mathsf{oracle}[w] := q$ and return g_1^q.
- *Initialization.* B starts adversary $\mathcal{D}$ and receives a graph G. B provides the following information for the graph. For each document j, let $\alpha_j \leftarrow \mathbb{Z}_p$ if $j > 1$ and let $\alpha_1 := 1$ for $j = 1$. For each user i, let $\Delta_i \leftarrow \mathbb{Z}_p$ if $i > 1$, and let $\Delta_1 := 1$ if $i = 1$.
  - $g_2^{k_j/\mathsf{uk}_0} := g_2^{dc\alpha_j}$, for $j \geq 1$.
  - $g_2^{k_j/\mathsf{uk}_i} := g_2^{ac\alpha_j/\Delta_i}$.
  - $g_2^{1/\mathsf{uk}_i} := g_2^{1/d\Delta_i}$.
- Adaptive step: If user is 0, B outputs $g_1^{a\mathsf{oracle}[w]}$. Otherwise, user $i > 0$, and B outputs $g_1^{d\Delta_i\mathsf{oracle}[w]}$. Note that it is crucial that $w \neq w_\mathsf{b}^*$ because B would not know $\mathsf{oracle}[w_\mathsf{b}^*]$ (which should be b).
- B receives w_0^* and w_1^*. B checks if w_b^* is indeed the I-th element $\mathcal{D}$ queried to H, B's guess. If not, B outputs a random bit and halts. Otherwise, B sends T to $\mathcal{D}$.
- B proceeds as in the adaptive step.
- B outputs $\mathcal{D}$'s decision.

Let us argue that B simulates the inputs to $\mathcal{D}$ correctly, whenever B does not halt early. All the inputs to the random oracle H are uniformly random distributed, and the chance that a equals some value q drawn by B is statistically small.

Consider the following change of variables and note that it preserves distributions:

$$a \leftrightarrow \mathsf{uk}_0,\ \ g_1^b \leftrightarrow H(w_\mathsf{b}^*),\ \ c \leftrightarrow k_1/\mathsf{uk}_1\mathsf{uk}_0,\ \ d \leftrightarrow \mathsf{uk}_1,\ \ \Delta_i \leftrightarrow \mathsf{uk}_i/\mathsf{uk}_1,\ \ \alpha_j \leftrightarrow k_j/k_1.$$

The quantities $\mathcal{D}$ receives are:

- For $g_2^{k_j/\mathsf{uk}_0}$ with $j \geq 1$: $g_2^{dc\alpha_j} = g_2^{\mathsf{uk}_1 \frac{k_1}{\mathsf{uk}_0\mathsf{uk}_1} \frac{k_j}{k_1}} = g_2^{k_j,\mathsf{uk}_0}$, as desired.
- For $g_2^{k_j/\mathsf{uk}_i}$: $g_2^{ac\alpha_j/\Delta_i} = g_2^{\frac{k_1}{\mathsf{uk}_1}\frac{\alpha_j}{\Delta_i}} = g_2^{k_j/\mathsf{uk}_i}$, as desired,

- For g_2^{1/uk_i}: $g_2^{1/d\Delta_i} = g_2^{\frac{1}{\mathsf{uk}_1 \Delta_i}} = g_2^{1/\mathsf{uk}_i}$, as desired.
- Adaptive step: $g_1^{a\mathsf{oracle}[w_\ell]} = H(w_\ell)^{\mathsf{uk}_0}$, and $g_1^{d\Delta_i \mathsf{oracle}[w]} = H(w)^{\mathsf{uk}_i}$ as desired.
- If T is random, the challenge step is as in Game 3. When $T = g_1^{ab} = H(w_{\mathsf{b}}^*)^{\mathsf{uk}_0}$, the challenge step is as in Game 2.

We can see that B simulates the inputs to $\mathcal{D}$ statistically close, whenever B does not halt early. Since $\mathcal{D}$ has a nonnegligible chance of distinguishing Game 3 from Game 2, when B does not halt, B also has a non-negligible chance of breaking the the XDHV assumption. The chance that B does not halt is at least $1/2p$, so the overall advantage of B remains non-negligible.

□

□

CHAPTER 6

Functional encryption

In this chapter, we present one [GKP+13a] of our two functional encryption schemes [GKP+13a] and [GKP+13b]. This scheme enables computing any function on encrypted data. We present the scheme briefly, and delegate formal proofs to our extended paper [GKP+12].

□ 6.1 Introduction

This thesis discusses the importance of computing on encrypted data. A fundamental question with this approach is: *who can decrypt the results of computations on encrypted data?* If data is encrypted using FHE, anyone can perform a computation on it (with knowledge of the public key), while the result of the computation can be decrypted only using the secret key. However, the secret key allows decrypting *all* data encrypted under the corresponding public key. This model suffices for certain applications, but it rules out a large class of applications in which the party computing on the encrypted data needs to determine the computation result on its own. For example, spam filters should be able to determine if an encrypted email is spam and discard it, without learning anything else about the email's content. With FHE, the spam filter can run the spam detection algorithm homomorphically on an encrypted email and obtain an encrypted result; however, it cannot tell if the algorithm deems the email spam or not. Having the data owner provide the decryption key to the spam filter is not a solution: the spam filter can now decrypt all the emails as well!

A promising approach to this problem is *functional encryption* [SW05, GPSW06, KSW08, LOS+10, OT10, O'N10, BSW11]. In functional encryption, anyone can encrypt data with a master public key mpk and the holder of the master secret key can provide keys for functions, for example sk_f for function f. Anyone with access to a key sk_f and a ciphertext c for x can obtain the result of the computation in plaintext form: $f(x)$. The security of FE requires that the adversary does not learn anything about x, other than the computation result $f(x)$. It is easy to see, for example, how to solve the above spam filter problem with a functional encryption scheme. A user Alice publishes her public key online and gives the spam filter a key for the filtering function. Users sending email to Alice will encrypt the email with her public key. The spam filter can now determine by itself, for each email, whether to store it in Alice's mailbox or to discard it as spam, without learning anything about Alice's email (except for whether it was deemed spam or not).

The recent impossibility result of Agrawal, Gorbunov, Vaikuntanathan and Wee [AGVW12] says that functional encryption schemes where an adversary can receive an arbitrary number of keys for general functions are impossible for a natural simulation-based security definition;[1] stated differently, any functional encryption scheme that can securely provide q keys for general functions must have ciphertexts growing linearly in q. Since any scheme that can securely provide a single key yields a scheme that can securely provide q keys by repetition, the question becomes if one can construct a functional encryption scheme that can securely provide a *single key for a general function* under this simulation-based security definition. Such a single-key functional encryption scheme is a powerful tool, enabling the applications we will discuss.

In this chapter, we construct the first single-key functional encryption scheme for a *general* function that is *succinct*: the size of the ciphertext grows with the depth d of the circuit computing the function and is independent of the size of the circuit. Up until our work, the known constructions of functional encryption were quite limited. First, the works of Boneh and Waters [BW07], Katz, Sahai and Waters [KSW08], Agrawal, Freeman and Vaikuntanathan [AFV11], and Shen, Shi and Waters [SSW09] show functional encryption schemes (based on different assumptions) for a very simple function: the inner product function f_y (or a variant of it), that on input x outputs 1 if and only if $\langle x, y \rangle = 0$.[2] These works do not shed light on how to extend beyond inner products. Second, Sahai and Seyalioglu [SS10a] and Gorbunov, Vaikuntanathan and Wee [GVW12] provide a construction for single-key functional encryption for one general function with a *non-succinct* ciphertext size (at least the size of a universal circuit computing the functions allowed by the scheme[3]). [SS10a] was the first to introduce the idea of single-key functional encryption and [GVW12] also extends it to allow the adversary to see secret keys for q functions of his choice, by increasing the size of the ciphertexts linearly with q where q is known in advance.[4] We emphasize that the non-succinctness of these schemes is particularly undesirable and it precludes many useful applications of functional encryption (e.g., delegation, reusable garbled circuits, FHE for Turing machines), which we achieve. For example, in the setting of delegation, a data owner wants to delegate her computation to a cloud, but the mere effort of encrypting the data is greater than computing the circuit directly, so the owner is better off doing the computation herself.

We remark that functional encryption (FE) arises from, and generalizes, a beautiful sequence of papers on attribute-based encryption (including [SW05, GPSW06, BSW07, GJPS08, LOS$^+$10, Wat11, Wat12, LW12a]), and more generally predicate encryption (including [BW07, KSW08, OT09]). We denote by attribute-based encryption (ABE) an encryption scheme where each ciphertext c of an underlying plaintext message m is tagged with a public attribute x. Each secret key sk_f is associated with a predicate f. Given a key sk_f and a ciphertext $c = \mathsf{enc}(x, m)$, the message m can be recovered if and only if $f(x)$ is true. Whether the message gets recovered or not, the attribute x is always public; in other words, the input to the computation of f, x, leaks

[1]This impossibility result holds for non-adaptive simulation-based security, which is weaker than some existing simulation-based definitions such as adaptive security. Nevertheless, this result does not carry over to indistinguishability-based definitions, for which possibility or impossibility is currently an open question. In this chapter, we are interested in achieving the simulation-based definition.

[2]These inner-product schemes allow an arbitrary number of keys.

[3]A universal circuit $\mathcal{F}$ is a circuit that takes as input a description of a circuit f and an input string x, runs f on x and outputs $f(x)$.

[4]Namely, parameter q (the maximum number of keys allowed) is fixed during setup, and the ciphertexts size grows linearly with q.

with attribute-based encryption, whereas with functional encryption, nothing leaks about x other than $f(x)$. Therefore, attribute-based encryption offers qualitatively weaker security than functional encryption. Attribute-based encryption schemes were also called public-index predicate encryption schemes in the literature [BSW11]. Boneh and Waters [BW07] introduced the idea of not leaking the attribute as in functional encryption (also called private-index functional encryption).

Very recently, the landscape of attribute-based encryption has significantly improved with the works of Gorbunov, Vaikuntanathan and Wee [GVW13], and Sahai and Waters [SW12], who construct attribute-based encryption schemes for general functions, and are a building block for our results.

◇ 6.1.1 Our Results

Our main result is the construction of a *succinct* single-key functional encryption scheme for *general functions*. We demonstrate the power of this result by showing that it can be used to address the long-standing open problem in cryptography of reusing garbled circuits, as well as making progress on other open problems.

We can state our main result as a reduction from *any* attribute-based encryption and *any* fully homomorphic encryption scheme. In particular, we show how to construct a (single-key and succinct) functional encryption scheme for any class of functions $\mathcal{F}$ by using a homomorphic encryption scheme which can do homomorphic evaluations for any function in $\mathcal{F}$ and an attribute-based encryption scheme for a "slightly larger" class of functions $\mathcal{F}'$; $\mathcal{F}'$ is the class of functions such that for any function $f \in \mathcal{F}$, the class $\mathcal{F}'$ contains the function computing the i-th bit of the FHE evaluation of f.

Theorem 4. *There is a single-key functional encryption scheme with succinct ciphertexts (independent of circuit size) for the class of functions $\mathcal{F}$ assuming the existence of*

- *a fully homomorphic encryption scheme for the class of functions $\mathcal{F}$, and*
- *a (single-key) attribute-based encryption scheme for a class of predicates $\mathcal{F}'$ (as above).*

The literature has considered two types of security for ABE and FE: selective and full security . We show that if the underlying ABE scheme is selectively or fully secure, our resulting FE scheme is selectively or fully secure, respectively.

Two very recent results achieve attribute-based encryption for general functions. Gorbunov, Vaikuntanathan and Wee [GVW13] achieve ABE for general circuits of bounded depth based on the subexponential Learning With Errors (LWE) intractability assumption. Sahai and Waters [SW12] achieve ABE for general circuits under the less standard k-Multilinear Decisional Diffie-Hellman (see [SW12] for more details); however, when instantiated with the only construction of multilinear maps currently known [GGH12], they also achieve ABE for general circuits of bounded depth. Our scheme can be instantiated with any of these schemes because our result is a reduction.

When coupling our theorem with the ABE result of [GVW13] and the FHE scheme of [BV11a, BGV12], we obtain:

Corollary 2 (Informal). *Under the subexponential LWE assumption, for any depth d, there is a single-key functional encryption scheme for general functions computable by circuits of depth d. The scheme has succinct ciphertexts: their size is polynomial in the depth d (and does not depend on the circuit size).*

This corollary holds for both selective and full security definitions, since [GVW13] constructs both selectively secure and fully secure ABE schemes. However, the parameters of the LWE assumption are different in the two cases. For selective security, the LWE assumption reduces to the (polynomial) hardness of approximating shortest vectors in a lattice up to sub-exponential approximation factors. This assumption is known as the gapSVP assumption with sub-exponential approximation factors. For full security, the LWE assumption reduces to the same assumption as above, but where the hardness is assumed to hold even against sub-exponential time adversaries. Namely, the assumption is that it is hard to approximate shortest vectors in a lattice up to sub-exponential approximation factors in sub-exponential time. Both of these assumptions are quite standard and well-studied assumptions that are believed to be true. (We refer the reader to our full paper [GKP$^+$12] for details.)

Another corollary of our theorem is that, given a *universal* ABE scheme (the scheme is for all classes of circuits, independent of depth) and any fully homomorphic encryption scheme, there is a universal functional encryption scheme whose ciphertext size does not depend on the circuit's size or even the circuit's depth.

As mentioned, extending our scheme to be secure against an adversary who receives q keys is straightforward. The basic idea is simply to repeat the scheme q times in parallel. This strategy results in the ciphertext size growing linearly with q, which is unavoidable for the simulation-based security definition we consider, because of the discussed impossibility result [AGVW12]. Stated in these terms, our scheme is also a q-collusion-resistant functional encryption scheme like [GVW12], but our scheme's ciphertexts are succinct, whereas [GVW12]'s are proportional to the circuit size.

From now on, we restrict our attention to the single-key case, which is the essence of the new scheme. In the body of the chapter we often omit the single-key or succinct adjectives and whenever we refer to a functional encryption scheme, we mean a succinct single-key functional encryption scheme.

We next show how to use our main theorem to make significant progress on some of the most intriguing open questions in cryptography today: the *reusability* of garbled circuits, a new paradigm for *general function obfuscation*, as well as applications to fully homomorphic encryption with evaluation running in *input-specific time* rather than in worst-case time, and to publicly verifiable delegation. Succinctness plays a central role in these applications and they would not be possible without it.

Main Application: Reusable Garbled Circuits

A circuit garbling scheme, which has been one of the most useful primitives in modern cryptography, is a construction originally suggested by Yao in the 80s in the context of secure two-party computation [Yao82]. This construction relies on the existence of a one-way function to encode an arbitrary circuit C ("garbling" the circuit) and then encode any input x to the circuit (where the size of the encoding is short, namely, it does not grow with the size of the circuit C); a party given the garbling of C and the encoding of x can run the garbled circuit on the encoded x and obtain $C(x)$. The most basic properties of garbled circuits are circuit and input privacy: an adversary learns nothing about the circuit C or the input x other than the result $C(x)$.

Over the years, garbled circuits and variants thereof have found many applications: two party secure protocols [Yao86], multi-party secure protocols [GMW87], one-time programs [GKR08], KDM-security [BHHI10], verifiable computation [GGP10], homomorphic computations [GHV10] and others. However, a basic limitation of the original construction remains: it offers only *one-time* usage. Specifically, providing an encoding of more than one input compromises the secrecy of the circuit. Thus, evaluating the circuit C on any new input requires an entirely new garbling of the circuit.

The problem of reusing garbled circuits has been open for 30 years. Using our newly constructed succinct functional encryption scheme we are now able to build *reusable garbled circuits* that achieve *circuit and input privacy*: a garbled circuit for any computation of depth d (where the parameters of the scheme depend on d), which can be run on *any polynomial number* of inputs without compromising the privacy of the circuit or the input. More generally, we prove the following:

Theorem 5 (Informal)**.** *There exists a polynomial p, such that for any depth function d, there is a reusable circuit garbling scheme for the class of all arithmetic circuits of depth d, assuming there is a single-key functional encryption scheme for all arithmetic circuits of depth $p(d)$.*[5]

Corollary 3 (Informal)**.** *Under the subexponential LWE assumption, for any depth function d, there exists a reusable circuit garbling scheme with circuit and input privacy for all arithmetic circuits of depth d.*

We note that the parameters of this LWE assumption imply its reducibility to the assumption that gapSVP is hard to break in sub-exponential time with sub-exponential approximation factors. (We refer the reader to our full paper [GKP+12] for details.)

Reusability of garbled circuits (for depth-bounded computations) implies a multitude of applications as evidenced by the research on garbled circuits over the last 30 years. We note that for many of these applications, depth-bounded computation suffices. We also note that some applications do not require circuit privacy. In that situation, our succinct single-key functional encryption scheme already provides reusable garbled circuits with input-privacy and, moreover, the encoding of the input is a public-key algorithm.

We remark that [GVW13] gives a restricted form of reusable circuit garbling: it provides authenticity of the circuit output, but does not provide input privacy or circuit privacy, as we do here. Informally, authenticity means that an adversary cannot obtain a different yet legitimate result from a garbled circuit. We note that most of the original garbling circuit applications (e.g., two party secure protocols [Yao86], multi-party secure protocols [GMW87]) rely on the privacy of the input or of the circuit.

One of the more intriguing applications of reusable garbled circuits pertains to a new model for program obfuscation, token-based obfuscation, which we discuss next.

Token-Based Obfuscation: a New Way to Circumvent Obfuscation Impossibility Results

Program obfuscation is the process of taking a program as input, and producing a functionally equivalent but different program, so that the new program reveals no information to a computationally bounded adversary about the original program, beyond what "black box access" to the program reveals. Whereas ad-hoc program obfuscators are built routinely, and are used in practice as the main software-based technique to fight reverse engineering of programs, in 2000 Barak et al. [BGI+01],

[5]For this application we need to assume that the underlying functional encryption scheme is fully secure (as opposed to only selectively secure).

followed by Goldwasser and Kalai [GK05], proved that program obfuscation for general functions is impossible using software alone, with respect to several strong but natural definitions of obfuscation.

The results of [BGI+01, GK05] mean that there exist functions which cannot be obfuscated. Still, the need to obfuscate or "garble" programs remains. A long array of works attempts to circumvent the impossibility results in various ways, including adding secure hardware components [GKR08, GIS+10, BCG+11], relaxing the definition of security [GR07], or considering only specific functions [Wee05, CKVW10].

The problem of obfuscation seems intimately related to the "garbled circuit" problem where given a garbling of a circuit C and an encoding for an input x, one can learn the result of $C(x)$ but nothing else. One cannot help but wonder whether the new reusable garbling scheme would *immediately* imply a solution for the obfuscation problem (which we know is impossible). Consider an example illustrating this intuition: a vendor obfuscates her program (circuit) by garbling it and then gives the garbled circuit to a customer. In order to run the program on (multiple) inputs x_i, the customer simply encodes the inputs according to the garbling scheme and thus is able to compute $C(x_i)$. Unfortunately, although close, this scenario does not work with reusable garbled circuits. The key observation is that encoding x requires knowledge of a secret key! Thus, an adversary cannot produce encoded inputs on its own, and needs to obtain "tokens" in the form of encrypted inputs from the data owner.

Instead, we propose a new *token-based* model for obfuscation. The idea is for a vendor to obfuscate an arbitrary program as well as provide tokens representing rights to run this program on specific inputs. For example, consider that some researchers want to obtain statistics out of an obfuscated database containing sensitive information (the obfuscated program is the program running queries with the secret database hardcoded in it). Whenever the researchers want to input a query x to this program, they need to obtain a token for x from the program owner. To produce each token, the program owner does little work. The researchers perform the bulk of the computation by themselves using the token and obtain the computation result without further interaction with the owner.

Claim 7. *Assuming a reusable garbling scheme for a class of circuits, there is a token-based obfuscation scheme for the same class of circuits.*

Corollary 4 (Informal). *Under the subexponential LWE assumption, for any depth function d, there exists a token-based obfuscation scheme for all arithmetic circuits of depth d.*

It is worthwhile to compare the token-based obfuscation model with previous work addressing obfuscation using trusted-hardware components such as [GIS+10, BCG+11]. In these schemes, after a user finishes executing the obfuscated program on an input, the user needs to interact with the trusted hardware to obtain the decryption of the result; in comparison, in our scheme, the user needs to obtain only a token before the computation begins, and can then run the computation and obtain the decrypted result by herself.

Computing on Encrypted Data in Input-Specific Time

All current FHE constructions work according to the following template. For a fixed input size, a program is transformed into an arithmetic circuit; homomorphic evaluation happens gate by gate on this circuit. The size of the circuit reflects the *worst-case* running time of the program: for example, every loop is unfolded into the maximum number of steps corresponding to the worst-case input, and each function is called the maximum number of times possible. Such a circuit can be potentially very large, despite the fact that there could be many inputs on which the execution is short.

A fascinating open question has been whether it is possible to perform FHE following a Turing-machine-like template: *the computation time is input-specific* and can terminate earlier depending on the input at hand. Of course, to compute in input-specific time, the running time must unavoidably leak to the evaluator, but such leakage is acceptable in certain applications and the efficiency gains can be significant; therefore, such a scheme provides weaker security than fully homomorphic encryption (namely, nothing other than the running time leaks about the input), at the increase of efficiency.

Using our functional encryption scheme, we show how to achieve this goal. The idea is to use the scheme to test when an encrypted circuit computation has terminated, so the computation can stop earlier on certain inputs. We overview our technique in Sec. 6.1.2.

Because the ciphertexts in our functional encryption scheme grow with the depth of the circuits, such a scheme is useful only for Turing machines that can be expressed as circuits of depth at most $d(n)$ for inputs of size n. We refer to such Turing machines as *d-depth-bounded* (see our full paper [GKP$^+$12] for details).

Theorem 6. *There is a scheme for evaluating Turing machines on encrypted inputs in input-specific time for any class of d-depth-bounded Turing machines, assuming the existence of a succinct single-key functional encryption scheme for circuits of depth d,*[6] *and a fully homomorphic encryption scheme for circuits of depth d.*

Corollary 5 (Informal)**.** *Under the subexponential LWE assumption, for any depth d, there is a scheme for evaluating Turing machines on encrypted data in input-specific time for any class of d-depth-bounded Turing machines.*

The parameters of this LWE assumption are the same as discussed in Corollary 2.

Publicly Verifiable Delegation with Secrecy

Recently, Parno, Raykova and Vaikuntanathan [PRV12] showed how to construct a 2-message delegation scheme that is *publicly verifiable*, in the preprocessing model, from any attribute-based encryption scheme. This reduction can be combined with [GVW13]'s ABE scheme to achieve such a delegation scheme.

However, this scheme does not provide *secrecy* of the inputs: the prover can learn the inputs. By replacing the ABE scheme in the construction of [PRV12] with our new functional encryption scheme, we add secrecy to the scheme; namely, we obtain a delegation scheme which is both *publicly verifiable* as in [PRV12] (anyone can verify that a transcript is accepting using only public information) and *secret* (the prover does not learn anything about the input of the function being delegated).[7] More specifically, we construct a 2-message delegation scheme in the preprocessing model that is based on the subexponential LWE assumption, and is for general depth-bounded circuits, where the verifier works in time that depends on the *depth* of the circuit being delegated, but

[6]As in previous applications, we need to assume that the underlying functional encryption scheme is fully secure (as opposed to only selectively secure).

[7]We note that secrecy can be easily obtained by using an FHE scheme, however, this destroys public-verifiability.

is independent of the size of the circuit, and the prover works in time dependent on the *size* of the circuit. For more details, see our full paper [GKP+12].

◇ 6.1.2 Technique Outline

Our functional encryption scheme. We first describe the ideas behind our main technical result: a reduction from attribute-based encryption (ABE) and fully homomorphic encryption (FHE) to functional encryption (FE).

Compute on encrypted data with FHE. A natural starting point is FHE because it enables computation on encrypted data, which is needed with functional encryption. Using FHE, the FE encryption of an input x consists of an FHE encryption of x, denoted $\hat{x}$, while the secret key for a function f is simply f itself. The semantic security of FHE provides the desired security (and more) because nothing leaks about x; however, using FHE evaluation, the evaluator obtains an encrypted computation result, $\widehat{f(x)}$, instead of the decrypted value $f(x)$. Giving the evaluator the FHE decryption key is not an option because the evaluator can use it to decrypt x as well.

Attempt to decrypt using a Yao garbled circuit. We would like the evaluator to decrypt the FHE ciphertext $\widehat{f(x)}$, but not be able to decrypt anything else. An idea is for the owner to give the evaluator a Yao garbled circuit for the FHE decryption function $\mathsf{FHE.Dec}$ with the FHE secret key hsk hardcoded in it, namely a garbled circuit for $\mathsf{FHE.Dec}_{\mathsf{hsk}}$. When the owner garbles $\mathsf{FHE.Dec}_{\mathsf{hsk}}$, the owner also obtains a set of garbled circuit labels $\{L_0^i, L_1^i\}_i$. The evaluator must only receive the input labels corresponding to $\widehat{f(x)}$: namely, the labels $\{L_{b_i}^i\}_i$ where b_i is the i-th bit of $\widehat{f(x)}$. But this is not possible because the owner does not know a priori $\widehat{f(x)}$ which is determined only after the FHE evaluation; furthermore, after providing more than one set of labels (which happens when encrypting another input x'), the security of the garbled circuit (and hence of the FHE secret key) is compromised. One idea is to have the owner and the evaluator interact, but the syntax of functional encryption does not allow interaction. Therefore, the evaluator needs to determine the set of labels corresponding to $\widehat{f(x)}$ *by herself*, and *should not obtain any other labels*.

Constraining decryption using ABE. It turns out that what we need here is very close to what ABE provides. Consider the following variant of ABE that can be constructed easily from a standard ABE scheme. One encrypts a value y together with two messages m_0, m_1 and obtains a ciphertext $c \leftarrow \mathsf{ABE.Enc}(y, m_0, m_1)$. Then, one generates a key for a predicate g: $\mathsf{sk}_g \leftarrow \mathsf{ABE.KeyGen}(g)$. The decryption algorithm on input c and sk_g outputs m_0 if $g(y) = 0$ or outputs m_1 if $g(y) = 1$.

Now consider using this ABE variant multiple times, once for every $i \in \{1, \ldots, \text{size of } \widehat{f(x)}\}$. For the i-th invocation of $\mathsf{ABE.Enc}$, let m_0, m_1 be the garbled labels L_0^i, L_1^i, and let y be $\hat{x}$: $\mathsf{ABE.Enc}(\hat{x}, L_0^i, L_1^i)$. Next, for the i-th invocation of $\mathsf{ABE.KeyGen}$, let g be $\mathsf{FHE.Eval}_f^i$ (the predicate returning the i-th bit of the evaluation of f on an input ciphertext): $\mathsf{ABE.KeyGen}(\mathsf{FHE.Eval}_f^i)$. Then, the evaluator can use $\mathsf{ABE.Dec}$ to obtain the needed label: $L_{b_i}^i$ where b_i is the i-th bit of $\widehat{f(x)}$. Armed with these labels and the garbled circuit, the evaluator decrypts $f(x)$.

The security of the ABE scheme ensures the evaluator cannot decrypt any other labels, so the evaluator cannot learn more than $f(x)$. Finally, note that the one-time aspect of garbled circuits does not restrict the number of encryptions with our FE scheme because the encryption algorithm generates a new garbled circuit every time; since the garbled circuit is for the FHE decryption algorithm (which is a fixed algorithm), the size of the ciphertexts remains independent of the size of f.

We now explain how to use this result to obtain the aforementioned applications.

From FE to reusable garbled circuits. The goal of garbled circuits is to hide the input and the circuit C. Our succinct single-key FE already provides a reusable garbling scheme with input privacy (the single key corresponds to the circuit to garble). To obtain circuit privacy, the insight is to leverage the secrecy of the inputs to hide the circuit. The first idea that comes to mind is to generate a key for the universal circuit instead of C, and include C in the ciphertext when encrypting an input. However, this approach will yield large ciphertexts, as large as the circuit size.

Instead, the insight is to garble C by using a semantically secure encryption scheme $\mathsf{E.Enc}$ together with our FE scheme: the garbling of C will be an FE secret key for a circuit U that contains $\mathsf{E.Enc}_{\mathsf{sk}}(C)$; on input (sk, x), U uses sk to decrypt C and then runs C on the input x. The token for an input x will be an FE encryption of (sk, x). Now, even if the FE scheme does not hide $\mathsf{E.Enc}_{\mathsf{sk}}(C)$, the security of the encryption scheme E hides C.

Computing on encrypted data in input-specific time. We now summarize our approach to evaluating a Turing machine (TM) M homomorphically over encrypted data without running in worst-case time on all inputs. We refer the reader to our full paper [GKP$^+$12] for a formal presentation.

Our idea is to use our functional encryption scheme to enable the evaluator to determine at various intermediary steps in the evaluation whether the computation finished or not. For each intermediary step, the client provides a secret key for a function that returns a bit indicating whether the computation finished or not. However, if the client provides a key for every computation step, then the amount of keys corresponds to the worst-case running time. Thus, instead, we choose intermediary points spaced at exponentially increasing intervals. In this way, the client generates only a logarithmic number of keys, namely for functions indicating if the computation finishes in $1, 2, 4, \ldots, 2^i, \ldots, 2^{\lceil \log t_{\mathsf{max}} \rceil}$ steps, where t_{max} is the worst-case running time of M on all inputs of a certain size.

Because of the single-key aspect of our FE scheme, the client cannot provide keys for an arbitrary number of TMs to the evaluator. However, this does not mean that the evaluator can run only an a priori fixed number of TMs on the encrypted data. The reason is that the client can provide keys for the universal TMs $U_0, \ldots, U_{\lceil \log t_{\mathsf{max}} \rceil}$, where TM U_i is the TM that on input a TM M and a value x, runs M on x for 2^i steps and outputs whether M finished.

Therefore, in an offline preprocessing phase, the client provides $1 + \lceil \log t_{\mathsf{max}} \rceil$ keys where the i-th key is for a circuit corresponding to U_i, each key being generated with a different master secret key. The work of the client in this phase is at least t_{max} which is costly, but this work happens only once and is amortized over all subsequent inputs in the online phase.

In an online phase, the client receives an input x and wants the evaluator to compute $M(x)$ for her. The client provides FE encryptions of (M, x) to the evaluator together with an FHE ciphertext $(\hat{M}, \hat{x})$ for (M, x) to be used for a separate FHE evaluation. The evaluator tries each key sk_{U_i} from the preprocessing phase and learns the smallest i for which the computation of M on x stops in 2^i steps. The evaluator then computes a universal circuit of size $\tilde{O}(2^i)$ and evaluates it homomorphically over $(\hat{M}, \hat{x})$, obtaining the FHE encryption of $M(x)$. Thus, we can see that the evaluator runs in time polynomial in the runtime of M on x.

□ 6.2 Preliminaries

Let κ denote the security parameter throughout this chapter. For a distribution $\mathcal{D}$, we say $x \leftarrow \mathcal{D}$ when x is sampled from the distribution $\mathcal{D}$. If S is a finite set, by $x \leftarrow S$ we mean x is sampled from the uniform distribution over the set S. Let $[n]$ denote the set $\{1, \ldots, n\}$ for $n \in \mathbb{N}^*$. When saying that a Turing machine A is p.p.t. we mean that A is a non-uniform probabilistic polynomial-time machine.

In this chapter, we only work with arithmetic circuits over GF(2). These circuits have two types of gates: $+ \bmod 2$ and $\times \bmod 2$. Unless the context specifies otherwise, we consider circuits with one bit of output (also called boolean).

◇ 6.2.1 Building Blocks

We present the building blocks that our construction relies on. We provide only informal definitions and theorems here, and refer the reader to our full paper [GKP+12] for their formal counterparts.

The LWE assumption. The security of our results will be based on the Learning with Errors (LWE) assumption, first introduced by Regev [Reg05]. Regev showed that solving the LWE problem *on average* is (quantumly) as hard as solving the approximate version of several standard lattice problems, such as gapSVP *in the worst case*. Peikert [Pei09] later removed the quantum assumption from a variant of this reduction. Given this connection, we state all our results under worst-case lattice assumptions, and in particular, under (a variant of) the gapSVP assumption. We refer the reader to [Reg05, Pei09] for details about the worst-case/average-case connection.

The best known algorithms to solve these lattice problems with an approximation factor 2^{ℓ^ϵ} in ℓ-dimensional lattices run in time $2^{\tilde{O}(\ell^{1-\epsilon})}$ [AKS01, MV10] for any constant $0 < \epsilon < 1$. Specifically, given the current state-of-the-art on lattice algorithms, it is quite plausible that achieving approximation factors 2^{ℓ^ϵ} for these lattice problems is hard for polynomial time algorithms.

FHE. Fully homomorphic encryption enables an evaluator to compute on encrypted data without learning anything about the underlying data. Formally, a $\mathcal{C}$-homomorphic encryption scheme FHE for a class of circuits $\mathcal{C}$ is a tuple of polynomial-time algorithms ($\mathsf{FHE.KeyGen}$, $\mathsf{FHE.Enc}$, $\mathsf{FHE.Dec}$, $\mathsf{FHE.Eval}$). The key generation algorithm $\mathsf{FHE.KeyGen}(1^\kappa)$ takes as input the security parameter 1^κ and outputs a public/secret key pair (hpk, hsk). The encryption algorithm $\mathsf{FHE.Enc}(\mathsf{hpk}, x \in \{0,1\})$ takes as input the public key hpk and a bit x and outputs a ciphertext ψ, whereas the decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \psi)$ takes as input the secret key hsk and a ciphertext ψ and outputs a decrypted bit. The homomorphic evaluation algorithm $\mathsf{FHE.Eval}(\mathsf{hpk}, C, \psi_1, \psi_2, \ldots, \psi_n)$ takes as input the public key hpk, n ciphertexts $\psi_1, \ldots, \psi_n$ (which are encryptions of bits $x_1, \ldots, x_n$) and a circuit $C \in \mathcal{C}$ that takes n bits as input. It outputs a ciphertext ψ_C which decrypts to $C(x_1, \ldots, x_n)$. The security definition is semantic security (or IND-CPA).

A fully homomorphic encryption scheme is homomorphic for the class of all polynomial-sized circuits. A special type of homomorphic encryption, called leveled fully homomorphic encryption, suffices for our purposes: in a d-leveled FHE scheme, $\mathsf{FHE.KeyGen}$ takes an additional input 1^d and the resulting scheme is homomorphic for all depth-d arithmetic circuits over $\mathsf{GF}(2)$.

Theorem 7 ([BV11a, BGV12]). *Assume that there is a constant $0 < \epsilon < 1$ such that for every sufficiently large ℓ, the approximate shortest vector problem* gapSVP *in ℓ dimensions is hard to approximate to within a $2^{O(\ell^\epsilon)}$ factor in the worst case. Then, for every n and every polynomial $d = d(n)$, there is an IND-CPA secure d-leveled fully homomorphic encryption scheme where encrypting n bits produces ciphertexts of length $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$, the size of the circuit for homomorphic evaluation of a function f is $\mathsf{size}(C_f) \cdot \mathrm{poly}(n, \kappa, d^{1/\epsilon})$ and its depth is $\mathsf{depth}(C_f) \cdot \mathrm{poly}(\log n, \log d)$.*

Garbled circuits. Garbled circuits were initially presented by Yao [Yao82], then proven secure by Lindell and Pinkas [LP09], and recently formalized by Bellare et al. [BHR12].

A garbling scheme for a family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where $\mathcal{C}_n$ is a set of boolean circuits taking n-bit inputs, is a tuple of p.p.t. algorithms $\mathsf{Gb} = (\mathsf{Gb.Garble}, \mathsf{Gb.Enc}, \mathsf{Gb.Eval})$ such that $\mathsf{Gb.Garble}(1^\kappa, C)$ takes as input the security parameter κ and a circuit $C \in \mathcal{C}_n$ for some n, and outputs the garbled circuit Γ and a secret key sk; $\mathsf{Gb.Enc}(\mathsf{sk}, x)$ takes as input $x \in \{0,1\}^*$ and outputs an encoding c whose size must not depend on the size of the circuit C; and $\mathsf{Gb.Eval}(\Gamma, c)$ takes as input a garbled circuit Γ and an encoding c, and outputs a value y that must be equal to $C(x)$.

The garbling scheme presented by Yao has a specific property that is useful in various secure function evaluation (SFE) protocols and in our construction as well. The secret key is of the form $\mathsf{sk} = \{L_i^0, L_i^1\}_{i=1}^n$ and the encoding of an n-bit input x is of the form $c = (L_1^{x_1}, \ldots, L_n^{x_n})$ where x_i is the i-th bit of x.

Two security guarantees are of interest: input privacy (the input to the garbled circuit does not leak to the adversary) and circuit privacy (the circuit does not leak to the adversary). In all known garbling schemes, these properties hold *only for one-time* evaluation of the circuit: the adversary can receive at most one encoding of an input to use with a garbled circuit; obtaining more than one encoding breaks these security guarantees. More formally, the security definition states that there exists a p.p.t. simulator $\mathsf{Sim}_{\mathsf{Garble}}$ that given the result $C(x)$ of a (secret) circuit C on a *single* (secret) input x, and given the sizes of C and x (but not the actual values of C and x), outputs a simulated garbled circuit $\tilde{\Gamma}$ and an encoding $\tilde{c}$, $(\tilde{\Gamma}, \tilde{c}) \leftarrow \mathsf{Sim}_{\mathsf{Garble}}(1^\kappa, C(x), 1^{|C|}, 1^{|x|})$, that are computationally indistinguishable from the real garbled circuit Γ and encoding c.

Theorem 8 ([Yao82, LP09]). *Assuming one-way functions exist, there exists a Yao (one-time) garbling scheme that is input- and circuit-private for all circuits over GF(2).*

◇ 6.2.2 Attribute-Based Encryption (ABE)

Attribute-based encryption is an important component of our construction. We present a slight (but equivalent) variant of ABE that better serves our goal.

Definition 23. *An attribute-based encryption scheme* (ABE) *for a class of predicates $\mathcal{P} = \{\mathcal{P}_n\}_{n\in\mathbb{N}}$ represented as boolean circuits with n input bits, and a message space $\mathcal{M}$ is a tuple of algorithms* (ABE.Setup, ABE.KeyGen, ABE.Enc, ABE.Dec) *as follows:*

- $\mathsf{ABE.Setup}(1^\kappa)$*: Takes as input a security parameter 1^κ and outputs a public master key* fmpk *and a master secret key* fmsk.
- $\mathsf{ABE.KeyGen}(\mathsf{fmsk}, P)$*: Given a master secret key* fmsk *and a predicate $P \in \mathcal{P}$, outputs a key* fsk_P *corresponding to P.*

- $\mathsf{ABE.Enc}(\mathsf{fmpk}, x, M_0, M_1)$: *Takes as input the public key* fmpk, *an attribute* $x \in \{0,1\}^n$, *for some* n, *and two messages* $M_0, M_1 \in \mathcal{M}$ *and outputs a ciphertext* c.
- $\mathsf{ABE.Dec}(\mathsf{fsk}_P, c)$: *Takes as input a secret key for a predicate and a ciphertext and outputs* $M^* \in \mathcal{M}$.

Correctness. *For any polynomial* $n(\cdot)$, *for every sufficiently large security parameter* κ, *if* $n = n(\kappa)$, *for all predicates* $P \in \mathcal{P}_n$, *attributes* $x \in \{0,1\}^n$, *messages* $M_0, M_1 \in \mathcal{M}$:

$$\Pr\left[\begin{array}{l}(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{ABE.Setup}(1^\kappa);\\ \mathsf{fsk}_P \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}, P);\\ c \leftarrow \mathsf{ABE.Enc}(\mathsf{fmpk}, x, M_0, M_1);\\ M^* \leftarrow \mathsf{ABE.Dec}(\mathsf{fsk}_P, c) : M^* = M_{P(x)}\end{array}\right] = 1 - \mathrm{negl}(\kappa).$$

Informally, the security of ABE guarantees that nothing leaks about M_0 if $P(x) = 1$ and nothing leaks about M_1 if $P(x) = 0$. However, the scheme does not hide the attribute x, and x may leak no matter what $P(x)$ is. The security of ABE is thus conceptually weaker than the security for FE: the input that the computation happens on leaks with ABE, while this input does not leak with FE.

We call an ABE or FE scheme for circuits of depth d a *d-leveled* ABE or *d-leveled* FE scheme, respectively.

Theorem 9 ([GVW13]). *Assume there is a constant* $0 < \epsilon < 1$ *such that for every sufficiently large* ℓ, *the approximate shortest vector problem* gapSVP *in* ℓ *dimensions is hard to approximate to within a* $2^{O(\ell^\epsilon)}$ *factor in* $\mathrm{poly}(\ell)$ *(resp.* 2^{ℓ^ϵ}*) time. Then, for every* n *and every polynomial* $d = d(n)$, *there is a selectively (resp. fully) secure* d*-leveled attribute-based encryption scheme where encrypting* n *bits produces ciphertexts of length* $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$ *(resp.* $\mathsf{poly}(n, \kappa, d^{1/\epsilon^2})$*).*

◇ 6.2.3 Functional Encryption (FE)

We recall the functional encryption definition from the literature [KSW08, BSW11, GVW12] with some notational changes.

Definition 24. *A functional encryption scheme* FE *for a class of functions* $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ *represented as boolean circuits with an* n*-bit input, is a tuple of four p.p.t. algorithms (*$\mathsf{FE.Setup}$, $\mathsf{FE.KeyGen}$, $\mathsf{FE.Enc}$, $\mathsf{FE.Dec}$*) such that:*

- $\mathsf{FE.Setup}(1^\kappa)$ *takes as input the security parameter* 1^κ *and outputs a master public key* fmpk *and a master secret key* fmsk.
- $\mathsf{FE.KeyGen}(\mathsf{fmsk}, f)$ *takes as input the master secret key* fmsk *and a function* $f \in \mathcal{F}$ *and outputs a key* fsk_f.
- $\mathsf{FE.Enc}(\mathsf{fmpk}, x)$ *takes as input the master public key* fmpk *and an input* $x \in \{0,1\}^*$ *and outputs a ciphertext* c.
- $\mathsf{FE.Dec}(\mathsf{fsk}_f, c)$ *takes as input a key* fsk_f *and a ciphertext* c *and outputs a value* y.

Correctness. *For any polynomial* $n(\cdot)$, *for every sufficiently large security parameter* κ, *for* $n =$

$n(\kappa)$, for all $f \in \mathcal{F}_n$, and all $x \in \{0,1\}^n$,

$$\begin{aligned}&\Pr[(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa); \mathsf{fsk}_f \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, f);\\ &\quad c \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, x) : \mathsf{FE.Dec}(\mathsf{fsk}_f, c) = f(x)]\\ &\quad = 1 - \mathrm{negl}(\kappa).\end{aligned}$$

Intuitively, the security of functional encryption requires that an adversary should not learn anything about the input x other than the computation result $C(x)$, for some circuit C for which a key was issued (the adversary can learn the circuit C). In this chapter, we present only the definition of full security and defer the definition of selective security to our full paper [GKP+12]. The security definition states that whatever information an adversary is able to learn from the ciphertext and the function keys can be simulated given only the function keys and the output of the function on the inputs.

Definition 25. *(FE Security) Let* FE *be a functional encryption scheme for the family of functions $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$. For every p.p.t. adversary $A = (A_1, A_2)$ and p.p.t. simulator S, consider the following two experiments:*

```
        $\exp^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa)$:                         $\exp^{\mathsf{ideal}}_{\mathsf{FE},A,S}(1^\kappa)$:

                      1: $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow$
                         $\mathsf{FE.Setup}(1^\kappa)$
                      2: $(f, \mathsf{state}_A) \leftarrow A_1(\mathsf{fmpk})$
                      3: $\mathsf{fsk}_f \leftarrow$
                         $\mathsf{FE.KeyGen}(\mathsf{fmsk}, f)$
                      4: $(x, \mathsf{state}'_A) \leftarrow$
                         $A_2(\mathsf{state}_A, \mathsf{fsk}_f)$
5: $c \leftarrow$                                  5: $\tilde{c} \leftarrow$
   $\mathsf{FE.Enc}(\mathsf{fmpk}, x)$                        $S(\mathsf{fmpk}, \mathsf{fsk}_f, f, f(x), 1^{|x|})$
6: Output $(\mathsf{state}'_A, c)$                 6: Output $(\mathsf{state}'_A, \tilde{c})$
```

The scheme is said to be (single-key) FULL-SIM*−secure if there exists a p.p.t. simulator S such that for all pairs of p.p.t. adversaries (A_1, A_2), the outcomes of the two experiments are computationally indistinguishable:*

$$\left\{ \exp^{\mathsf{real}}_{\mathsf{FE},A}(1^\kappa) \right\}_{\kappa\in\mathbb{N}} \overset{c}{\approx} \left\{ \exp^{\mathsf{ideal}}_{\mathsf{FE},A,S}(1^\kappa) \right\}_{\kappa\in\mathbb{N}}.$$

□ 6.3 Our Functional Encryption

In this section, we present our main result: the construction of a functional encryption scheme FE. We refer the reader to the introduction (Sec. 6.1.2) for an overview of our approach, and we proceed directly with the construction here.

We use three building blocks in our construction: a (leveled) fully homomorphic encryption scheme FHE, a (leveled) two-outcome attribute-based encryption scheme ABE, and a Yao garbling scheme Gb.

For simplicity, we construct FE for functions outputting one bit; functions with larger outputs can be handled by repeating our scheme below for every output bit. Let $\lambda = \lambda(\kappa)$ be the length of the ciphertexts in the FHE scheme (both from encryption and evaluation). The construction of $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ proceeds as follows.

Setup $\mathsf{FE.Setup}(1^\kappa)$: Run the setup algorithm for the ABE scheme λ times:

$$(\mathsf{fmpk}_i, \mathsf{fmsk}_i) \leftarrow \mathsf{ABE.Setup}(1^\kappa) \text{ for } i \in [\lambda].$$

Output as master public key and secret key:

$$\mathsf{MPK} = (\mathsf{fmpk}_1, \dots, \mathsf{fmpk}_\lambda) \text{ and } \mathsf{MSK} = (\mathsf{fmsk}_1, \dots, \mathsf{fmsk}_\lambda).$$

Key Generation $\mathsf{FE.KeyGen}(\mathsf{MSK}, f)$: Let n be the number of bits in the input to the circuit f. If hpk is an FHE public key and $\psi_1, \dots, \psi_n$ are FHE ciphertexts, let $\mathsf{FHE.Eval}^i_f(\mathsf{hpk}, \psi_1, \dots, \psi_n)$ be the i-th bit of the homomorphic evaluation of f on $\psi_1, \dots, \psi_n$ ($\mathsf{FHE.Eval}(\mathsf{hpk}, f, \psi_1, \dots, \psi_n)$), where $i \in [\lambda]$. Thus, $\mathsf{FHE.Eval}^i_f : \{0,1\}^{|\mathsf{hpk}|} \times \{0,1\}^{n\lambda} \to \{0,1\}$.

1. Run the key generation algorithm of ABE for the functions $\mathsf{FHE.Eval}^i_f$ (under the different master secret keys) to construct secret keys:

$$\mathsf{fsk}_i \leftarrow \mathsf{ABE.KeyGen}(\mathsf{fmsk}_i, \mathsf{FHE.Eval}^i_f) \text{ for } i \in [\lambda].$$

2. Output the tuple $\mathsf{fsk}_f := (\mathsf{fsk}_1, \dots, \mathsf{fsk}_\lambda)$ as the secret key for the function f.

Encryption $\mathsf{FE.Enc}(\mathsf{MPK}, x)$: Let n be the number of bits of x, namely $x = x_1 \dots x_n$. Encryption proceeds in three steps.

1. Generate a fresh key pair $(\mathsf{hpk}, \mathsf{hsk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$ for the (leveled) fully homomorphic encryption scheme. Encrypt each bit of x homomorphically: $\psi_i \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, x_i)$. Let $\psi := (\psi_1, \dots, \psi_n)$ be the encryption of the input x.
2. Run the Yao garbled circuit generation algorithm to produce a garbled circuit for the FHE decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$ together with 2λ labels L^b_i for $i \in [\lambda]$ and $b \in \{0,1\}$. Namely,

$$\left(\Gamma, \{L^0_i, L^1_i\}_{i=1}^{\lambda}\right) \leftarrow \mathsf{Gb.Garble}(1^\kappa, \mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)),$$

   where Γ is the garbled circuit and the L^b_i are the input labels.
3. Produce encryptions $c_1, \dots, c_\lambda$ using the ABE scheme:

$$c_i \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i,\ (\mathsf{hpk}, \psi), L^0_i, L^1_i\right) \text{ for } i \in [\lambda],$$

   where (hpk, ψ) comes from the first step, and the labels (L^0_i, L^1_i) come from the second step.
4. Output the ciphertext $c = (c_1, \dots, c_\lambda, \Gamma)$.

Decryption $\mathsf{FE.Dec}(\mathsf{fsk}_f, c)$:

1. Run the ABE decryption algorithm on the ciphertexts $c_1, \ldots, c_\lambda$ to recover the labels for the garbled circuit. In particular, let

$$L_i^{d_i} \leftarrow \mathsf{ABE.Dec}(\mathsf{fsk}_i, c_i) \text{ for } i \in [\lambda],$$

where d_i is equal to $\mathsf{FHE.Eval}_f^i(\mathsf{hpk}, \psi)$.

2. Now, armed with the garbled circuit Γ and the labels $L_i^{d_i}$, run the garbled circuit evaluation algorithm to compute

$$\mathsf{Gb.Eval}(\Gamma, L_1^{d_1}, \ldots, L_\lambda^{d_\lambda}) = \mathsf{FHE.Dec}(\mathsf{hsk}, d_1 d_2 \ldots d_\lambda) = f(x).$$

We now provide a proof for our main Theorem 4, delegating certain details to the full paper [GKP$^+$12].

of Theorem 4. Let us first argue that the scheme is correct. We examine the values we obtain in $\mathsf{FE.Dec}(\mathsf{fsk}_f, c)$. In Step (1), by the correctness of the ABE scheme used, d_i is $\mathsf{FHE.Eval}_f^i(\mathsf{hpk}, \psi)$: $\mathsf{FHE.Eval}_f^i$ comes from fsk_f and (hpk, ψ) come from c_i. Therefore, the inputs to the garbled circuit Γ in Step (2) are the set of λ labels corresponding to the value of $\mathsf{FHE.Eval}_f(\mathsf{hpk}, \psi)$. By the correctness of the FHE scheme, this value corresponds to an FHE encryption of $f(x)$. Finally, by the correctness of the garbled circuit scheme, and by how Γ was constructed in FE.Enc, the FHE ciphertext gets decrypted by Γ correctly, yielding $f(x)$ as the output from FE.Dec.

We now prove the succinctness property—namely, that the size of FE ciphertexts is independent of the size of the circuit. FE's ciphertext is the output of FE.Enc, which outputs λ ciphertexts from ABE.Enc and a garbled circuit from Gb.Garble. These add up as follows. First, $\lambda = \mathsf{ctsize}_{\mathsf{FHE}}$, the size of the ciphertext in FHE. Second, we denote the size of the ciphertext produced by ABE.Enc as $\mathsf{ctsize}_{\mathsf{ABE}}(\cdot)$, which is a function of ABE.Enc's input size. The input provided by FE.Enc to ABE.Enc consists of $\mathsf{pksize}_{\mathsf{FHE}}$ bits for hpk, $n \cdot \mathsf{ctsize}_{\mathsf{FHE}}$ bits for ψ, and $\mathrm{poly}(\kappa)$ bits for the labels. Finally, the garbled circuit size is polynomial in the size of the input circuit passed to Gb.Garble, which in turn is polynomial in $\mathsf{sksize}_{\mathsf{FHE}}$ and $\mathsf{ctsize}_{\mathsf{FHE}}$. Thus, we obtain the overall ciphertext size of FE: $\mathsf{ctsize}_{\mathsf{FE}} = \mathsf{ctsize}_{\mathsf{FHE}} \cdot \mathsf{ctsize}_{\mathsf{ABE}}(\mathsf{pksize}_{\mathsf{FHE}} + n \cdot \mathsf{ctsize}_{\mathsf{FHE}} + \mathrm{poly}(\kappa)) + \mathrm{poly}(\kappa, \mathsf{sksize}_{\mathsf{FHE}}, \mathsf{ctsize}_{\mathsf{FHE}})$. We can thus see that if FHE and ABE produce ciphertexts and public keys independent of the circuit size, then so will our functional encryption scheme.

Finally, we prove security of our scheme based on Def. 25. We construct a p.p.t. simulator S that achieves Def. 25. S receives as input $(\mathsf{MPK}, \mathsf{fsk}_f, f, f(x), 1^n)$ and must output $\tilde{c}$ such that the real and ideal experiments in Def. 25 are computationally indistinguishable. Intuitively, S runs a modified version of FE.Enc to mask the fact that it does not know x.

Simulator S on input $(\mathsf{MPK}, \mathsf{fsk}_f, f, f(x), 1^n)$:

1. Choose a key pair $(\mathsf{hpk}, \mathsf{hsk}) \leftarrow \mathsf{FHE.KeyGen}(1^\kappa)$ for the homomorphic encryption scheme (where S can derive the security parameter κ from the sizes of the inputs it gets). Encrypt 0^n (n zero bits) with FHE by encrypting each bit individually and denote the ciphertext $\hat{0} := (\hat{0}_1 \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, 0), \ldots, \hat{0}_n \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, 0))$.

2. Let $\mathsf{Sim}_{\mathsf{Garble}}$ be the simulator for the Yao garbling scheme (described in Sec. 6.2.1) for the class of circuits corresponding to $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)$. Run $\mathsf{Sim}_{\mathsf{Garble}}$ to produce a simulated garbled circuit $\tilde{\Gamma}$ for the FHE decryption algorithm $\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\lambda \to \{0,1\}$ together with the

simulated encoding consisting of one set of λ labels $\tilde{L}_i$ for $i = 1 \dots \lambda$. Namely,

$$\left(\tilde{\Gamma}, \{\tilde{L}_i\}_{i=1}^{\lambda}\right) \leftarrow \mathsf{Sim}_{\mathsf{Garble}}(1^{\kappa}, f(x), 1^{|\mathsf{FHE.Dec(hsk,\cdot)}|}, 1^{\lambda}).$$

The simulator S can invoke $\mathsf{Sim}_{\mathsf{Garble}}$ because it knows $f(x)$, and can compute the size of the $\mathsf{FHE.Dec(hsk}, \cdot)$ circuit, and λ from the sizes of the input parameters.

3. Produce encryptions $\tilde{c}_1, \dots, \tilde{c}_\lambda$ under the ABE scheme in the following way. Let

$$\tilde{c}_i \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \hat{0}), \tilde{L}_i, \tilde{L}_i\right),$$

where S uses each simulated label $\tilde{L}_i$ twice.

4. Output $\tilde{c} = (\tilde{c}_1, \dots, \tilde{c}_\lambda, \tilde{\Gamma})$.

To prove indistinguishability of the real and ideal experiments (Def. 25), we define a sequence of hybrid experiments, and then invoke the security definitions of the underlying schemes (FHE, garbled circuit, and ABE respectively) to show that the outcome of the hybrid experiments are computationally indistinguishable.

Hybrid 0 is the output of the ideal experiment from Def. 25 for our FE construction with simulator S.

Hybrid 1 is the same as Hybrid 0, except that the simulated ciphertext for Hybrid 1 (which we denote $\tilde{c}^{(1)}$), changes. Let $\tilde{c}^{(1)}$ be the ciphertext obtained by running the algorithm of S, except that in Step (3), encrypt x instead of 0, namely:

$$\tilde{c}_i^{(1)} \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), \tilde{L}_i, \tilde{L}_i\right),$$

where $\psi \leftarrow (\mathsf{FHE.Enc}(\mathsf{hpk}, x_1), \dots, \mathsf{FHE.Enc}(\mathsf{hpk}, x_n))$. Let

$$\tilde{c}^{(1)} = (\tilde{c}_1^{(1)}, \dots, \tilde{c}_\lambda^{(1)}, \tilde{\Gamma}).$$

Hybrid 2 is the same as Hybrid 1, except that in Step (2), the ciphertext contains a real garbled circuit

$$\left(\Gamma, \{L_i^0, L_i^1\}_{i=1}^{\lambda}\right) \leftarrow \mathsf{Gb.Garble}(\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot)).$$

Let $d_i = \mathsf{FHE.Eval}_f^i(\mathsf{hpk}, \psi)$. In Step (3), include L^{d_i} twice in the ABE encryption; namely:

$$\tilde{c}_i^{(2)} \leftarrow \mathsf{ABE.Enc}\left(\mathsf{fmpk}_i, (\mathsf{hpk}, \psi), L_i^{d_i}, L_i^{d_i}\right), \text{ and}$$

$$\tilde{c}^{(2)} = (\tilde{c}_1^{(2)}, \dots, \tilde{c}_\lambda^{(2)}, \Gamma).$$

Hybrid 3 is the output of the real experiment from Def. 25 for our FE construction.

In our full paper [GKP+12], we prove that each pair of consecutive hybrids are computationally indistinguishable: Hybrid 0 and Hybrid 1 by the security of the homomorphic scheme FHE, Hybrid 1 and Hybrid 2 by the security of the garbled circuit scheme Gb, and Hybrid 2 and Hybrid 3 by the security of the ABE scheme ABE, thus completing our proof. □

Instantiating the components with the leveled fully homomorphic encryption scheme of [BV11a] (see Theorem 7), the leveled attribute-based encryption scheme of [GVW13] (see Theorem 9) and Yao garbled circuit from one-way functions (see Theorem 8), we get the following corollary of Theorem 4:

Corollary 6 (The LWE Instantiation). *Assume that there is a constant $0 < \epsilon < 1$ such that for every sufficiently large ℓ, the approximate shortest vector problem* gapSVP *in ℓ dimensions is hard to approximate to within a $2^{O(\ell^\epsilon)}$ factor in the worst case in time $\mathrm{poly}(\ell)$ (resp. 2^{ℓ^ϵ}) time. Then, for every n and every polynomial $d = d(n)$, there is a selectively secure (resp. fully secure) functional encryption scheme for depth d circuits, where encrypting n bits produces ciphertexts of length* $\mathsf{poly}(n, \kappa, d^{1/\epsilon})$ *(resp.* $\mathsf{poly}(n^{1/\epsilon}, \kappa, d^{1/\epsilon^2})$*).*

□ 6.4 Reusable Garbled Circuits

In this section, we show how to use our functional encryption scheme to construct *reusable* garbled circuits. The syntax of a reusable garbling scheme RGb = (RGb.Garble, RGb.Enc, RGb.Eval) is the same as the syntax for a one-time garbling scheme (Sec. 6.2.1). The security of the scheme (defined in our full paper [GKP+12]), intuitively says that a garbled circuit can be used with many encodings while still hiding the circuit and the inputs. More formally, a garbling scheme is input- and circuit-private with reusability if there exists a stateful p.p.t. simulator $S = (S_1, S_2)$ such that S_1, when given as input a circuit size $|C|$, outputs a simulated garbled circuit, and S_2 when given as input $C(x)$ outputs a simulated encoding of x. The simulated garbled circuit and the encodings must be computationally indistinguishable from the real garbled circuit and encodings. Note that S never gets x or C and the adversary can invoke S_2 many times (reusability). Sec. 6.1.2 already gave an overview of the idea behind our construction, so we proceed to the construction. Let E = (E.KeyGen, E.Enc, E.Dec) be a semantically secure symmetric-key encryption scheme.

Garbling $\mathsf{RGb.Garble}(1^\kappa, C)$**:**

1. Generate FE keys $(\mathsf{fmpk}, \mathsf{fmsk}) \leftarrow \mathsf{FE.Setup}(1^\kappa)$ and a secret key $\mathsf{sk} \leftarrow \mathsf{E.KeyGen}(1^\kappa)$.
2. Let $E := \mathsf{E.Enc}(\mathsf{sk}, C)$.
3. Define U_E to be the following universal circuit:

   U_E takes as input a secret key sk and a value x:

   (a) Compute $C := \mathsf{E.Dec}(\mathsf{sk}, E)$.

   (b) Run C on x.
4. Let $\Gamma \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, U_E)$ be the reusable garbled circuit.
5. Output $\mathsf{gsk} := (\mathsf{fmpk}, \mathsf{sk})$ as the secret key and Γ as the garbling of C.

Encoding $\mathsf{RGb.Enc}(\mathsf{gsk}, x)$**:** Compute $c_x \leftarrow \mathsf{FE.Enc}(\mathsf{fmpk}, (\mathsf{sk}, x))$ and output c_x.

Evaluation $\mathsf{RGb.Eval}(\Gamma, c_x)$**:** Compute and output $\mathsf{FE.Dec}(\Gamma, c_x)$.

The existence of a semantically secure encryption scheme does not introduce new assumptions because the FE scheme itself is a semantically secure encryption scheme if no key (computed by FE.KeyGen) is ever provided to an adversary.

Tightness of the scheme. The astute reader may have observed that the resulting scheme requires that the encodings be generated in the secret key setting because the encoding of x includes sk. It turns out that generating encodings privately is in fact necessary; if the encodings were publicly generated, the power of the adversary would be the same as in traditional obfuscation, which was shown impossible [BGI+01, GK05] (see discussion in Sec. 6.1.1).

One might wonder though, whether a reusable garbling scheme exists where the encoding generation is secret key, but RGb.Garble is public key. We prove in our full paper that this is also not possible based on the impossibility result of [AGVW12]; hence, with regard to public versus private key, our reusable garbling result is tight.

Proof sketch of Theorem 5. Let us first argue RGb.Eval is correct. By the definition of RGb.Eval, $\mathsf{RGb.Eval}(\Gamma, c_x)$ equals $\mathsf{FE.Dec}(\Gamma, c_x)$, which equals $U_E(\mathsf{sk}, x)$ by the correctness of FE. Finally, by the definition of U_E, $U_E(\mathsf{sk}, x) = C(x)$.

Notice that the encoding algorithm RGb.Enc produces ciphertexts that do not depend on the circuit size, because of the succinctness property of FE.

We can see that to obtain a RGb scheme for circuits of depth d, we need a FE scheme for polynomially deeper circuits: the overhead comes from the fact that U is universal and it also needs to perform decryption of E to obtain C.

Intuitively, the scheme is secure because E hides the circuit C. Now since FE hides the inputs to FE.Enc, it hides x and sk, and reveals only the result of the computation which is $C(x)$. To prove security formally, we need to construct a simulator $S = (S_1, S_2)$ such that the simulated garbled circuit and encodings are computationally indistinguishable from the real ones. (Our full paper [GKP+12] precisely defines security for RGb, including the games for the simulator.) To produce a simulated garbled circuit $\tilde{\Gamma}$, S_1 on input $(1^\kappa, 1^{|C|})$ runs:

1. Generate fresh fmpk, fmsk, and sk as in RGb.Garble.
2. Compute $\tilde{E} := \mathsf{E.Enc}(\mathsf{sk}, 0^{|C|})$. (The reason for encrypting $0^{|C|}$ is that S_1 does not know C).
3. Compute and output $\tilde{\Gamma} \leftarrow \mathsf{FE.KeyGen}(\mathsf{fmsk}, U_{\tilde{E}})$.

S_2 receives queries for values $x_1, \ldots, x_t \in \{0,1\}^*$ for some t and needs to output a simulated encoding for each of these. To produce a simulated encoding for x_i, S_2 receives inputs ($C(x_i)$, $1^{|x_i|}$, and the latest simulator's state) and invokes the simulator $\mathsf{Sim}_{\mathsf{FE}}$ of the FE scheme and outputs

$$\tilde{c}_x := \mathsf{Sim}_{\mathsf{FE}}(\mathsf{fmpk}, \mathsf{fsk}_{U_{\tilde{E}}}, U_{\tilde{E}}, C(x), 1^{|\mathsf{sk}|+|x_i|}).$$

A potentially alarming aspect of this simulation is that S generates a key for the circuit $0^{|C|}$. Whatever circuit $0^{|C|}$ may represent, it may happen that there is no input x to $0^{|C|}$ that results in the value $C(x)$. The concern may then be that $\mathsf{Sim}_{\mathsf{FE}}$ may not simulate correctly. However, this is not a problem because, by semantic security, E and $\tilde{E}$ are computationally indistinguishable so $\mathsf{Sim}_{\mathsf{FE}}$ must work correctly, otherwise it breaks semantic security of the encryption scheme E.

To prove indistinguishability of the simulated/ideal and real experiment outputs, we introduce a hybrid experiment. This experiment is the same as the ideal experiment, but $\tilde{E}$ is replaced with $E = \mathsf{E.Enc}(\mathsf{sk}, C)$. This means that the adversary receives a real garbled circuit, but the encodings are still simulated. Note that the view of the adversary in the ideal and the hybrid experiment differs only in $\tilde{E}$ and E. Since these views do not contain sk or any other function of sk other than $E/\tilde{E}$, by semantic security of the encryption scheme, they are computationally indistinguishable. Finally, the hybrid and the real experiment are computationally indistinguishable based on the properties of $\mathsf{Sim}_{\mathsf{FE}}$ guaranteed by the security of FE. For the full proof, we refer the reader to our full paper [GKP$^+$12]. $\square$

CHAPTER 7

Impact: industry and academia

The work in this thesis has had both academic and industrial impact: it spurred new research in the academic community investigating a rich set of directions (Sec. 7.2) as well as determined real deployment by existing companies (Sec. 7.1). CryptDB has had the most impact; hence, the academic impact section below focuses only on CryptDB.

□ 7.1 Industrial impact and deployments

The following companies or institutions adopted CryptDB or Mylar.

▷ **Google** recently deployed a system for performing SQL-like queries over an encrypted database following (and giving credit to) the CryptDB design. Their service, called the *Encrypted BigQuery*, uses the encryption building blocks from CryptDB (RND, DET, HOM, and SEARCH), rewrites queries and annotates the schema as in CryptDB. Google made available the source code of the Encrypted BigQuery client [Goo].

▷ **SAP AG**, a well-known software company, implemented CryptDB on top of their HANA database system. The resulting system is called *SEEED*. Grofig et al. [GHH+14] describes their experiences and observations with the system.

▷ **Lincoln Labs** implemented CryptDB's design on top of their D4M Accumulo no-SQL engine (using the RND, DET, OPE and HOM building blocks). The resulting system, called *CMD*, is documented in Kepner et al. [KGM+14].

▷ **sql.mit.edu** is a SQL server at MIT hosting many MIT-ran applications. Volunteering users of Wordpress ran their Wordpress applications through CryptDB, using our source code.

▷ **Newton-Wellesley Hospital, Boston**. The endometriosis medical application of this hospital is a web application that collects private information from patients suffering from the disease endometriosis. It was secured with Mylar, was initially tested by patients who suffered from endometriosis, and is in alpha deployment.

□ 7.2 Academic impact

A number of papers followed the work on CryptDB. In this section, we summarize only the highlights of such works. We can split such work in the following categories (not necessarily mutually exclusive): extending the functionality CryptDB supports (for example, to full SQL), applying CryptDB to different system settings (e.g., Java programs or MapReduce), using it in various applications (e.g., genomic computation), or to Java programs), security-related proposals (e.g., giving users' control on the encryption policy), and miscellaneous.

◇ 7.2.1 Extending CryptDB's functionality

Since CryptDB supports a limited set of SQL, some work focused on extending CryptDB to support all of SQL. It is currently unknown how to achieve this goal in a practical way by simply using richer encryption schemes. Instead, these works chose either to weaken the threat model or to perform more work at a trusted entity (e.g., the client).

- ▷ *Cipherbase* [ABE+13, AEK+13] extends CryptDB's design to support any SQL queries at the expense of adding trusted hardware on the server side. The trusted hardware has the ability to decrypt the data and compute on it decrypted. To execute a query, Cipherbase works as follows. If CryptDB's approach supports that query, the query is processed as in CryptDB over encrypted data without invoking the trusted hardware. If CryptDB's approach does not support the query, it is sent to a trusted hardware module, which pulls in and decrypts the relevant data, executes the query on unencrypted data, and returns encrypted results.

- ▷ *Monomi* [TKMZ13] also aims to extend CryptDB to support arbitrary SQL queries. Instead of using trusted hardware, the approach is to split each query into computation that the CryptDB server can handle over encrypted data and computation that should be performed at the client over unencrypted data. Hence, the trusted client now does more work: it downloads some data and decrypts it, it performs some location computation, and sometimes reencrypts the data with a new encryption scheme.

◇ 7.2.2 Adapting CryptDB beyond SQL

Computation on encrypted data is also useful for systems or operations other than SQL databases or SQL queries: for example, Java programs, algebraic manipulations, MapReduce jobs, or untrusted VMs. The line of work in this subsection extends CryptDB's approach to such settings. They largely use the same encryption mechanisms and building blocks as in CryptDB, but combine them using new systems techniques. It is interesting to note that the same set of primitive operations that CryptDB uses were found to be useful in these settings as well.

- ▷ *MrCrypt* [TLMM13] extends CryptDB to Java programs. MrCrypt statically analyzes a program to identify the set of operations performed on each input data who is a column in a confidential database. Then, it selects an appropriate encryption scheme from CryptDB's list for that column, and transforms the program to operate over encrypted data.

- ▷ The *CMD* system of Kepner et al. [KGM+14] applies CryptDB's design to support a wide range of linear algebraic operations on encrypted data. CMD is especially fit for databases performing linear algebra operations on sparse matrices, such as SciDB or Apache Accumulo. The authors show the use of CMD for two applications: complex DNA matching and database operations over social media data.

▷ *Cryptsis* [SSSE14] is a system that applies CryptDB's design to execute MapReduce jobs on encrypted data. The motivation of Cryptsis is to provide privacy for big data. Cryptsis analyzes entire data flow programs (written in Pig Latin) to find as many opportunities to run the scripts over encrypted data as possible. The unsupported computation is ran over unencrypted data.

▷ *Autocrypt* [TSCS13] is a system for protecting data confidentiality from an untrusted VM running on a trusted cloud platform. The untrusted VM only sees encrypted data and performs computation on encrypted data as in CryptDB. To be able to combine the various encryption schemes in CryptDB and compose them to support general computation, Autocrypt has a mechanism to convert between encryption schemes using a small TCB in the trusted cloud hypervisor. Autocrypt transforms a subset of existing C functionality in the web stack to operate on encrypted sensitive content. For example, Autocrypt supports several standard Unix utilities available in a typical LAMP stack, with no developer effort.

◇ 7.2.3 Using CryptDB in applications handling private data

Some works suggest using CryptDB (largely unchanged) for various applications that handle sensitive data:

▷ *Erman et al.* [ARH+13] follow a similar approach to CryptDB's to store encrypted genomic data in a biobank (represented as a database) and then execute queries over the encrypted database.

▷ *Clome* [NA14] proposes using CryptDB for home applications in the cloud. Users of home applications store sensitive data about their homes in the cloud, and CryptDB can serve to protect its privacy from the cloud.

▷ *Corena and Ohtsuki* [CO12] apply CryptDB's building blocks for aggregating financial information at an untrusted cloud.

▷ *SensorCloud* [HHCW12] applies CryptDB's approach to storing and querying sensor data in the cloud.

▷ *Kagadis et al.* [KKM+] propose using CryptDB for medical imagining.

◇ 7.2.4 Follow-up work related to the encryption schemes CryptDB uses

A category of work improves or further analyzes the encryption schemes used in CryptDB.

Some works [BCO11], [PLZ13], [LW12b], [LW13], [KS12], [MTY13], [LCY+14] provide various improvements or analysis for order-preserving encryption, often with the purpose of using this scheme in a CryptDB-like setting.

Other works [CJJ+13], [MMA14] try to improve the performance of searchable encryption in a database by adding indexing.

Kerschbaum et al. [KHG+13] studied an optimal way of applying CryptDB's adjustable join in order to adjust the smallest amount of columns possible, when join queries are not a priori known. Naïve strategies may perform too many or even infinitely many re-encryptions. They provide two strategies, with $O(n^{3/2})$ and $O(n \log n)$ re-encryptions for n columns respectively. They further show that no algorithm can be better than $O(n \log n)$. Similarly, Bkakria et al. [BSK+14] provide a tool that computes the optimal balance between security and functionality.

◇ 7.2.5 Security

Some works enhance CryptDB by enabling the user to specify a security policy for the data, try to find an optimal balance between security and functionality or performance, or try to quantify the leakage in a global system.

▷ SMC [LSSD14] is a tool for quantifying leakage in systems that compute functions over encrypted data such as CryptDB. It solves a problem of model counting – determining the number of solutions that satisfy a given set of constraints – to which the problem of estimating leakage in CryptDB can be reduced.

▷ Securus [KJ14] allows users to specify the privacy requirements for data in the database and the queries they expect to run, and tries to find a combination of encryption schemes as in CryptDB that addresses these specifications as closely as possible.

◇ 7.2.6 Miscellaneous

A few other works addressed miscellaneous topics related to CryptDB.

▷ *Ferretti et al.* [FPCM13] measure CryptDB's onion encryption strategy in a cloud setting scenario and conclude that it can be well applied to a cloud database paradigm, because most performance overheads are masked by network latencies. In [FCM13], the same authors provide a solution for a distributed proxy for the multi-principal mode in CryptDB (we did not discuss the multi-principal mode of CryptDB in this thesis, so please refer to our CryptDB paper [PRZB11]).

▷ *Ferretti et al.* [FCM12] improve CryptDB's design by distributing CryptDB's proxy at each client. This improves scalability for many clients. SecureDBaaS [FCM14] also distribute the CryptDB proxy to the client (thus, physically eliminating the proxy) and allows concurrent accesses from multiple clients to the encrypted database.

▷ *Bohli et al.* [BGJ+13] discuss applying CryptDB's design to a setting with multiple clouds.

▷ *Tomiyama et al.* [TKK12] apply CryptDB's design to computing over encrypted data streams in a public cloud.

▷ *Islam and Chanchary* [IC12] provide a mechanism for migrating database data to new applications while maintaining data privacy using CryptDB as well as other useful properties.

CHAPTER 8

Related work

This chapter surveys the related work to this thesis. We organize it in two components: work related to the global vision of this work (and hence common to all our systems) and work related to a specific system.

□ 8.1 Global related work

◇ 8.1.1 Secure systems work

Untrusted servers. Some systems also attempted to prevent against attackers that get access to all server data. They also stored the data encrypted on the server. However, the crucial difference of these systems from our work is that they *do not compute on the encrypted data*. Hence, they can only support storage-like systems. Most systems, though, need to compute on the sensitive data: for example, database systems, web applications, mobile applications, and machine learning tools.

SUNDR [LKMS04] uses cryptography to provide privacy and integrity in a file system on top of an untrusted file server. Using a SUNDR-like model, SPORC [FZFF10] and Depot [MSL+10] show how to build low-latency applications, running mostly on the clients, without having to trust a server. However, existing server-side applications that involve separate database and application servers cannot be used with these systems unless they are rewritten as distributed client-side applications to work with SPORC or Depot. Many applications are not amenable to such a structure.

Disk encryption. Various commercial database products, such as Oracle's Transparent Data Encryption [Ora], encrypt data on disk, but decrypt it to perform query processing. As a result, the server must have access to decryption keys, and an adversary compromising the DBMS software can gain access to the entire data.

Trusted hardware. An alternative approach to computing over encrypted data is to rely on trusted hardware [KMC11, BS11, ABE+13]. Such approaches are complementary to CryptDB and Mylar, and could be used to extend the kinds of computations that our systems can perform over encrypted data at the server, as long as the application developer and the users believe that trusted hardware is trustworthy.

Software security. Many tools help programmers either find or mitigate mistakes in their code that may lead to vulnerabilities, including static analysis tools like PQL [LL05, MLL05] and UrFlow [Chl10], and runtime tools like Resin [YWZK09] and CLAMP [PMW+09]. However, if an attacker manages to get access to the server (e.g., perhaps by having legitimate access to the server as in the case of an employee of a cloud), the attacker can see all sensitive data. Our systems protect data confidentiality even against such situations.

◇ 8.1.2 Cryptography work

Using cryptographic tools directly to enable systems to run over encrypted data results in using encryption schemes that can support complex or arbitrary functions because systems today compute complex operations. Hence, one would use encryption schemes such as fully homomorphic encryption [Gen09, DGHV10, SS10b, BV11b, BV11a, Vai11, BGV12, GHS12a, GHS12b, LTV12, Bra12, GSW13, Hal13, HS14], general functional encryption [GKP+13a, GKP+13b, GGH+13], or generic secure multi-party computation [Yao82, GMW87, LP07, IPS08, LP09, HKoS+10, MNPS04, BDNP08, BHKR13]. As we discussed in Chapter 1 and 2, these schemes are prohibitively impractical. Our thesis constructs practical systems.

□ 8.2 Work related to a specific chapter

◇ 8.2.1 Work related to CryptDB

Data tokenization. Companies like Navajo Systems and Ciphercloud provide a trusted application-level proxy that intercepts network traffic between clients and cloud-hosted servers (e.g., IMAP), and encrypts sensitive data stored on the server. These products appear to break up sensitive data (specified by application-specific rules) into tokens (such as words in a string), and encrypt each of these tokens using an order-preserving encryption scheme, which allows token-level searching and sorting. In contrast, CryptDB supports a richer set of operations (most of SQL), reveals only relations for the necessary classes of computation to the server based on the queries issued by the application, and allows chaining of encryption keys to user passwords, to restrict data leaks from a compromised proxy.

Search and queries over encrypted data. Song et al. [SWP00] describe cryptographic tools for performing keyword search over encrypted data, which we use to implement SEARCH. Amanatidis et al. [ABO07] propose methods for exact searches that do not require scanning the entire database and could be used to process certain restricted SQL queries. Bao et al. [BDDY08] extend these encrypted search methods to the multi-user case. Yang et al. [YZW06] run selections with equality predicates over encrypted data. Evdokimov and Guenther present methods for the same selections, as well as Cartesian products and projections [EG]. Agrawal et al. develop a statistical encoding that preserves the order of numerical data in a column [AKSX04], but it does not have sound cryptographic properties, unlike the scheme we use [BCLO09]. Boneh and Waters show public-key schemes for comparisons, subset checks, and conjunctions of such queries over encrypted data [BW07], but these schemes have ciphertext lengths that are exponential in the length of the plaintext, limiting their practical applicability.

When applied to processing SQL on encrypted data, these techniques suffer from some of the following limitations: certain basic queries are not supported or are too inefficient (especially joins and order checks), they require significant client-side query processing, users either have to build and maintain indexes on the data at the server or to perform sequential scans for every selection/search, and implementing these techniques requires unattractive changes to the innards of the DBMS.

Some researchers have developed prototype systems for subsets of SQL, but they provide no confidentiality guarantees, require a significant DBMS rewrite, and rely on client-side processing [HILM02, DdVJ+03, CdVF+09]. For example, Hacigumus et al. [HILM02] heuristically split the domain of possible values for each column into partitions, storing the partition number unencrypted for each data item, and rely on extensive client-side filtering of query results. Chow et al. [CLS09] require trusted entities and two non-colluding untrusted DBMSes.

Privacy-preserving aggregates. Privacy-preserving data integration, mining, and aggregation schemes are useful [KC05, XCL07], but are not usable by many applications because they support only specialized query types and require a rewrite of the DBMS. Differential privacy [Dwo08] is complementary to CryptDB; it allows a trusted server to decide what answers to release and how to obfuscate answers to aggregation queries to avoid leaking information about any specific record in the database.

Query integrity. Techniques for SQL query integrity can be integrated into CryptDB because CryptDB allows relational queries on encrypted data to be processed just like on plaintext. These methods can provide integrity by adding a MAC to each tuple [LKMS04, SMB03], freshness using hash chains [PLM+11, SMB03], and both freshness and completeness of query results [NDSK07]. In addition, the client can verify the results of aggregation queries [THH+09], and provide query assurance for most read queries [Sio05].

Outsourced databases. Curino et al. advocate the idea of a relational cloud [CJP+11], a context in which CryptDB fits well.

◇ 8.2.2 Work related to Mylar

Mylar is the first system to protect data confidentiality in a wide range of web applications against arbitrary server compromises.

Much of the work on web application security focuses on preventing security vulnerabilities caused by bugs in the application's source code, either by statically checking that the code follows a security policy [XA06, Chl10], or by catching policy violations at runtime [Kro04, YWZK09, GLS+12]. In contrast, Mylar assumes that *any* part of the server can be compromised, either as a result of software vulnerabilities or because the server operator is untrustworthy, and protects data confidentiality in this setting.

On the browser side, prior work has explored techniques to mitigate vulnerabilities in Javascript code that allow an adversary to leak data or otherwise compromise the application [YNKM09, FSF04, ASS12]. Mylar assumes that the developer does not inadvertently leak data from client-side code, but in principle could be extended to use these techniques for dealing with buggy client-side code.

There has been some work on using encryption to protect confidential data in web applications, as we describe next. Unlike Mylar, none of them can support a wide range of complex web applications, nor compute over encrypted data at the server, nor address the problem of securely managing access to shared data.

A position paper by Christodorescu [Chr08] proposes encrypting and decrypting data in a web browser before sending it to an untrusted server, but lacks any details of how to build a practical system.

Several data sharing sites encrypt data in the browser before uploading it to the server, and decrypt it in the browser when a user wants to download the data [Sau13, Def13, Meg13]. The key is either stored in the URL's hash fragment [Sau13, Meg13], or typed in by the user [Def13], and both the key and data are accessible to any Javascript code from the page. As a result, an active adversary could serve Javascript code to a client that leaks the key. In contrast, Mylar's browser extension verifies that the client-side code has not been tampered with.

Several systems transparently encrypt and decrypt data sent to a server [PKZ11, Ras11, Cip, BKW11]. These suffer from the same problems as above: they cannot handle active attacks, and cannot compute over encrypted data at the server without revealing a significant amount of information.

Cryptocat [The13], an encrypted chat application, distributes the application code as a browser extension rather than a web application, in order to deal with active attacks [The12]. Mylar's browser extension is general-purpose: it allows verifying the code of web applications without requiring users to install a *separate* extension for each application. Cryptocat could also benefit from Mylar's search scheme to perform keyword search over encrypted data at the server.

◇ 8.2.3 Work related to multi-key search

Most of the research on searchable encryption [SWP00, Goh, BCOP04, CM05, BBO07, CGKO06, BW07, BDDY08, ZNS11, YLW11, Raj12, KPR12, CJJ$^+$13] focused on the case when the data is encrypted with the same key, and considered various aspects of the resulting cryptosystem, such as public- versus secret-key, more expressive computation such as conjunctions and disjunctions, indexable schemes, and others.

To the best of our knowledge, Lopez-Alt et al. [LTV12] is the only work considering computation over data encrypted with different keys. They design a fully homomorphic encryption (FHE) scheme in which anyone can evaluate a function over data encrypted with different keys. However, the decryption requires all the parties to come together and run an MPC protocol. Translated to our setting, this requires a client to retrieve all the keys under which the data is encrypted so the client still needs to do work proportional in the number of keys, which is what we are trying to avoid. Moreover, due to the semantic security of FHE, the server does not learn whether a document matches a keyword: it only learns the encryption of whether the document matches; therefore, the server would have to return the entire data, which is not practical.

A related scheme is the one of Bao et al. [BDDY08], who consider a setting where users have different keys but all the data is encrypted with one key and the search happens over data encrypted with one key. One cannot directly apply their scheme to the multi-key setting by creating an instance of the scheme for every key because this results in many search tokens; the reason is that the search tokens are tied to a secret different for every different key. Moreover, one requires different security definitions and security proofs when considering data encrypted under different keys with users only accessing a subset of them. Other works [CGKO06, YLW11, ZNS11, Raj12] fall in the same category of multi-user one-key schemes, and have similar properties.

CHAPTER 9

Conclusion

This dissertation shows how to protect data confidentiality even against attackers who get access to all server data. To achieve this goal, it introduces a new approach to building secure systems: building practical systems that compute on encrypted data.

To implement this approach, we identified a meta-strategy to guide building such systems. We also provided background on cryptographic resources that could be useful in implementing this strategy. We illustrated this strategy by presenting in depth the design of two novel systems: CryptDB for securing databases and Mylar for securing web applications, as well as a novel encryption scheme, multi-key search, which we designed to enable Mylar; we also presented our theoretical work on functional encryption that enables computing any function on encrypted data.

We showed that the systems we designed are practical. For example, CryptDB supports all queries from TPC-C, an industry standard benchmark for SQL, and decreases throughput by only 26% as compared to vanilla MySQL (a popular SQL DBMS). Crucially, CryptDB requires no changes to existing DBMSs, and almost no change to applications, making it easy to adopt. Moreover, we ported 6 web applications to Mylar which required changing just 35 lines of code on average; the performance overheads introduced by Mylar were modest and largely did not affect user experience.

We believe that cloud computing will gravitate towards processing encrypted data whenever possible because of its benefits: both clients and cloud providers are protected against cloud employees or hackers accessing their sensitive data or the data of their customers, respectively. CryptDB proved that such an approach can be practical and in the few years since its publication, we already started to see such a movement with Google, SAP AG, MIT's Lincoln Labs, and other companies adopting CryptDB's approach. We hope that companies will continue to adopt such an approach.

□ 9.1 Future directions

Looking forward, practical computation on encrypted data promises to address many other important problems. Here are some examples:

- *Network middleboxes processing encrypted traffic.* Network middleboxes execute a variety of useful functions, such as intrusion detection, exfiltration detection, running firewalls, WAN optimization and others. For various security reasons, many people use https connections which encrypt the content of the network packets. Hence, such encryption prevents middleboxes from analyzing the traffic. Enabling middleboxes to compute on the encrypted traffic could both maintain the confidentiality of the data packets and allow middleboxes to perform their tasks.

- *Big data analytics over encrypted data.* Big data systems can also store sensitive data, so computing on encrypted data could protect its confidentiality and enable data analytics. Due to the large amount of data, compression is crucial, but encryption and compression seem contradictory. A useful solution would enable both the space gains of compression and the confidentiality guarantees of encryption.

- *Machine learning or classification over encrypted data.* A lot of machine learning algorithms run over sensitive data, such as medical or financial, and such an approach would protect data confidentiality.

- *Data mining over encrypted genomics data.* There is an increasing amount of genomics data so performing analysis and research using this data is valuable. However, privacy concerns limit access to such data. A solution that computes on encrypted data and decrypts only allowed computation results has the potential to enables both privacy and research.

- *Integrity of server results.* Most of my work has focused on protecting data confidentiality, but a malicious server can return incorrect results to client queries. This can affect client functionality and even data confidentiality. Hence, it is useful to build systems in which clients can check efficiently the correctness of query results. Hopefully, one can come up with a meta strategy for achieving this goal similar to the one we presented in this thesis for confidentiality.

Bibliography

[ABE+13] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramaratnam Venkatesan. Orthogonal security with Cipherbase. In *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, January 2013.

[ABO07] Georgios Amanatidis, Alexandra Boldyreva, and Adam O'Neill. Provably-secure schemes for basic query support in outsourced databases. In *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, Redondo Beach, CA, July 2007.

[AEK+13] Arvind Arasu, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramaratnam Venkatesan. A secure coprocessor for database applications. In *Field Programmable Logic and Applications (FPL)*, 2013.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

[AGVW12] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. *Cryptology ePrint Archive, Report 2012/468*, 2012.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[AKSX04] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, Paris, France, June 2004.

[All14] Online Trust Alliance. Online trust alliance finds data breaches spiked to record level in 2013, 2014.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, pages 535–548, 2013.

[ARH$^+$13] Erman Ayday, Jean Louis Raisaro, Urs Hengartner, Adam Molyneaux, and Jean-Pierre Hubaux. Privacy-preserving processing of raw genomic data. In *EPFL-REPORT-187573*, 2013.

[ASS12] Devdatta Akhawe, Prateek Saxena, and Dawn Song. Privilege separation in HTML5 applications. In *Proceedings of the 21st Workshop on Usable Security (USEC)*, Bellevue, WA, August 2012.

[BBO07] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.

[BCG$^+$11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum. Program obfuscation with leaky hardware. In *ASIACRYPT*, pages 722–739, 2011.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt)*, Cologne, Germany, April 2009.

[BCO11] Alexandra Boldyreva, Nathan Chenette, and Adam ONeill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011.

[BCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT'04*, pages 506–522, 2004.

[BCS09] Adam Barth, Juan Caballero, and Dawn Song. Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2009.

[BDDY08] Feng Bao, Robert H. Deng, Xuhua Ding, and Yanjiang Yang. Private query on encrypted data in multi-user settings. In *Proceedings of the 4th International Conference on Information Security Practice and Experience*, Sydney, Australia, April 2008.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: A system for secure multi-party computation. In *CCS*, pages 17–21, 2008.

[Ben13] David Benjamin. Adapting Kerberos for a browser-based environment. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Comput er Science, September 2013.

[BFO08] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, 2008.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGJ$^+$13] Jens-Matthias Bohli, Nils Gruschka, Meiko Jensen, Luigi Lo Iacono, and Ninja Marnau. Security and privacy-enhancing multicloud architectures. In *IEEE Transactions on Dependable and Secure Computing*, 2013.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography - TCC'05*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.

[BHJP14] Cristopher Bosch, Pieter Hartel, Willem Jonker, and Andreas Peter. A survey of provably secure searchable encryption, 2014.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE SP*, pages 478–492, 2013.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Garbling schemes. *Cryptology ePrint Archive, Report 2012/265*, 2012.

[BJM08] Adam Barth, Collin Jackson, and John C. Mitchell. Securing frame communication in browsers. In *Proceedings of the 17th Workshop on Usable Security (USEC)*, San Jose, CA, July–August 2008.

[BKW11] Filipe Beato, Markulf Kohlweiss, and Karel Wouters. Scramble! your social network data. In *Proceedings of the 11th Privacy Enhancing Technologies Symposium (PETS)*, Waterloo, Canada, July 2011.

[Bla93] Matt Blaze. A cryptographic file system for UNIX. In *CCS*, pages 9–16, 1993.

[Bor13] Don Borelli. The name Edward Snowden should be sending shivers up CEO spines. *Forbes*, September 2013. http://www.forbes.com/sites/realspin/2013/09/03/the-name-edward-snowden-should-be-sending-shivers-up-ceo-spines/.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, pages 868–886, 2012.

[BS11] Sumeet Bajaj and Radu Sion. TrustedDB: a trusted hardware based database with privacy and data confidentiality. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 205–216, Athens, Greece, June 2011.

[BSK$^+$14] Anis Bkakria, Andreas Schaad, Florian Kerschbaum, Frederic Cuppens, Nora Cuppens-Boulahia, and David Gross-Amblard. Optimized and controlled provisioning of encrypted outsourced data. In *SACMAT*, 2014.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 28th IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273. Springer Berlin Heidelberg, 2011.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[CdVF$^+$09] Valentina Ciriani, Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *Proceedings of the 14th European Symposium on Research in Computer Security*, September 2009.

[CGKO06] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS*, pages 79–88, 2006.

[Che10] Adrian Chen. GCreep: Google engineer stalked teens, spied on chats. *Gawker*, September 2010. http://gawker.com/5637234/.

[Chl10] Adam Chlipala. Static checking of dynamically-varying security policies in database-backed applications. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, October 2010.

[Chr08] Mihai Christodorescu. Private use of untrusted web servers via opportunistic encryption. In *Proceedings of the Web 2.0 Security and Privacy Workshop*, Oakland, CA, May 2008.

[Cip] CipherCloud. Cloud data protection solution. http://www.ciphercloud.com.

[CJJ$^+$13] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. Cryptology ePrint Archive, Report 2013/169, 2013. http://eprint.iacr.org/.

[CJP$^+$11] Carlo Curino, Evan P. C. Jones, Raluca Ada Popa, Nirmesh Malviya, Eugene Wu, Sam Madden, Hari Balakrishnan, and Nickolai Zeldovich. Relational cloud: A database-as-a-service for the cloud. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research*, pages 235–241, Pacific Grove, CA, January 2011.

[CKVW10] Ran Canetti, Yael Tauman Kalai, Mayank Varia, and Daniel Wichs. On symmetric encryption and point obfuscation. In *TCC*, pages 52–71, 2010.

[CLS09] Sherman S. M. Chow, Jie-Han Lee, and Lakshminarayanan Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *Proceedings of the 16th Network and Distributed System Security Symposium (NDSS)*, February 2009.

[CM05] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, pages 442–455, 2005.

[CO12] Juan Camilo Corena and Tomoaki Ohtsuki. Secure and fast aggregation of financial data in cloud-based expense tracking applications. In *Journal of Network and Systems Management*, 2012.

[Coo09] Michael Cooney. IBM touts encryption innovation; new technology performs calculations on encrypted data without decrypting it. *Computer World*, June 2009.

[Cou] Transaction Processing Performance Council. TPC-C. http://www.tpc.org/tpcc/default.asp.

[DdVJ$^+$03] E. Damiani, S. De Capitani di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, Washington, DC, October 2003.

[Def13] Defuse Security. Encrypted pastebin. https://defuse.ca/pastebin.htm, September 2013.

[Des00] Anand Desai. New paradigms for constructing symmetric encryption schemes secure against chosen-ciphertext attack. In *Proceedings of the 20th Annual International Conference on Advances in Cryptology*, pages 394–412, August 2000.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *EUROCRYPT*, pages 24–43, 2010.

[Dwo08] Cynthia Dwork. Differential privacy: a survey of results. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, Xi'an, China, April 2008.

[EG] Sergei Evdokimov and Oliver Guenther. Encryption techniques for secure database outsourcing. Cryptology ePrint Archive, Report 2007/335.

[ElG84] Taher ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. 1984.

[FCM12] Luca Ferretti, Michele Colajanni, and Mirco Marchetti. Supporting security and consistency for cloud database. In *Cyberspace Safety and Security*, Lecture Notes in Computer Science, 2012.

[FCM13] Luca Ferretti, Michele Colajanni, and Mirco Marchetti. Access control enforcement on query-aware encrypted cloud databases. In *IEEE International Conference on Cloud Computing Technology and Science*, 2013.

[FCM14] Luca Ferretti, Michele Colajanni, and Mirco Marchetti. Distributed, concurrent, and independent access to encrypted cloud databases. In *IEEE transactions on parallel and distributed systems*, 2014.

[Fer06] Niels Ferguson. AES-CBC + Elephant diffuser a disk encryption algorithm for Windows Vista, 2006.

[FPCM13] Luca Ferretti, Fabio Pierazzi, Michele Colajanni, and Mirco Marchetti. Security and confidentality solutions for public cloud database services. In *The 7th International Conference on Emerging Security Information, Systems and Technologies*, 2013.

[FSF04] Robert Fischer, Margo Seltzer, and Michael Fischer. Privacy from untrusted web servers. Technical Report YALEU/DCS/TR-1290, Yale University, Department of Computer Science, May 2004.

[FZFF10] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. SPORC: Group collaboration using untrusted cloud resources. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, October 2010.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC)*, Bethesda, MD, May–June 2009.

[GGH12] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices and applications. Cryptology ePrint Archive, Report 2012/610, 2012.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GGP10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology (CRYPTO)*, Santa Barbara, CA, August 2010.

[GHH$^+$14] Patrick Grofig, Martin Haerterich, Isabelle Hang, Florian Kerschbaum, Mathias Kohler, Andreas Schaad, Axel Schroepfer, and Walter Tighzert. Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data. In *Lecture Notes in Informatics, Sicherheit*, 2014.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, 2012.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *EUROCRYPT*, pages 506–522, 2010.

[GIS$^+$10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GJPS08] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP*, pages 579–591, 2008.

[GK05] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with auxiliary input. In *FOCS*, pages 553–562, 2005.

[GKP$^+$12] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. Cryptology ePrint Archive, Report 2012/733, 2012. http://eprint.iacr.org/.

[GKP$^+$13a] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *ACM Symposium on Theory of Computing (STOC)*, 2013.

[GKP$^+$13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *International Cryptology Conference (CRYPTO)*, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GLS$^+$12] Daniel B. Giffin, Amit Levy, Deian Stefan, David Terei, David Mazières, John C. Mitchell, and Alejandro Russo. Hails: Protecting data privacy in untrusted web applications. In *Proceedings of the 10th Symposium on Operating Systems Design and Implementation (OSDI)*, Hollywood, CA, October 2012.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC*, 1982.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.

[Goh] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Volume I Basic Tools*. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.

[Goo] Google. Encrypted bigquery client. https://code.google.com/p/encrypted-bigquery-client/.

[Goo13] Google, Inc. User data requests – Google transparency report, September 2013. http://www.google.com/transparencyreport/userdatarequests/.

[GPS08] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156, 2008.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[GR07] Shafi Goldwasser and Guy N. Rothblum. On best-possible obfuscation. In *TCC*, pages 194–213, 2007.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, August 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[GZ07] Tingjian Ge and Stan Zdonik. Answering aggregation queries in a secure system model. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, Vienna, Austria, September 2007.

[Hal13] Shai Halevi. HElib - an implementation of homomorphic encryption. https://github.com/shaih/HElib, 2013. URL: https://github.com/shaih/HElib.

[HHCW12] Rene Hummen, Martin Henze, Daniel Catrein, and Klaus Wehrle. A cloud design for user-controlled storage and processing of sensor data. In *CloudCom*, 2012.

[HILM02] Hakan Hacigumus, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Madison, WI, June 2002.

[HKoS$^+$10] Wilko Henecka, Stefan K ögl, Ahmad-Reza Sadeghi, Thomãs Schneider, and Immo Wehrenberg. Tasty: Tool for automating secure two-party computations. In *CCS*, pages 451–462, 2010.

[HR03] Shai Halevi and Phillip Rogaway. A tweakable enciphering mode. In *Advances in Cryptology (CRYPTO)*, 2003.

[HS14] Shai Halevi and Victor Shoup. Algorithms in HElib. In *CRYPTO*, 2014.

[IC12] Samiul Islam and Farah Habib Chanchary. Data migration: Connecting databases in the cloud. In *ICCIT*, 2012.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591. 2008.

[KC05] Murat Kantarcioglu and Chris Clifton. Security issues in querying encrypted data. In *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, Storrs, CT, August 2005.

[KGM$^+$14] Jeremy Kepner, Vijay Gadepally, Peter Michaleas, Nabil Schear, Mayank Varia, Arkady Yerukhimovich, and Robert K. Cunningham. Computing on masked data: a high performance method for improving big data veracity. *CoRR*, 2014.

[KHG$^+$13] Florian Kerschbaum, Martin HŁrterich, Patrick Grofig, Mathias Kohler, Andreas Schaad, Axel Schrpfer, and Walter Tighzert. Optimal re-encryption strategy for joins in encrypted databases. In *Data and Applications Security and Privacy*, 2013.

[Kiq] KiqueDev. kChat. https://github.com/KiqueDev/kChat/.

[KJ14] Jens Khler and Konrad Jnemann. Securus: From confidentiality and access requirements to data outsourcing solutions. In *Privacy and Identity Management for Emerging Services and Technologies*, 2014.

[KKM$^+$] George C. Kagadis, Christos Kloukinas, Kevin Moore, Jim Philbin, Panagiotis Papadimitroulas, Christos Alexakos, Paul G. Nagy, Dimitris Visvikis, and William R. Hendee. Cloud computing in medical imaging. In *Cloud computing in medical imaging*.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2007.

[KMC11] Jayanthkumar Kannan, Petros Maniatis, and Byung-Gon Chun. Secure data preservers for web services. In *Proceedings of the 2nd USENIX Conference on Web Application Development*, Portland, OR, June 2011.

[Koh08] Eddie Kohler. Hot crap! In *Proceedings of the Workshop on Organizing Workshops, Conferences, and Symposia for Computer Systems*, San Francisco, CA, April 2008.

[KPR12] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *CCS*, pages 965–976, 2012.

[Kro04] Maxwell Krohn. Building secure high-performance web services with OKWS. In *Proceedings of the 2004 USENIX Annual Technical Conference*, Boston, MA, June–July 2004.

[KS88] Voratas Kachitvichyanukul and Bruce W. Schmeiser. Algorithm 668: H2PEC: Sampling from the hypergeometric distribution. *ACM Transactions on Mathematical Software*, 14(4):397–398, 1988.

[KS12] Vladimir Kolesnikov and Abdullatif Shikfa. On the limits of privacy provided by order-preserving encryption. In *Wiley Subscription Services, Inc., A Wiley Company*, 2012.

[KSS09] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *In Cryptology and Network Security (CANS*, 2009.

[KSS13] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. In *Journal of Computer Security*, 2013.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LCY$^+$14] Zheli Liua, Xiaofeng Chenb, Jun Yanga, Chunfu Jiaa, and Ilsun Youc. New order preserving encryption model for outsourced databases in cloud environments, 2014.

[LKMS04] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. Secure untrusted data repository (SUNDR). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 91–106, San Francisco, CA, December 2004.

[LL05] V. Benjamin Livshits and Monica S. Lam. Finding security vulnerabilities in Java applications with static analysis. In *Proceedings of the 14th Workshop on Usable Security (USEC)*, pages 271–286, Baltimore, MD, August 2005.

[LN14] Tancrede Lepoint and Michael Naehrig. A comparison of the homomorphic encryption schemes fv and yashe. Technical Report MSR-TR-2014-22, January 2014.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicio us adversaries. In *Advances in Cryptology - EUROCRYPT 2007*, pages 52–78. 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptol.*, 22:161–188, April 2009. URL: http://dl.acm.org/citation.cfm?id=1530703.1530708, doi:10.1007/s00145-008-9036-8.

[LSSD14] Loi Luu, Shweta Shinde, Prateek Saxena, and Brian Demsky. A model counter for constraints over unbounded strings. In *PLDI*, 2014.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *STOC*, pages 1219–1234, 2012.

[LW12a] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, 2012.

[LW12b] Dongxi Liu and Shenlu Wang. Programmable order-preserving secure index for encrypted database query. In *International conference on Cloud Computing (CLOUD)*, 2012.

[LW13] Dongxi Liu and Shenlu Wang. Nonlinear order preserving index for encrypted database query in service cloud environments. In *Concurrency and Computation: Practice and Experience*, 2013.

[Meg13] Mega. The privacy company. https://mega.co.nz/#privacycompany, September 2013.

[Met13] Meteor, Inc. Meteor: A better way to build apps. http://www.meteor.com, September 2013.

[MLL05] Michael Martin, Benjamin Livshits, and Monica Lam. Finding application errors and security flaws using PQL: a program query language. In *Proceedings of the 2005 Conference on Object-Oriented Programming, Systems, Languages and Applications*, pages 365–383, San Diego, CA, October 2005.

[MMA14] Eirini C. Micheli, Giorgos Margaritis, and Stergios V. Anastasiadis. Efficient multi-user indexing for secure keyword search. In *EDBT/ICDT*, 2014.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[MSL$^+$10] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Depot: Cloud storage with minimal trust. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)*, Vancouver, Canada, October 2010.

[MTY13] Tal Malkin, Isamu Teranishi, and Moti Yung. Order-preserving encryption secure beyond one-wayness. *IACR Cryptology ePrint Archive*, pages 409–409, 2013.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *STOC*, pages 351–358, 2010.

[NA14] Zubair Nabi and Atif Alvi. Clome: The practical implications of a cloud-based smart home. *CoRR*, abs/1405.0047, 2014.

[Nat11] National Vulnerability Database. CVE statistics. http://web.nvd.nist.gov/view/vuln/statistics, February 2011.

[NDSK07] Viet Hung Nguyen, Tran Khanh Dang, Nguyen Thanh Son, and Josef Kung. Query assurance verification for dynamic outsourced XML databases. In *Proceedings of the 2nd Conference on Availability, Reliability and Security*, Vienna, Austria, April 2007.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[Ope13] OpenID Foundation. OpenID. http://openid.net, September 2013.

[Ora] Oracle Corporation. Oracle advanced security. http://www.oracle.com/technetwork/database/options/advanced-security/.

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PBB09] Raluca Ada Popa, Hari Balakrishnan, and Andrew J. Blumberg. VPriv: Protecting privacy in location-based vehicular services. In *USENIX Security Symposium*, 2009.

[PBBL11] Raluca Ada Popa, Andrew J. Blumberg, Hari Balakrishnan, and Frank H. Li. Privacy and accountability for location-based aggregate statistics. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[PBC] PBC library: The pairing-based cryptography library. http://crypto.stanford.edu/pbc/.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[PKZ11] Krishna Puttaswamy, Chris Kruegel, and Ben Zhao. Silverline: Toward data confidentiality in storage-intensive cloud applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*, Cascais, Portugal, October 2011.

[PLM$^+$11] Raluca Ada Popa, Jacob R. Lorch, David Molnar, Helen J. Wang, and Li Zhuang. Enabling security in cloud storage SLAs with CloudProof. In *USENIX Annual Technical Conference (USENIX)*, 2011.

[PLZ13] Raluca Ada Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy (Oakland)*, 2013.

[PMW^{+}09] Bryan Parno, Jonathan M. McCune, Dan Wendlandt, David G. Andersen, and Adrian Perrig. CLAMP: Practical prevention of large-scale data leaks. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2009.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[PRZB11] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[PRZB12] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Processing queries on an encrypted database. In *Communications of the ACM (CACM)*, 2012.

[PSV^{+}14] Raluca Ada Popa, Emily Stark, Steven Valdez, Jonas Helfer, Nickolai Zeldovich, M. Frans Kaashoek, and Hari Balakrishnan. Building web applications on top of encrypted data using Mylar. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.

[PZ12] Raluca Ada Popa and Nickolai Zeldovich. Cryptographic treatment of CryptDB's adjustable join. In *MIT-CSAIL-TR-2012-006*, 2012.

[PZ13] Raluca Ada Popa and Nickolai Zeldovich. Multi-key searchable encryption. In *Cryptology ePrint Archive, 2013/508*, 2013.

[RAD78] R. Rivest, L. Adleman, and M. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of Secure Computation*, pages 169–177. Academic Press, 1978.

[Raj12] Remya Rajan. Efficient and privacy preserving multi user keyword search for cloud storage services. In *IJATER*, pages 48–51, 2012.

[Ras11] Fahmida Y. Rashid. Salesforce.com acquires SaaS encryption provider Navajo Systems. *eWeek.com*, August 2011.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[Sau13] Sèbastien Sauvage. ZeroBin - because ignorance is bliss. http://sebsauvage.net/wiki/doku.php?id=php:zerobin, February 2013.

[SBC^{+}07] Elaine Shi, John Bethencourt, Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *Proceedings of the 28th IEEE Symposium on Security and Privacy*, Oakland, CA, May 2007.

[SHB09] Emily Stark, Michael Hamburg, and Dan Boneh. Symmetric cryptography in Javascript. In *Proceedings of the Annual Computer Security Applications Conference*, Honolulu, HI, December 2009.

[Sho09] Victor Shoup. NTL: A library for doing number theory. http://www.shoup.net/ntl/, August 2009.

[Sio05] Radu Sion. Query execution assurance for outsourced databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 601–612, Trondheim, Norway, August–September 2005.

[Sla11] Slashdot. WordPress hacked, attackers get root access, 2011.

[SMB03] Hovav Shacham, Nagendra Modadugu, and Dan Boneh. Sirius: Securing remote untrusted storage. In *Proceedings of the 10th Network and Distributed System Security Symposium (NDSS)*, 2003.

[SS10a] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, pages 463–472, 2010.

[SS10b] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, pages 377–394, 2010.

[SSPZ12] Meelap Shah, Emily Stark, Raluca Ada Popa, and Nickolai Zeldovich. Language support for efficient computation over encrypted data. In *Off the Beaten Track Workshop: Underrepresented Problems for Programming Language Researchers*, 2012.

[SSSE14] Julian James Stephen, Savvas Savvides, Russell Seidel, and Patrick Eugster. Practical confidentiality preserving big data analysis. In *HotCloud*, 2014.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

[Sta13] Emily Stark. From client-side encryption to secure web applications. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2013.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW12] Amit Sahai and Brent Waters. Attribute-based encryption for circuits from multilinear maps. Cryptology ePrint Archive, Report 2012/592, 2012.

[SWP00] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium of Security and Privacy*, 2000.

[Tat10] Ryan Tate. Why you shouldn't trust Facebook with your data: An employee's revelations, 2010.

[Tay] Monty Taylor. MySQL proxy. https://launchpad.net/mysql-proxy.

[The12] The Cryptocat Project. Moving to a browser app model. https://blog.crypto.cat/2012/08/moving-to-a-browser-app-model/, August 2012.

[The13] The Cryptocat Project. Cryptocat. http://www.cryptocat.com, September 2013.

[THH$^+$09] Brian Thompson, Stuart Haber, William G. Horne, Tomas S, and Danfeng Yao. Privacy-preserving computation and verification of aggregate queries on outsourced databases. Technical Report HPL-2009-119, HP Labs, 2009.

[Tho11] Thoughtcrime Labs. Convergence. http://convergence.io/, 2011.

[TKK12] Katsuhiro Tomiyama, Hideyuki Kawashima, and Hiroyuki Kitagawa. A security aware stream data processing scheme on the cloud and its efficient execution methods. In *CloudDB*, 2012.

[TKMZ13] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. In *International Conference on Very Large Data Bases*, 2013.

[TLMM13] Sai Deep Tetali, Mohsen Lesani, Rupak Majumdar, and Todd Millstein. Mrcrypt: static analysis for secure cloud computations. 2013.

[TSCS13] Shruti Tople, Shweta Shinde, Zhaofeng Chen, and Prateek Saxena. AUTOCRYPT: enabling homomorphic computation on servers to protect sensitive web content. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS)*, Berlin, Germany, October–November 2013.

[Tud13] Jan Tudor. Web application vulnerability statistics, June 2013. http://www.contextis.com/files/Web_Application_Vulnerability_Statistics_-_June_2013.pdf.

[Vai11] Vinod Vaikuntanathan. Computing blindfolded: New developments in fully homomorphic encryption. In *FOCS*, pages 5–16, 2011.

[WAP08] Dan Wendlandt, David G. Andersen, and Adrian Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proceedings of the 2008 USENIX Annual Technical Conference*, Boston, MA, June 2008.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

[Wat12] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

[Wee05] Hoeteck Wee. On obfuscating point functions. In *STOC*, pages 523–532, 2005.

[XA06] Yichen Xie and Alex Aiken. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th Workshop on Usable Security (USEC)*, Vancouver, Canada, July 2006.

[XCL07] Li Xiong, Subramanyam Chitti, and Ling Liu. Preserving data privacy for outsourcing data aggregation services. Technical Report TR-2007-013, Emory University, Department of Mathematics and Computer Science, 2007.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[Yao86] Andrew C. Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

[YLW11] Yanjiang Yang, Haibing Lu, and Jian Weng. Multi-user private keyword search for cloud computing. In *CloudCom*, pages 264–271, 2011.

[YNKM09] Alexander Yip, Neha Narula, Maxwell Krohn, and Robert Morris. Privacy-preserving browser-side scripting with BFlow. In *Proceedings of the ACM EuroSys Conference*, Nuremberg, Germany, March 2009.

[YWZK09] Alexander Yip, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. Improving application security with data flow assertions. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 291–304, Big Sky, MT, October 2009.

[YZW06] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving queries on encrypted data. In *European Symposium on Research in Computer Security*, 2006.

[Zal12] Michal Zalewski. *The Tangled Web*. No Starch Press, 2012.

[ZE13] Samee Zahur and David Evans. Circuit structures for improving efficiency of security and privacy tools. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, San Francisco, CA, May 2013.

[ZNS11] Fangming Zhao, Takashi Nishide, and Kouichi Sakurai. Multi-user keyword search scheme for secure data sharing with fine-grained access control. In *ICISC*, pages 406–418, 2011.