Author:
Gui, Zichen

Title:
New Perspectives on Structured Encryption

Attacks, Constructions and Foundations

New Perspectives on Structured Encryption

Attacks, Constructions and Foundations

By

Zichen Gui

Department of Computer Science
University of Bristol

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Doctor of Philosophy in the Faculty of Engineering.

September 2021

Word count: Sixty-six thousand nine hundred and seventeen

Abstract

With the increasing volume of data generated by individuals and organisations, it becomes more and more challenging to store and process data locally. Structured encryption (STE) aims to provide an outsourced storage and query solution to this problem for structured data while preserving user privacy. This thesis focuses on two subclasses of STE, namely encrypted range queries (numerically labelled data) and searchable encryption (text-based data).

We develop a multitude of novel attacks on encrypted range queries and searchable encryption with devastating consequences on user privacy. In particular, we identify *system-wide* leakage as a new source of leakage for STE that one of our attacks can exploit. We experimentally demonstrate that all state-of-the-art STE schemes suffer from this leakage in their efficient instantiations. We devise the first searchable encryption scheme that is free from system-wide leakage and is practically efficient. Finally, we propose a new security notion for STE that aims to prevent attacks at the definitional level.

Dedication and Acknowledgements

From the bottom of my heart I would like to say thank you to Bogdan Warinschi and Oliver Johnson for your supervision. None of this research would have been possible without you. I came to Bristol as an undergraduate uncertain what I wanted to do other than having a passion for mathematics and computer science. I thought those subjects were just a bit harder than what I have learnt in my secondary school – but no. Your lectures have completely changed how I think about science and inspired me to pursue a research career. It was my honour to be supervised by Bogdan on my master thesis on the same topic which has lead to this thesis. Of course, this thesis will not have been possible without a tremendous amount of help from Oliver. Thank you for all the useful technical discussions and your help in improving my writing. You are the best supervisors!

我由衷感谢Bogdan Warinschi教授和Oliver Johnson教授对我的指导。这份研究没有你们就不可能完成。我初到布大时除了有对数学和计算机科学的热爱以外，并没有一个确切的梦想。我甚至一度以为大学的授课内容就只是高中的升级版–然而我错了。你们的授课完全改变了我对科学的认知，让我有了做科研的梦想。我有幸在大四时被Bogdan指导硕士论文，而我的博士论文就是当时的衍生。当然，没有Oliver的帮助就不会有这篇毕业论文。感谢你和我的技术交谈和你对我写作的帮助。你们是最棒的导师！

During my third year of PhD, I was recommended by Bogdan (thank you again!) to Kenny Paterson for a research internship. Despite my awful introduction, Kenny accepted me as a visitor for half a year. Kenny and his group offered some of the best research experience I could have ever imagined. In particular, Kenny, Sikhar Patranabis (a member of the group at the time) and I have worked on system-level leakage attacks and SWiSSSE, which we believe will revolutionise research in structured encryption. On top of that, the group is very friendly and we have spent a lot of time discussing everything cryptography and playing together. For that, I want to thank Kenny and all members of his group (in no particular order), including Sikhar, Benjamin, Matilda, Mia, Felix, Varun, Igors and Barbara.

在我大三时，Bogdan把我推荐给了Kenny做研究实习。尽管我当时的自我介绍烂透了，Kenny还是诚心地接受了我。在Kenny组里的时候，我经历了最棒的研究体验。其中，Kenny、Sikhar和我发现了system-level leakage attacks，设计了SWiSSSE。我们相信，这两个成果会给structured encryption的研究带来革命。另外，Kenny的研究小组里的人都很友善。在那边的时候，我们一起讨论了很多密码学相关的内容，也一起玩了很多东西。为此，我要感谢Kenny和他小组的所有人：Sikhar, Benjamin, Matilda, Mia, Felix, Varun, Igors and Barbara。

I want to express my gratitude towards Alexandra (Sasha) Boldyreva, who has spent her free time reading the chapter on foundations and discussing the issues with Bogdan

and me. I hope that the results in the chapter will lead to a powerful paper which will reshape how we think about leakage.

我要感谢Alexandra (Sasha) Boldyreva利用自己的闲暇时间来读foundations的章节，并与Bogdan和我讨论相关的问题。我希望这个章节里的成果能变成一篇改变我们思考leakage方式的强力论文。

My PhD journey is certainly not a full smooth ride. When I was at lows, it was my friends and colleagues who cheered me up. For that, I thank all of you (in no particular order): Joey, Carolyn, Bin, Jake, Maria, Ben, Miranda, Thinh, Sarah, Dragos, Arnab, Yan, Rebecca and many more.

我的博士路途并非一路顺风。当我失落的时候，是我的朋友和同事让我有动力再次出发。为此，我要感谢所有帮助过我的人：Joey, Carolyn, Bin, Jake, Maria, Ben, Miranda, Thinh, Sarah, Dragos, Arnab, Yan, Rebecca等。

Finally, I want to thank my family for their love and support. Thank you for raising me and it is now my turn to make you proud!

最后，我要感谢我家人对我的爱和支持。感谢你们把我抚养长大。我希望现在的我能让你们自豪！

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: .....

Contents

Chapter 1

Introduction

Structured encryption aims to provide a way of outsourcing a database to an untrusted server while preserving the privacy of the user and supporting as many search functionalities as possible. Given the boom of cloud services and e-commerce, encrypted search sounds like a perfect solution to individuals, organizations and businesses. However, despite years of research, we see very limited adoptions of it in the real world, which begs the question: why so? In this chapter, we overview potential answers to this question and outline how our work can bridge the gap between the research community and the real world.

Contents

1.1 Structured Encryption

There is no denial that we are living in an age of data. According to World Economic Forum [60], for 2019, every day, there were 500 million tweets sent, 294 billion emails sent, 5 billion searches were made, and 65 billion WhatsApp messages were sent – and those numbers are still growing [161].

There is a multitude of ways we benefit from the data we generated, from health monitoring with wearable devices to self-driving cars, from quality-of-life improvements with internet of things (IoT) devices to better user experience on the Internet. In recent years, the idea of data-driven business has been popularized and it has benefited businesses of all sizes tremendously. For example, the maintenance and logistics cost can be reduced with big data, products are shaped by the feedbacks from users, and advertisements are personalised to increase sales. No wonder, the global big data and business analytics market was valued at 189.1 billion U.S. dollars in 2019 and is forecasted to grow to 274.3 billion U.S. dollars by 2022 [131].

However, handling the enormous amount of data does not come without challenges. For individuals and small and medium businesses, they may not have the right knowledge or tools to gather the data they need or analyse it. It is also possible that it is impractical for them to store the data locally. This spurred the growth of the cloud storage and computing industry. According to Microsoft, 78% of small businesses would have fully adopted cloud computing by 2020 [129]. Amazon, Apple, Google, IBM, Microsoft, just to name a few key players in the industry, offer a wide range of services in this category, including infrastructure-as-a-service, platform-as-a-service, packaged software-as-a-service, email, file hosting, collaborative document edition, and many more.

This raises concerns with data privacy as the user no longer holds onto the data. In terms of storage, it is not always possible to encrypt the data as the user may want to use the cloud service to manipulate or analyse the data at the same time. Since the cloud service provider has the data in plain, he can learn as much, if not more from the data as the user. The user has no way other than to trust the service provider if he opts for the service. Even then, there can be security vulnerabilities in the service or malicious employers in the service provider that will lead to data breaches [173].

Structured encryption has therefore been developed as a mechanism for dealing with these privacy concerns for *structured data* – data in a predefined format (e.g. JSON). On a high level, structured encryption provides a way to outsource data storage and queries to a cloud server, while preserving the privacy of the user and efficiency of the processes. In a perfect world, it should meet all the needs of the user, including data storage, database logging, database auditing, data analysis, and so on. It should also incur minimal overhead over the unencrypted counterpart. There is a long way we need to go before getting to the perfect world. We see in this thesis that constructing a secure and efficient scheme with basic functionalities (e.g. single keyword search) is already challenging. We focus on two of the fundamental query types for a database. The first query type is single keyword search – given a set of text-based documents, a single keyword search on a keyword returns all documents associated to the keyword. The second query type is range query – given a set of documents each associated to a value, a range query on range a to b returns all documents with values between a and b.

The setup of structured encryption is shown in Figure 1.1. Here, there are two parties, namely a user and a server. The user uploads a database to the server and he can

perform queries on the database with the help of the server. In the simplest setting, the server supports a single type of query on the encrypted data, for instance, simple keyword search queries on a text-based database or range queries on numerical data.

In terms of security guarantees, a man in-the-middle and even the server should not be able to learn any important information about the plaintext database and the queries. For example, for a text-based database that supports single-keyword queries, one may require that an honest-but-curious server should not learn the keywords associated to the queries and the keywords associated to the documents in the database. Depending on the security requirements, one may give the adversary additional powers. For instance, one may require the above to be satisfied even if the adversary is allowed to inject malicious documents into the encrypted database.

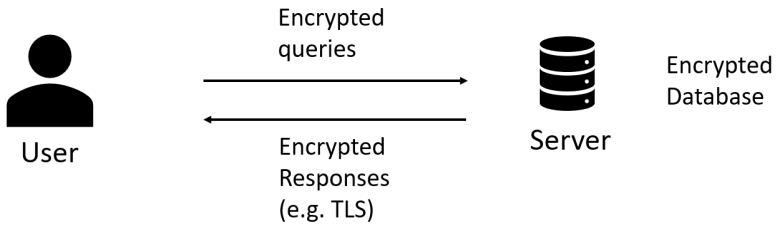


Figure 1.1: Basic setup of structured encryption.

In this thesis, we only consider the basic scenario mentioned above. There are many ways structured encryption can be extended. For example, instead of having a single user, there can be multiple users. It is possible to have refined roles too, such as having a database owner who possesses the data, a group of users with different privileges, and multiple servers that are responsible for different functionalities. One can design structured encryption schemes for other types of queries, such as SQL queries and graph-based queries, too. Structured encryption can also be extended to support normal database functionalities, such as database audition, (encrypted) backlogging, query cacheing, and many more.

1.2 Development of Structured Encryption

This section provides a high level introduction to the development of structured encryption, see Chapter 4 for more detail.

1.2.1 Early Constructions and Security Notions

Structured encryption was first studied in the literature in the context of single-keyword search on text data, which later became known as *searchable (symmetric) encryption*. The first scheme of this kind was proposed by Song, Wagner and Perrig [166]. The construction (and its security notion) is a special symmetric encryption scheme on the data itself where given a *token* of a keyword (a token is a string which the server can use to execute the query), the server can search over the encrypted data to recover the locations of the keyword. However, with a lack of additional data structures, the search process has to scan over the entire data, which makes the query complexity linear in the size of the database and hence, not scalable.

As pointed out by Goh [82] and Chang and Mitzenmacher [39] later, the security notion in [166] is insufficient for searchable encryption, as the definition does not capture security of the tokens, meaning that a scheme that is secure as symmetric encryption may still leak information about the queries through the tokens. For example, the tokens may be computed using a public function, and the server can use brute force to recover the keywords associated to the tokens. The authors proposed new security notions to fix the problem and new constructions that are secure with respect to their security notions.

1.2.2 Modern Security Notion

The modern security notion for searchable encryption was proposed by Curtmola et al. in [53]. The notion is simulation-based, and captures what can be inferred from the execution of a scheme by a *leakage profile* (see Section 4.3). A security proof in this notion means that an adversary cannot learn more than what the leakage profile specifies. The notion is later generalised to structured encryption by Chase and Kamara [40] and extended to the universal composability framework by Kurosawa and Ohtaki [113].

Many works have introduced additional security properties on top of the standard security notion [53, 40]. This includes forward security [25], backward security [29], volume-hiding [149], d-private access pattern [41], and so on.

1.2.3 Modern Constructions

Structure-only Encryption vs End-to-End Encryption. There are two main approaches to designing a structured encryption scheme. The first approach is to design a specialised encrypted data structure for which the user can retrieve the encrypted *indices* of the actual data. And the retrieval of the actual data is supported by primitives such as private information retrieval or oblivious random-access memory (see Section 1.3.1) or hosted on a second trusted server. The second approach is to design an end-to-end scheme, meaning that the specialised encrypted data structure supports retrieval of encrypted data directly. The majority of the schemes in the literature, for example [53, 36, 35, 97, 116], take the first approach, as it leads to more efficient constructions for retrieval of encrypted indices.

Structured Encryption for Various Data Types. There are three main classes of structured encryption studied in the literature, namely searchable encryption, encrypted range queries and graph-based encryption.

Searchable encryption includes variants of keyword search on encrypted text-based database. It has received a lot of attention in the past few years. There is a line of work [53, 35, 41, 98, 149] focusing on improving security and efficiency of searchable encryption. There are also works focusing on dynamism [100, 25, 109] and public-key setting [22, 199, 45]. Many other works tried to expand on the functionality of searchable encryption. For instance, [19, 81] studied fuzzy keyword search, and [36, 146, 116] studied boolean queries.

Encrypted range queries is a primitive to perform range queries on encrypted labelled data. It has been studied in many settings [163, 31, 58, 202, 57]. It may also be instantiated by a property-preserving encryption scheme, as demonstrated in [20, 151,

103, 46, 158].

Graph-based encryption is a collection of structured encryption schemes for graph-based queries such as k-nearest neighbour query and shortest path query. It was first studied in [40] by Chase and Kamara, and followed by many works [47, 108, 117].

1.2.4 Leakage-abuse Cryptanalysis Against Structured Encryption

As we mentioned earlier, security of a structured encryption scheme is parametrised by a leakage profile. This leakage profile quantifies the information leakage which an adversary can obtain by observing the operations of the structured encryption scheme. For instance, the leakage profile may include search pattern (if two encrypted queries are the same query) and query response length pattern (how many data items is returned by the query), which were believed to be "harmless".

(Un)surprisingly, leakage-abuse attacks have shown that many of these benign leakage profiles can be exploited to recover private information about the database and/or the queries. These include attacks against searchable encryption [94, 33, 155, 17, 143], attacks against encrypted range queries [33, 115, 87, 88, 89, 111], and attacks against other primitives that can be used to build a structured encryption scheme [135, 139].

1.3 Related Work

The data retrieval problem has been studied in different settings for many years. In some sense, structured encryption can be viewed as a specialised data retrieval primitive that supports *searching* as opposed to retrieving an element by its position.

In this section, we provide an overview of the other data retrieval primitives in the literature. We show how they can be used to build structured encryption and argue that the natural constructions arise from the primitives are inefficient.

1.3.1 Data Retrieval Primitives

Private Information Retrieval. Private Information Retrieval (PIR) [49] is a primitive that allows a user to retrieve an element from a server without letting it learn which element was accessed. In the standard setting, the database is not encrypted, and hence, known by the server. Recent constructions [6, 52] have shown that data retrieval in PIR can be achieved in sub-linear time and there are use-cases where they can be efficiently deployed.

Homomorphic encryption. Homomorphic encryption (FHE) was introduced by Ronald Rivest, Leonard Adleman, and Michael Dertouzos [157] as a tool for a server to compute on encrypted data without learning the underlying plaintexts. Since the seminal paper by Gentry [76] on fully homomorphic encryption, a lot of breakthroughs [181, 78, 30, 107, 106] have been made in the field. FHE can be used as a building block for data retrieval primitives such as PIR [6, 52] and private database query systems [23, 12].

Oblivious random-access memory. Oblivious random-access memory (ORAM) was

proposed by Goldreich and Ostrovsky in [84] as a primitive to access elements of a memory obliviously, meaning that the access pattern of different elements cannot be distinguished by an attacker. There has been many follow-up works in this direction, including works on multi-server ORAM [170, 123], ORAMs with server computation [195, 61, 74] and Oblivious Data Structures (ODS) [190].

1.3.2 Structured Encryption from Other Primitives

Certain classes of structured encryption can be realised with generic data retrieval primitives listed above. For example, one can build a structured encryption scheme for single keyword search by encrypting the database and its index (a map between the keywords to the document identifiers, the documents corresponding to the document identifiers must contain the given keyword), applying a suitable PIR scheme on top, and outsource the resultant database to a server.

Solutions like this certainly meet the functionality requirements of structured encryption, but they are typically too slow for practical applications. There are two main sources of inefficiency. Firstly, the underlying data retrieval primitives are built for the strongest possible security guarantees, and they are necessarily "inefficient". For example, the theoretical lower bound on bandwidth overhead for ORAM is proven to be $\mathcal{O}(\log(n))$ where n is the number of data elements [84]. For some real-world applications, this kind of logarithmic overhead is unacceptable.

The second source of efficiency comes from the nature of the primitives used. For example, ORAM is built for accessing one data element at a time. On the other hand, structured encryption is built for accessing a list of search results for each query. So if one was to use ORAM to retrieve all of the search results, the overall bandwidth overhead will be $\mathcal{O}(m \log(n))$, where m is the number of data elements in the search result. For a large m, i.e. anything on the order of $\mathcal{O}\left(\frac{n}{\log(n)}\right)$, the bandwidth will be on the order of $\mathcal{O}(n)$ and it is better to return the entire database in that case. PIR suffers a similar problem in terms of computation and communication overheads. In short, the other primitives are built for single element data retrieval and are not optimised for multi-element data retrieval. The overhead as a result of this can be significant, making the final scheme inefficient and impractical.

We show a concrete overhead comparison for different constructions of structured encryption in Section 6.1.

1.4 Challenges of Structured Encryption and Our Contributions

Understanding Leakage-abuse Attacks. There is no doubt that certain leakage can be problematic given the plenitude of attacks. However, our understanding of these attacks is very limited. In particular, the optimality of many attacks [94, 33, 155, 17] is still unknown. This implies that even if we can show that a scheme is secure with respect to one of these attacks, it does not mean that the scheme is protected from *all* leakage-abuse attacks with the same attack goal.

As the community is more aware of these leakage-abuse attacks, many schemes with

countermeasures [33, 26, 99, 41, 149, 198, 57] have been proposed. However, the majority of the attacks in the literature only work on unperturbed leakage, and there is no obvious way to extend these attacks to attack the schemes with countermeasures.

Most importantly, all of the attacks make arbitrary choices of auxiliary information and query distribution. This is understandable as it is hard to find real-world auxiliary datasets and there has not been any formal studies on query distribution. However, this gives practitioners a reason to dismiss some of the attacks as unrealistic attacks. Indeed, attacks such as [94, 33, 17] assume that the attacker has access to the target database, which is hard to justify in practice.

This thesis addresses all of the issues above. In Chapter 5, we study leakage-abuse attacks on encrypted range queries under many different scenarios. We show that even for realistic query distributions and schemes with countermeasures, our attacks are sufficient to breach privacy. In Chapter 6, we demonstrate the same idea on searchable encryption. Along the way, we identify a new source of leakage which we call *system-level* leakage, and argue that all structured-only schemes with an efficient data retrieval phase have this leakage.

Structure-only Encryption vs End-to-End Encryption. As we have pointed out earlier, there are two main approaches to designing a structured encryption scheme. The first approach designs schemes for retrieval of encrypted indices and encrypted documents separately, whereas the second approach does both at the same time. The majority of the schemes in the literature take the first approach and they are known as structure-only schemes. There are a handful of schemes taking the second approach and they are referred to as end-to-end schemes.

As the structured-only schemes often only specifies how encrypted indices should be retrieved, it raises the question of how retrieval of actual data should be handled. There are two main proposed solutions to this question. Firstly, one may use generic tools such as ORAM (see Section 1.3.1) to achieve this. But these tools are typically very expensive in terms of communication overhead and/or storage overhead, leading to an overall impractical structured encryption scheme (see Section 6.1). Secondly, one may use a second trusted server for the job, but one can argue that in that case, one should just use the trusted server for an unencrypted database.

In addition to the two proposed solutions above, we note that the structure-only schemes are capable of retrieving actual data, except that they are too inefficient to be practical (see Section 6.1.1). This is because the techniques used by these schemes, for instance, document identifier duplication and full padding, do not scale on actual data.

In that light, we are forced to design end-to-end structured encryption schemes from scratch. There have been attempts at doing this, but to the best of our knowledge, all of the end-to-end constructions [41, 57] are either inefficient or broken by leakage-abuse attacks. In Chapter 7, we propose the first end-to-end searchable encryption schemes that are scalable and resilient to all known leakage-abuse attacks. To fully test the limits of our constructions, we developed a new leakage-abuse attack in Section 7.5. The attack is further refined which leads to the results in Chapter 6.

Preventing Leakage-abuse Attacks via a Quantitative Security Notion. Advances in leakage-abuse attack forced many researchers to take a defensive approach in designing new schemes. For instance, leaking query response volume is known to

be dangerous in many settings [135, 143], so new schemes [149, 98] are designed to suppress the leakage completely. The techniques used by these schemes either leads to an inefficient scheme or a lossy one (some of the structures in the plaintext database are lost after encryption). We demonstrate this in Section 6.1.

On the other hand, we believe that leakage lies at the heart of structured encryption. Instead of the all-or-nothing approach taken by many, we believe that we should suppress leakage moderately to the point that leakage-abuse attacks are no longer viable and the resultant scheme can remain efficient. The current security definition does not allow us to achieve this goal, as leakage is a part of the security notion, and the security of a scheme has to be assured via leakage cryptanalysis. The latter process can be problematic in itself, as security with respect to a particular leakage-abuse attack does not imply security against *all* leakage-abuse attacks.

There is a need for a new security notion which one can prove quantitatively the security of a scheme with respect certain classes of leakage-abuse attack. That is, instead of defining security by whether an attack can succeed, we need a security notion for which we can show an upper bound of the damage an attack can do. A scheme can be secure under our notion even if there is a small chance for an attacker to succeed. We explore this idea in Chapter 8.

Research efforts in understanding leakage vs efficiency trade-offs. As discussed before, the main goals of this thesis is to understand the leakage versus efficiency trade-off in structured encryption and propose constructions that are as efficient as possible with acceptable leakage. This is not the first time such trade-off has been considered in the literature. In fact, similar efforts have been made in the literature of storage systems [197], PIR [178], ORAM [185], oblivious algorithms [38] and many more. Thus, this thesis can be seen as a continuation of work in this direction. In Section 7.2, we give more details on how our work is connected to the others in the literature.

1.5 Organization of the Thesis

In this thesis, we deal with all of the challenges discussed above. It is organised as follows. The mathematical and cryptographic foundations are laid in Chapter 2 and 3 respectively. Chapter 4 gives a detailed overview of structured encryption. Chapter 5 and 6 explore attacks on encrypted range queries and searchable encryption respectively. This motivates Chapter 7 which devises the first scalable and secure end-to-end searchable encryption schemes. Chapter 8 proposes a new security notion for structured encryption which aims to prevent leakage-abuse attacks at a definitional level. Finally, Chapter 9 offers a discussion on the results from previous chapters and concludes the thesis.

1.6 Published Results

Section 5.3.1 to 5.3.5 are based on the [89] paper. The other results in Chapter 5 will be published in a paper in the future. Chapter 6 will appear in IEEE Symposium on Security and Privacy 2023 and the ePrint version is available online [90]. Chapter 7 uses material from [91]. Finally, the results from Chapter 8 are still under active research

and will be published as a paper in the future.

Chapter 2

Background I: Mathematical Foundation

This chapter serves as an introduction to the mathematical ideas, notions and tools used in the later chapters. Section 2.1 introduces probability theory and statistics. Section 2.2 outlines the key ideas in complexity theory. Section 2.3 presents the common notations used in the later chapters. For more detailed studies of the subjects, see [159, 160, 96, 51] respectively.

Contents

2.1 Probability Theory and Statistics

Probability theory is the studies of the likelihood of an event occurring from a pool of possible events. For example, in a coin-tossing experiment, one may be interested in the likelihood of the coin landing on its head, and determine if the coin is unbiased. Here, there are two possible outcomes: head (H) or tail (T). The set $S = \{\mathrm{H}, \mathrm{T}\}$ is called the *sample space*, and the subsets of the set are called the *events*.

The concept of probability is used throughout this thesis (and cryptography in general) to argue about security of schemes. See Section 3.1.2 for an example of a typical security proof.

2.1.1 Axioms of Probability

One way of defining the probability of an event E is in terms of its relative frequency. Let $n(E)$ be the number of times event E has occurred in the first n independent trials of the experiment, then we can define the probability of event E happening as:

$$\mathsf{Pr}\left[E\right] := \lim_{n\to\infty} \frac{n(E)}{n}.$$

This definition is certainly intuitive, but there is a major drawback: how do we know that $\mathsf{Pr}\left[E\right]$ converges? In modern probability theory, a different approach is taken. Instead of assuming the convergence of $\mathsf{Pr}\left[E\right]$, we start with a simpler set of *axioms* [159], and then show that $\mathsf{Pr}\left[E\right]$ does converge in some sense.

Definition 2.1 (Axioms of Probability)**.** Consider an experiment whose sample space is S. For each event E of the sample space S, we assume that a number $\mathsf{Pr}\left[E\right]$ is defined and satisfies the following three axioms:

1. $0 \leq \mathsf{Pr}\left[E\right] \leq 1$.
2. $\mathsf{Pr}\left[S\right] = 1$.
3. For any sequence of mutually exclusive events $E_1, E_2, \ldots$ (i.e. the intersection of events $E_i \cap E_j = \emptyset$ for all $i \neq j$),

$$\mathsf{Pr}\left[\cup_{i=1}^{\infty} E_i\right] = \sum_{i=1}^{\infty} \mathsf{Pr}\left[E_i\right].$$

We refer to $\mathsf{Pr}\left[E\right]$ as the probability of the event E.

2.1.2 Conditional Probability

Sometimes it is useful to look at multiple events together as a single event. For example, one may be interested in the probability that three dice give a sum of 8. In a more complicated setting, one may also be interested in the probability that three dice give a sum of 8 given that one of the dice gives 3. Probabilities of this form are known as conditional probabilities.

Definition 2.2 (Conditional Probability)**.** Let E and F be two events and $\Pr[F] > 0$. The conditional probability of E given F is defined as

$$\Pr[E \mid F] := \frac{\Pr[E \cap F]}{\Pr[F]}.$$

This gives us a way to define independent events, i.e. events that do not affect each other's probability in a conditional probability.

Definition 2.3 (Independent Events)**.** Let E and F be two events. We say that E and F are independent if $\Pr[E \mid F] = \Pr[E]$, or equivalently

$$\Pr[E \cap F] = \Pr[E] \Pr[F].$$

A useful theorem involving conditional probability is the law of total probability.

Theorem 2.1 (Law of Total Probability)**.** If events $\{E_1, E_2, \ldots\}$ is a countably finite partition of a sample space Ω (i.e. $\Omega = \cup_i E_i$ for mutually exclusive E_i's), then for any event F of the same sample space,

$$\Pr[F] = \sum_i \Pr[F \cap E_i].$$

In other words,

$$\Pr[F] = \sum_i \Pr[F \mid E_i] \Pr[E_i].$$

2.1.3 Probability Spaces

We define probability spaces in this section. A probability space is a mathematical construct that provides a formal model of a random process or "experiment". This then allows us to define random variables later. To begin with, we need to define σ-algebra.

Definition 2.4 (σ-algebra)**.** Let X be a set and $\mathcal{P}(X)$ be the power set of X. Then a subset $\Sigma \subseteq \mathcal{P}(X)$ is called an σ-algebra if the following conditions are satisfied:

1. X is in Σ and X is considered to be the universal set in the following context.
2. Σ is closed under complementation: if A is in Σ then so is its complement $X \backslash A$.
3. Σ is closed under countable unions: if $A_1, A_2, \ldots \in \Sigma$ then $A = A_1 \cup A_2 \cup \ldots \in \Sigma$.

Elements of the σ-algebra are called measurable sets. An ordered pair (X, Σ), where X is a set and Σ is a σ-algebra over X, is called a measurable space. A function between two measurable spaces is called a measurable function if the preimage of every measurable set is measurable.

Definition 2.5 (Probability Space)**.** A probability space is a triple $(\Omega, \mathcal{F}, \Pr)$ consisting of:

- The sample space Ω – an arbitrary set that is non-empty,

- The event space $\mathcal{F}$ – a σ-algebra,
- The probability measure $\mathsf{Pr} : \mathcal{F} \to [0, 1]$ – a function on $\mathcal{F}$ such that:
  - Pr is σ-additive: if $A_1, A_2, \ldots \subseteq \mathcal{F}$ is a countable collection of pairwise disjoint sets, then $\mathsf{Pr}\left[\cup_{i=1}^{\infty} A_i\right] = \sum_{i=1}^{\infty} \mathsf{Pr}\left[A_i\right]$,
  - the measure of entire sample space is equal to one: $\mathsf{Pr}\left[\Omega\right] = 1$.

The formalism of probability space appears again in Chapter 8 where the low-level details are required. In other cases, we implicitly assume the use of the standard probability space where the event space $\mathcal{F}$ is the power set of the sample space Ω.

2.1.4 Random Variables

We are now ready to define random variables. A random variable is built on top of the events in a probability space in the sense that its random behaviour is determined by the probability space just as before, but the *outcomes* of the random variable are computed from the events.

Definition 2.6 (Random Variable)**.** Let $(\Omega, \mathcal{F}, \mathsf{Pr})$ be a probability space and $(E, \mathcal{E})$ a measurable space. Then an $(E, \mathcal{E})$-valued random variable is a measurable function $X : \Omega \to E$.

If $E = \mathbb{R}$, we say that X is real-valued.

Definition 2.7 (Discrete Random Variables)**.** We say that X is a discrete random variable if the image of the random variable X is countable. Real-valued discrete random variables can be characterized by their *probability mass functions* (PMFs). Let X be a real-valued discrete random variable, then its PMF is written as $\mathsf{Pr}\left[X = x\right]$. This is a shorthand for $\mathsf{Pr}\left[\omega \in \Omega \mid X(\omega) = x\right]$ which is widely accepted in the literature.

For example, for a fair coin, we may assign head to 1 and tail to 0. The random variable X which represents the outcome of a coin toss experiment can then be written as:

$$\mathsf{Pr}\left[X = x\right] = \begin{cases} 0.5 & \text{if } \mathrm{x} = 1, \\ 0.5 & \text{otherwise.} \end{cases}$$

Discrete random variables are typically used in cryptography to describe the distribution of inputs and outputs of cryptosystems (e.g. binary strings or elements of a finite field) as they are finite in nature. See Section 3.1.1 for an example of how discrete random variables are used in cryptography.

Definition 2.8 (Cumulative Distribution Function (CDF))**.** An equivalent way to characterize a random variable is through its *cumulative distribution function*. As opposed to a PMF, a CDF uses the probability of a random variable less or equal to a value to characterize it, i.e. $\mathsf{Pr}\left[X \leq x\right]$.

Common discrete random variables include Bernoulli random variables, binomial random variables, Poisson random variables, geometric random variables, hypergeometric random variables and so on [159]. A special property of these random variables is that

they are classes of random variables that are parametrized by one or more parameters. For example, a Bernoulli random variable is parametrized by a probability p and a binomial random variable is parametrized by a natural number n and a probability p.

Although informative, a PMF is often not straightforward enough in describing a random variable. That is why two numbers, namely expectation and variance, are often used to summarize a random variable. The expectation of a random variable describes the average outcome expected from a random variable and the variance of a random variable describes how far away one expects the outcome to deviate from the expectation.

Definition 2.9 (Expectation and Variance of a Discrete Random Variable)**.** Let X be a discrete random variable. Then the expectation of X is defined to be

$$\mathbf{E}[X] = \sum_x x\mathsf{Pr}[X = x].$$

The variance of X is defined to be

$$\begin{aligned}\mathsf{var}[X] &= \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[X^2\right] - \left(\mathbf{E}[X]^2\right).\end{aligned}$$

Definition 2.10 (Continuous Random Variables)**.** We say that X is a continuous random variable if the image of the random variable X is uncountable. A real-valued continuous random variable can be described by a probability density function (PDF) or a cumulative distribution function (CDF).

Common continuous random variables include continuous uniform distribution, Laplace distribution, normal distribution and so on. Similar to the discrete case, these random variables are parametrized by one or more parameters. For example, a continuous uniform distribution is parametrized by two real numbers a and b, and Laplace distribution is parametrized by a real number μ and a positive real number b.

In this thesis, continuous random variables are used to approximate discrete random variables as the earlier are typically easier to work with. Continuous random variables are heavily used in Section 5.3.1, 6.4 and 7.5.

Similar to the discrete case, we can define the expectation and variance of a continuous random variable as follows.

Definition 2.11 (Expectation and Variance of a Continuous Random Variable)**.** Let X be a continuous random variable with PDF $f(x)$. Then the expectation of X is defined to be

$$\mathbf{E}[X] = \int_x xf(x).$$

The variance of X is defined to be

$$\mathsf{var}[X] = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right].$$

2.1.5 Central Limit Theorem (CLT)

The central limit theorem states that for 'well-behaved' independent random variables, their normalised sum tends toward a normal distribution regardless of their original

distributions. This theorem is a key concept in probability theory as it allows many problems involving other types of distributions to be solved via normal approximation. There are many variants of CLT concerning different types of random variables. The variant that is stated below requires the random variables to be independent, but not necessarily identically distributed. The variant of CLT is known as Lyapunov CLT [15].

CLT is used in Section 6.4 and 7.5 as an approximation technique which speeds up the computation.

Theorem 2.2 (Lyapunov CLT)**.** Suppose $X_1, \ldots, X_n$ is a sequence of independent random variables, each with finite expected value μ_i and variance σ_i^2. Define

$$s_n^2 = \sum_{i=1}^{n} \sigma_i^2.$$

If for some $\delta > 0$, Lyapunov's condition

$$\lim_{n\to\infty} \frac{1}{s_n^{2+\delta}} \sum_{i=1}^{n} \mathbf{E}\left[|X_i - \mu_i|^{2+\delta}\right] = 0$$

is satisfied, then a sum of $\frac{X_i - \mu_i}{s_n}$ converges in distribution to a standard normal random variable, as n goes to infinity:

$$\frac{1}{s_n} \sum_{i=1} n\left(X_i - \mu_i\right) \xrightarrow{d} N(0,1).$$

2.1.6 Parameter Estimation

In the real world, we are often interested in the reverse problem: given the data, what can we say about the random variable? If the random variable can be approximated by a known distribution, and we can recover the parameter of that random variable, then we can use our model to predict the future, or to infer some hidden underlying causes.

Bayesian Parameter Estimation. One of the approaches, known as the Bayesian parameter estimation, relies on the famous Bayes' theorem.

Theorem 2.3 (Bayes' Theorem)**.** *Let E_1 and E_2 be two events where $Pr[E_2] \neq 0$. Then*

$$Pr[E_1 \mid E_2] = \frac{Pr[E_2 \mid E_1]\, Pr[E_1]}{Pr[E_2]}.$$

By treating the parameter (say θ) of a random variable (say X) as a random variable itself, we can interpret Bayes' theorem as follows. If we treat event E_1 as a particular value of θ and event E_2 as the samples from the random variable X, then the left-hand side of the equation can be interpreted as the probability distribution of the parameter given the samples. The right-hand side then rewrites it into three probabilities:

- $\mathsf{Pr}[E_2 \mid E_1]$: the probability of observing the data given a particular value of $\theta \in E_1$.
- $\mathsf{Pr}[E_1]$: the probability of observing the particular value of $\theta \in E_1$.

- $\mathsf{Pr}[E_2]$: the probability of observing the data.

The first probability can be computed explicitly as we know everything about X in the conditional probability. The second probability can be computed if we assume a distribution on θ. The only thing that cannot be explicitly computed (or at least, hard to be), is $\mathsf{Pr}[E_2]$. However, just knowing that $\mathsf{Pr}[E_1 \mid E_2] \propto \mathsf{Pr}[E_2 \mid E_1]\,\mathsf{Pr}[E_1]$ allows us to compare the likelihoods of different parameters of X.

Frequentist Parameter Estimation. Another school of statistical inference, known as frequentist inference [136], tackles the parameter estimation problem differently. Instead of treating the unknown parameter as a random variable, the parameter is believed to be fixed and hidden. In a parameter estimation problem, a frequentist inference outputs a single solution and a confidence interval associated to it.

A common method to estimate the parameter of a random variable for frequentists is maximum likelihood estimation. The idea is to find the parameter which maximises the likelihood function:

$$\hat{\theta} = \arg\max_{\theta} \mathsf{Pr}[S \mid \theta],$$

where θ is the parameter of a random variable, S is the collection of samples and $\hat{\theta}$ is the maximum likelihood estimation of the parameter.

We used frequentist parameter estimation in Chapter 6 and 7.

2.2 Complexity Theory

Thanks to advances in computer science and engineering, we are able to solve computational problems such as speech recognition, machine learning and black hole simulation we would only dare to dream of a few decades ago. We may be tempted to believe that we will be able to solve *any* computational problems if we can make our computers fast enough. Unfortunately, this does not appear to be true. There are many problems that can be solved theoretically, but the known algorithms are completely impractical. This is not all bad news, as modern cryptography relies on a subset of these hard problems which are *hard* to solve but *easy* to verify whether a solution is valid.

This section provides an overview of complexity theory and outlines its use in cryptography.

2.2.1 Deterministic Turing Machines

Before discussing the different classes of problems, we need to establish a formal model of *computation*. The classical model of computation is called the deterministic Turing machine (DTM) [96] which is presented below.

Definition 2.12 (Deterministic Turing Machine)**.** A deterministic Turing machine consists of:

1. a finite alphabet Σ containing the blank symbol $*$;

2. a 2-way infinite tape divided into squares, one of which is the special starting square. Each square contains a symbol from the alphabet Σ. All but a finite number of the squares contain the special blank symbol $*$, denoting an empty square;

3. a read-write head that examines a single square at a time and can move left ($\leftarrow$) or right ($\rightarrow$);

4. a control unit along with a finite set of states Γ including a distinguished starting state s_0 and a set of halting states F.

The computation of a DTM is controlled by a transition function:

$$\delta : \Gamma \times \Sigma \rightarrow \Gamma \times \Sigma \times \{\leftarrow, \rightarrow\}.$$

The control unit is initialised in the starting state s_0 and the read-write head is on the starting square. The transition function tells the machine what to do next given the current state and the alphabet in the current square. For example, if the control unit is in state s and the current square contains the symbol σ, then the value of $\delta(s, \sigma)$ tells the machine three things:

1. the new state for the control unit (if this is a halting state then the computation ends);

2. the symbol to write in the current square;

3. whether to move the read-write head to the left or right by one square.

The *computation* of a DTM on input $x \in (\Sigma \backslash \{*\})^*$ is simply the result of applying the transition function repeatedly starting with x written in the first $|x|$ tape squares. If the machine never enters a halting state then the computation does not finish, otherwise the computation ends when a halting state is reached. We say that a DTM computes a function $f : (\Sigma \backslash \{*\})^* \rightarrow (\Sigma \backslash \{*\})^*$ if the machine halts on every input $x \in (\Sigma \backslash \{*\})^*$, and the output in each case is $f(x)$.

2.2.2 Common Complexity Classes

With DTM, we are able to answer two fundamental problems in complexity theory:

- Is a problem Π intrinsically easy or hard to solve?
- Given two problems Π_1 and Π_2, which is easier to solve?

To address the first problem, we establish classes of problems such that for each class, the amount of *computational resources* required for solving problems in the class are comparable. Then, if we can show that the problem Π requires as much (and not less) resources as the problems in a certain complexity class, we can say that the problem Π is as hard as the other problems in that complexity class.

The second question can be addressed by showing that an algorithm for solving one of the problems can be used to solve the second problem, therefore, the first problem must

be 'at least as hard' as the second problem. This technique is known as *reduction* and it lies in the core of modern cryptography.

The most commonly analysed problems in complexity theory are decision problems – those problems that can be posed as yes-no questions. For example, the question 'is the natural number n prime' is a decision problem. As the set of prime numbers can be represented as a set, deciding whether a number is prime is equivalent to testing if a number is in the set of prime numbers. For that reason, we often refer to decision problems as *languages*. In terms of the DTM model of computation, we say a DTM M decides a language L if for all $x \in L$, $M(x)$ halts and accepts x; for all $x \notin L$, $M(x)$ halts and rejects x.

Before diving into different complexity classes, we introduce the big-O notation which will be useful later.

Definition 2.13 (Big-O Notation)**.** Let f and g be real-valued functions.Let both functions be defined on some unbounded subset of the positive real numbers, and $g(x)$ be strictly positive for all large enough values of x. One writes

$$f(x) = \mathcal{O}(g(x)) \text{ as } x \to \infty$$

if the absolute value of $f(x)$ is at most a positive constant multiple of $g(x)$ for all sufficiently large values of x. That is, $f(x) = \mathcal{O}(g(x))$ if there exists a positive real number M and a real number x_0 such that

$$|f(x)| \leq Mg(x) \text{ for all } x \geq x_0.$$

Similarly, we use big-Ω as a lower bound on the growth of functions and big-Θ as a tight bound on the growth of functions. We omit the formal definitions for the later notations.

Complexity Class **P**. To put it simply, the complexity class **P** are all the problems which are considered computationally feasible in the DTM model. As the name of the complexity class suggests, 'P' refers to the time complexity of the DTM. We can formally define the time complexity of a DTM M as follows.

Definition 2.14 (Time Complexity of a DTM)**.** Let M be a DTM which halts on every input $x \in (\Sigma \setminus \{*\})^*$. Let $t_M(x)$ be the number of steps M takes to compute on x. Then the time complexity $T_M : \mathbb{N} \to \mathbb{N}$ of M is defined by:

$$T_M(n) = \max \{t \mid \exists x \in (\Sigma \setminus \{*\})^n : t_M(x) = t\}.$$

The class of polynomial time decidable languages **P** can then be defined as:

Definition 2.15 (Complexity Class **P**)**.**

$$\mathbf{P} = \{L \subseteq (\Sigma \setminus \{*\})^* \mid \text{there is a DTM } M \text{ which decides } L \text{ and a polynomial } p(n) \text{ such that } T_M(n) \leq p(n) \text{ for all } n \geq 1\}.$$

The problems in complexity class **P** are, in some sense, *easy* to solve, and hence, unsuitable as building blocks of cryptosystems. Instead, cryptographers are interested in problems that are hard to solve on their own, but are easy if some information is given.

For example, such information could be a solution path of the computation, which can be efficiently verified by a DTM. By sharing the information between different parties, those parties can solve the problem efficiently whereas an eavesdropper cannot. The class of problems with this property is known as the complexity class **NP**.

Definition 2.16 (Complexity Class **NP**)**.** A language L is in **NP** if there exists a polynomial-time DTM, referred to as the verifier, that takes as input a string w and a certificate string c, and accepts w if $w \in L$ and rejects w if $w \notin L$.

It is worth to note that the **P** versus **NP** problem, that is, if **P** = **NP** is still unsolved. If **P** = **NP**, then all cryptosystems that rely on **NP** problems are broken.

Complexity Class **PP** Another important complexity class is the class **PP**. The key difference between **P** and **PP** is that languages in **PP** relies on a modified Turing machine, known as a probabilistic Turing machine.

A probabilistic Turing machine (PTM) is similar to a DTM (see Definition 2.12), except that there are two transition functions δ_1 and δ_2. At each step, the Turing machine randomly (50% chance to choose either transition function) applies either the transition function δ_1 or the transition function δ_2. This choice is made independent of all prior choices. Hence, the random choice at each step can be thought of as a coin flip. Due to the probabilistic nature of the transition functions, the Turing machine may accept a string on some occasions but reject on the others. To accommodate this, a language L is said to be recognized with error probability ϵ by a PTM M if:

1. $w \in L \Rightarrow \mathsf{Pr}[M \text{ accepts } w] \geq 1 - \epsilon$;
2. $w \notin L \Rightarrow \mathsf{Pr}[M \text{ rejects } w] \geq 1 - \epsilon$.

We can then define the complexity class **PP** [80] as follows.

Definition 2.17 (Complexity Class **PP**)**.** A language L is in **PP** if there exists a probabilistic Turing machine M, such that:

- M runs for polynomial time on all inputs;
- For all $x \in L$, M outputs 1 with probability strictly greater than $\frac{1}{2}$.
- For all $x \notin L$, M outputs 1 with probability less than or equal to $\frac{1}{2}$.

For a secure cryptosystem that is based on a decisional problem, we want to show that there is no probabilistic Turing machine (equivalently, *probabilistic polynomial-time (PPT) algorithm*) to decide the problem. Similarly, for a secure cryptosystem that is based on a computational problem, we want to show that there is no probabilistic Turning machine to solve the problem. All of the security notions in this thesis feature PPT adversaries.

Other Complexity Classes. There are many other complexity classes. For example, complexity classes **AC** [7], **ACC** [184] and **TC** [93] are complexity classes used in circuit complexity; complexity classes **AM** [11] and **IP** [125] which are used in interactive proof systems; complexity classes **L** and **NL** [73] which concern the space complexity; and

complexity classes **BQP** and **QMA** [192] which characterise hardness of problems on quantum computers.

2.2.3 Reductions

A reduction is an algorithm for transforming one problem into another problem. If the reduction itself is efficient, then it means that the second problem is at least as hard as the first problem.

Turing Reduction. Consider the following two computational problems:

- Problem A: given an integer n, find one of its prime factors.
- Problem B: given an integer n, find all of its prime factors.

If we want to show that problem B is at least as hard as problem A, we can used a reduction called *Turing reduction* [180] which uses an algorithm to solve problem B as an oracle to construct an algorithm for problem A. To demonstrate how it works, let M be a DTM for solving problem B, then we can construct another DTM for solving problem A as:

- Given an integer n, call M to generate the prime factors $p_1, \ldots, p_k$.
- Return p_1.

Turing reduction is by far the most commonly used reduction in security proofs of cryptosystems. To show that a cryptosystem is secure, we show that breaking the cryptosystem is at least as hard as breaking the underlying computational hardness assumptions. Reduction is used in Chapter 7 for security proofs.

Many-one Reduction. Many-one reduction [154] is another important reduction. In many-one reduction, if one wants to show that problem B is at least as hard as problem A, he has to construct an algorithm which transforms inputs of problem A so that an oracle call to an algorithm which solves problem B outputs solutions to problem A. Many-one reduction can be seen as a special case of Turing reduction where only one oracle call is allowed in the end.

2.3 Table of Notations

This section lists the notations used in this thesis.

Notation	**Description**
$\|X\|$	Absolute value if X is a number, number of elements in X if it is a set
$\|\|f\|\|$	Size of the support of function f where f is non-zero
$\leftarrow$	Assignment
$\xleftarrow{R}$	Uniform random sampling (from a set or a list)
$\{\}$	Set
$\{\{\}\}$	Multiset
$[\,]$	List
$[x_1, \ldots x_l] + [y_1, \ldots, y_m]$	List concatenation
$s_1 \| \| s_2$	String concatenation
$\texttt{negl}(\lambda)$	Negligible function
$\texttt{poly}(n)$	Polynomial function in n

Chapter 3

Background II: Cryptographic Foundation

This chapter serves as an introduction to cryptography with a focus on the primitives and cryptosystems used in structured encryption. For a more thorough treatment of the subject, [83, 164, 24] are good places to start.

Section 3.1 introduces computational ciphers and semantic security, and presents the formal security frameworks. Section 3.2 introduces pseudo-random generators and show how they can be used as encryption schemes. Section 3.3 studies another way of building an encryption scheme, namely block cipher, and shows how to encrypt messages of arbitrary lengths with it. Section 3.4 studies pseudo-random functions. Section 3.5 offers a brief introduction to the other primitives that are used by or related to structured encryption.

Contents

3.1 Encryption

Suppose Alice wants to deliver a message m to Bob without leaking the content to an eavesdropper. A simple way to do that is for Alice and Bob to agree on some secret way of transforming the message, for example, using a substitution table for the alphabets, so that Alice can apply the transformation to the message m to get a ciphertext c, and Bob can undo the transformation to recover the message. There is apparently a problem here. How can Alice and Bob agree on something before sending the message? This question is answered by key-exchange algorithms [62, 86], but for now, we assume that Alice and Bob have a way to synchronise the secret.

In modern cryptography, the transformation, known as the encryption algorithm, and the reverse transformation, known as the decryption algorithm, are publicly known, and the only secret is the *secret key* sk which is an input to both algorithms [1].

The same algorithm can be used for other purposes. For example, Alice can encrypt a message m and store it somewhere. Whenever she needs it later, she can recover the message m later with her secret key. This is exactly how one can store encrypted files on a public file storage system.

In this chapter, we show how to build a mathematically perfect solution to the encryption problem.

3.1.1 The Perfect Encryption Scheme

In this section, we introduce the perfect encryption scheme – one-time pad, and prove that it is *perfectly secure*. Before doing that, we give a formal definition of Shannon cipher [162].

Definition 3.1 (Shannon cipher)**.** A Shannon cipher is a pair $\mathcal{E} = (E, D)$ of functions. Let $\mathcal{K}$ be the set of all keys, $\mathcal{M}$ be the set of all messages, and $\mathcal{C}$ be the set of all ciphertexts.

- The function E is the encryption function which takes as input a key $\mathsf{sk} \in \mathcal{K}$ and a message $m \in \mathcal{M}$ (also known as a plaintext), and produces as output a ciphertext c. That is,

$$c = E(\mathsf{sk}, m).$$

- The function D is the decryption function which takes as input a key $\mathsf{sk} \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and produces a message $m \in \mathcal{M}$. That is,

$$m = D(\mathsf{sk}, c).$$

- We say that $\mathcal{E}$ is correct if

$$m = D(\mathsf{sk}, E(\mathsf{sk}, m)). \tag{3.1}$$

[1] We focus on the symmetric case here. It is possible to encrypt with a public key and decrypt with a secret key but that is not the focus of this thesis.

$\mathcal{E} = (E, D)$ can be anything: phrases of a language, integers, polynomials, matrices, or group elements. But in practice, for ease of storage and communication, the elements of the sets are often represented as sequences of bits. For simplicity of the mathematical treatment, we assume that $\mathcal{K}, \mathcal{M}$ and $\mathcal{C}$ are all finite in size.

ONE-TIME PAD. One-time pad is a Shannon cipher $\mathcal{E} = (E, D)$ where the keys, messages, and ciphertexts are bit strings of the same length, that is,

$$\mathcal{K} = \mathcal{M} = \mathcal{C} := \{0,1\}^L,$$

for some fixed parameter L. The encryption and decryption algorithms are defined as:

$$E(\mathsf{sk}, m) = \mathsf{sk} \oplus m,$$
$$D(\mathsf{sk}, c) = \mathsf{sk} \oplus c.$$

Here, "$\oplus$" is bit-wise exclusive-OR, or component-wise addition modulo 2. It is easy to verify that $\mathcal{E}$ is correct by checking Equation 3.1 holds:

$$\begin{aligned} D(\mathsf{sk}, E(\mathsf{sk}, m)) &= D(\mathsf{sk}, \mathsf{sk} \oplus m) \\ &= \mathsf{sk} \oplus (\mathsf{sk} \oplus m) \\ &= (\mathsf{sk} \oplus \mathsf{sk}) \oplus m \\ &= m. \end{aligned}$$

PERFECT SECURITY. With a cipher like one-time pad, the natural question to ask is: *how secure is it?* In this section, we answer that question by giving a formal definition of perfect security and prove that one-time pad satisfies the definition.

To begin with, for the definition of perfect security, we assume that the secret key is uniformly randomly distributed. We see later that the entropy of the key is the source of perfect security. In practice, this means the key needs to be carefully sampled to ensure uniformity, as otherwise perfect security will be violated. However, we assume that this is possible and we have access to independent and identically distributed (IID) fair coin flips.

Given the randomness of the key, we can look at the probability over the distribution of the key of a message m producing a ciphertext c, or $\mathsf{Pr}_{\mathsf{sk}}[E(\mathsf{sk}, m) = c]$. If this probability is the same for all messages, then all messages are equally likely given the ciphertext c. This is exactly we want in perfect security.

Definition 3.2 (Perfect Security)**.** Let $\mathcal{E} = (E, D)$ be a Shannon cipher. Let $\mathsf{sk} \in \mathcal{K}$ be uniformly distributed. If over the randomness of sk, for all $m_0, m_1 \in \mathcal{M}$, and all $c \in \mathcal{C}$,

$$\mathsf{Pr}[E(\mathsf{sk}, m_0) = c] = \mathsf{Pr}[E(\mathsf{sk}, m_1) = c],$$

we say that $\mathcal{E}$ is perfectly secure Shannon cipher [162].

As one can probably see, one-time pad is perfectly secure. This is because for ever pair of $m \in \mathcal{M}$ and $c \in \mathcal{C}$, there is a unique secret key sk such that $E(\mathsf{sk}, m) = c$, which means if the secret key sk is hidden, every message is equally likely. Below, we state the perfect security of one-time pad as a theorem and prove it.

Theorem 3.1 (One-time pad is perfectly secure)**.** Let $\mathcal{E}$ be a one-time pad cipher, then $\mathcal{E}$ is perfectly secure.

Proof. Let $m_0, m_1 \in \mathcal{M}$ be two messages and $c \in \mathcal{C}$ be a ciphertext. Then

$$\begin{aligned}\mathsf{Pr}[E(\mathsf{sk}, m_0) = c] =& \mathsf{Pr}[\mathsf{sk} \oplus m_0 = c] \\ =& \mathsf{Pr}[\mathsf{sk} = c \oplus m_0].\end{aligned}$$

Similarly, we get

$$\mathsf{Pr}[E(\mathsf{sk}, m_1) = c] = \mathsf{Pr}[\mathsf{sk} = c \oplus m_1].$$

Since $\mathsf{sk} \in \mathcal{K}$ is uniformly randomly distributed,

$$\begin{aligned}&\mathsf{Pr}[\mathsf{sk} = c \oplus m_0] = \mathsf{Pr}[\mathsf{sk} = c \oplus m_1] \\ \Rightarrow &\mathsf{Pr}[E(\mathsf{sk}, m_0) = c] = \mathsf{Pr}[E(\mathsf{sk}, m_1) = c].\end{aligned}$$

□

THE BAD NEWS. Careful readers may have already realised that one-time pad needs a key as long as the message itself. This is problematic as exchanging keys is expensive – whether Alice and Bob decide to meet physically and exchanging tens of hard disk drives, or using a modern key exchange protocol. So it is natural to ask if it is possible to achieve perfect security with a shorter key. Unfortunately, the answer is no. Intuitively, this is because the entropy of the ciphertext comes from nothing but the secret key, and if the key is shorter than the message, then there is not enough entropy to make every ciphertext equally likely. Formally, this is known as Shannon's theorem [162] which is shown below.

Theorem 3.2 (Shannon's theorem)**.** Let $\mathcal{E} = (E, D)$ be a Shannon cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. If $\mathcal{E}$ is perfectly secure, then $|\mathcal{K}| \geq |\mathcal{M}|$.

3.1.2 Computational Ciphers and Semantic Security

As we have seen in Theorem 3.2, it is impractical to use a perfectly secure cipher for real-world applications. As a way around, we relax the security requirements. Instead of considering all possible adversaries, we only consider *computationally feasible* adversaries who must perform their computations in a reasonable amount of time and use a reasonable amount of memory. In terms of the probabilities, we no longer require something as strong as $\mathsf{Pr}_{\mathsf{sk}}[E(\mathsf{sk}, m_0) = c] = \mathsf{Pr}_{\mathsf{sk}}[E(\mathsf{sk}, m_1) = c]$. We allow for differences that are reasonably small, such that with reasonable amount of sampling, the distributions cannot be distinguished.

In the same vein, we are interested in ciphers that are practical. That is, the encryption and decryption algorithms are efficient (polynomial time). These ciphers are called *computational ciphers*. Formally, we can define them as follows.

Definition 3.3 (Computational cipher)**.** A computational cipher is a pair $\mathcal{E} = (E, D)$ of functions. Let $\mathcal{K}$ be the set of all keys, $\mathcal{M}$ be the set of all messages, and $\mathcal{C}$ be the set of all ciphertexts, such that $\mathcal{K}, \mathcal{M}, \mathcal{C}$ are all finite.

- The function E is the encryption function which takes as input a key $\mathsf{sk} \in \mathcal{K}$ and a message $m \in \mathcal{M}$ (also known as a plaintext), and produces as output a ciphertext c. The function is allowed to be probabilistic, i.e.

$$c \xleftarrow{R} E(\mathsf{sk}, m).$$

- The function D is the decryption function which takes as input a key $\mathsf{sk} \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and produces a message $m \in \mathcal{M}$. That is,

$$m = D(\mathsf{sk}, c).$$

- We say that $\mathcal{E}$ is correct if

$$m = D(\mathsf{sk}, E(\mathsf{sk}, m))$$

  with probability 1.

In a more general setting, a computational cipher is parametrised by a security parameter 1^λ which is usually a positive integer, and a system parameter Λ which is usually a bit string.

We are now ready to define semantic security. As opposed to the straightforward probabilistic definition of perfect security, semantic security is defined via a *security game*.

Definition 3.4 (Semantic Security Game)**.** Let $\mathcal{E} = (E, D)$ be a computational ciphertext defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Let $\mathcal{A}$ be an adversary. We define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we have

Experiment b:

- The adversary computes $m_0, m_1 \in \mathcal{M}$ where $|m_0| = |m_1|$ and sends them to the challenger.
- The challenger computes $\mathsf{sk} \xleftarrow{R} \mathcal{K}, c \xleftarrow{R} E(\mathsf{sk}, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

With the two experiments, we can then define $\mathcal{A}$'s semantic security advantage.

Definition 3.5 (Semantic Security Advantage)**.** For the semantic security game above, for $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s semantic security advantage with respect to $\mathcal{E}$ as:

$$\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}} := \left|\mathsf{Pr}\left[W_0\right] - \mathsf{Pr}\left[W_1\right]\right|.$$

We note that the events W_0 and W_1 include the randomness from the choice of k, the choices (if any) made by the encryption algorithm, and the random choices made by the adversary. The value of $\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}}$ measures the distinguishing power of the adversary $\mathcal{A}$ at telling the two experiments apart.

Finally, we are ready to define semantic security.

Definition 3.6 (Negligible Function)**.** *Let $f : \mathbb{N} \to \mathbb{R}$ be a function. We call f a negligible function if for every positive integer c there exists an integer N_c such that for all $x > N_c$,*

$$|f(x)| < \frac{1}{x^c}.$$

Definition 3.7 (Semantic security)**.** A computational cipher $\mathcal{E}$ is semantically secure if for all efficient (PPT) adversaries $\mathcal{A}$, the value $\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}}$ is negligible in the security parameter λ.

3.2 Pseudo-random Generators (PRGs)

In the last section, we have defined computational ciphers and semantic security. In this section, we show how to construct a computational cipher which uses a short key.

Recall that for a one-time pad cipher, the keys, messages and ciphertexts are all L-bit strings for some constant L. The idea is to come up with an algorithm G that takes a much shorter input, say an l-bit string, and "stretch" it to an L-bit string to make it look like a key in a one-time pad cipher. If the output of G is *close enough* to a uniformly randomly sampled key in a one-time pad cipher, then we can hope that the resultant scheme is semantically secure. The class of algorithms that achieves this is called pseudo-random generators.

3.2.1 Formal Definition

A pseudo-random generator [132] is an efficient and deterministic algorithm G that takes as input a seed $s \in \mathcal{S}$ and outputs $r \in \mathcal{R}$. Typically, $\mathcal{S} = \{0,1\}^l$ and $\mathcal{R} = \{0,1\}^L$ for some $l < L$. We say G is a secure PRG if $G(s)$ for a randomly picked $s \in \mathcal{S}$ and a truly random string $r \xleftarrow{R} \mathcal{R}$ are computationally indistinguishable. We give the formal definition below.

Definition 3.8 (PRG Advantage)**.** For a given PRG G defined over $(\mathcal{S}, \mathcal{R})$, and for an adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define: **Experiment b:**

- If $b = 0$, the challenger samples $s \xleftarrow{R} \mathcal{S}$, then computes $r \leftarrow G(S)$.
- If $b = 1$, the challenger samples $r \xleftarrow{R} \mathcal{R}$.
- The challenger sends r to the adversary.
- Given r, the adversary computes and outputs a bit $\hat{b} \in \{0,1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s advantage with respect to G as

$$\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},G} := \left|\mathsf{Pr}\left[W_0\right] - \mathsf{Pr}\left[W_1\right]\right|.$$

Definition 3.9 (Secure PRG)**.** A PRG G is secure if the value $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{A},G}$ is negligible for all efficient adversaries $\mathcal{A}$.

3.2.2 Encryption with PRG

In this section, we show how to use a PRG as a primitive for semantically secure encryption. We introduce reduction as a core technique in the proof of semantic security. The same technique will be used over and over again in the later chapters.

As we have mentioned in the beginning, we can construct an encryption scheme by padding the plaintext with the output of a PRG $G(s)$ for which the seed s is randomly

sampled and kept secret to the user. More formally, let G be a PRF, we define a stream cipher $\mathcal{E} = (E, D)$ over $(\{0,1\}^l, \{0,1\}^L, \{0,1\}^L)$ for some $l \leq L$ be:

$$E(s, m) := G(s) \oplus m,$$
$$D(s, c) := G(s) \oplus c.$$

GAME HOPPING. To prove semantic security of $\mathcal{E}$, we use a technique called game hopping [59]. This technique will be used over and over again in the later chapters. The idea is to start with the original semantic security game for $\mathcal{E}$, change some parts of the game, and express the new advantage with the original semantic security advantage and other terms. This process is repeated several times until the original semantic security $\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}}$ can be bounded by some constants and advantages of other security games. That means if we assume that the constants and advantages from the other security games are negligible, we can show that $\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}}$ is negligible too. Below, we show how to prove the security of the stream cipher $\mathcal{E}$ defined above.

Theorem 3.3 (Semantic Security of $\mathcal{E}$)**.** *If G is a secure PRG, then $\mathcal{E}$ is semantically secure.*

Proof. Assume that G is a secure PRF. Let experiments Experiment 0 and Experiment 1 be the semantic security games for cipher $\mathcal{E}$ with adversary $\mathcal{A}$, i.e. for $b = 0, 1$, we get **Experiment** b:

- The adversary computes $m_0, m_1 \in \mathcal{M}$ where $|m_0| = |m_1|$ and sends them to the challenger.
- The challenger computes $s \xleftarrow{R} \mathcal{S}, c \xleftarrow{R} E(s, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

We modify Experiment 1 to be a one-time padding scheme as follows. Experiment 2:

- The adversary computes $m_0, m_1 \in \mathcal{M}$ where $|m_0| = |m_1|$ and sends them to the challenger.
- The challenger computes $r \xleftarrow{R} \mathcal{R}, c \xleftarrow{R} r \oplus m_1$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let $\mathcal{B}$ be an adversary against PRG security of G. Let W_b be the event that Experiment b outputs 1, for $b = 0, 1, 2$. Then we have:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}} &= |\mathsf{Pr}[W_0] - \mathsf{Pr}[W_1]| \\
&= |\mathsf{Pr}[W_0] - \mathsf{Pr}[W_2] + \mathsf{Pr}[W_2] - \mathsf{Pr}[W_1]| \\
&\leq |\mathsf{Pr}[W_0] - \mathsf{Pr}[W_2]| + |\mathsf{Pr}[W_2] - \mathsf{Pr}[W_1]| \\
&= \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G} + \mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G} \\
&= 2\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}.
\end{aligned}$$

In the first two lines of the equations, we have added and subtracted Experiments 2 from the advantage. The next line uses triangle inequality to split the probability into two halves. Following that, we note that the PRG advantage between Experiment 0 and Experiment 2 is $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}$, as the adversary can simply use $c - m_0$ with $\mathcal{B}$ to test if the output of the challenger comes from a PRG or is purely random. Similarly, we argue that the PRG advantage between Experiment 1 and Experiment 2 is $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}$. This tells us that the semantic security of $\mathcal{E}$ is upper-bounded by $2\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}$. As we know that G is a secure PRG, $\mathsf{Adv}^{\mathsf{PRG}}_{\mathcal{B},G}$ is negligible, so $\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{A},\mathcal{E}}$ must be negligible. That is, $\mathcal{E}$ is semantically secure. □

3.2.3 Drawbacks of Encryption with Stream Cipher

In this section, we discuss the drawbacks of encryption with stream cipher and motivate the need of better encryption schemes. Recall from above that a stream cipher is defined with a single seed s, which means if the same cipher is used to encrypt the same message twice, the ciphers will be the same. In other words, stream cipher is vulnerable to frequency analysis. If a stream cipher is used to encrypt two messages m_0 and m_1, although an adversary is not able to tell which of the ciphertexts comes from m_0 or m_1, the ciphertexts still can leak important information about the plaintexts. For example, one can XOR the ciphertexts to obtain the bit-wise XOR of the plaintexts:

$$E(s, m_0) \oplus E(s, m_1) = G(s) \oplus m_0 \oplus G(s) \oplus m_1 = m_0 \oplus m_1.$$

If one of the plaintexts, say m_0 is known to the adversary, he can freely change the ciphertext into any message m_1 he wants by XOR-ing $m_0 \oplus m_1$ with the ciphertext:

$$E(s, m_0) \oplus (m_0 \oplus m_1) = G(s) \oplus m_0 \oplus m_0 \oplus m_1 = G(s) \oplus m_1 = E(s, m_1).$$

To defend against these attacks while maintaining a short key length, block cipher with modes of operation was proposed [137]. The idea is to develop an encryption scheme for a fixed length *message block*, and build an encryption scheme for messages of arbitrary lengths by using the block cipher iteratively in a secure way.

3.3 Block Ciphers and Modes of Operation

Block cipher is a primitive which encrypts messages of a fixed length. The security requirement for a block cipher is much stronger than that of a stream cipher. Specifically, we require the block cipher to look like a random permutation. In this section, we give a definition of block cipher, show its formal security requirements, and demonstrate how to use a block cipher to encrypt messages of arbitrary lengths securely. We also give the security definition for the case where the same key is used to encrypt multiple messages.

We use block ciphers for the construction of SWiSSSE in Chapter 7.

3.3.1 Block Ciphers

We define a block cipher as a deterministic cipher $\mathcal{E} = (E, D)$ where the key space is $\mathcal{K}$ and the message and ciphertext space are $\mathcal{X}$. The correctness of a block cipher requires that for all $\mathsf{sk} \in \mathcal{K}$ and $m \in \mathcal{X}$, $D(\mathsf{sk}, E(\mathsf{sk}, m)) = m$. As the message and

ciphertext space have the same size, this necessarily means that $E(\mathsf{sk}, \cdot)$ and $D(\mathsf{sk}, \cdot)$ are permutations and they are inverse of each other.

The security of a block cipher is modelled by the following pseudo-random permutation (PRP) experiments.

Definition 3.10 (PRP Advantage)**.** Let f be a keyed permutation $\mathcal{K} \times \mathcal{X} \to \mathcal{X}$. Define Experiment b for $b = 0, 1$ for an adversary $\mathcal{A}$ as follows:

- The adversary can query the challenger adaptively multiple times.
- Suppose m is the message the adversary challenged. If $b = 0$, the challenger samples $\mathsf{sk} \xleftarrow{R} \mathcal{K}$, computes $c \leftarrow E(\mathsf{sk}, m)$, and send c to the adversary.
- Suppose m is the message the adversary challenged. If $b = 1$, the challenger picks a truly random permutation over $\mathcal{X}$ (and fixes it thereafter), say P, and send $P(m)$ to the adversary.
- At a certain point, the adversary outputs a bit b and the experiments terminates.

For $b = 0, 1$, let W_b be the event that the adversary outputs 1 in Experiment b. We define $\mathcal{A}$'s advantage as:

$$\mathsf{Adv}^{\mathsf{PRP}}_{\mathcal{A},f} = \left|\mathsf{Pr}\left[W_0\right] - \mathsf{Pr}\left[W_1\right]\right|.$$

Finally, we say that $\mathcal{A}$ is a Q-query PRP adversary if $\mathcal{A}$ issues at most Q queries.

Definition 3.11 (Secure PRP)**.** We say $f : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ is a secure PRP if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{PRP}}_{\mathcal{A},f}$ is negligible.

We note that it is very hard for a block cipher to meet this security. Let $P(\mathcal{X})$ be all permutations on $\mathcal{X}$, then the number of permutations is

$$|\mathsf{Perm}(\mathcal{X})| = |\mathcal{X}|!,$$

where $\mathsf{Perm}(\cdot)$ represents the set of all possible permutations. On the other hand, the number of permutations a block cipher $\mathcal{E}$ can generate is limited by its key space, or $|\mathcal{K}|$ possible permutations in total. To illustrate the difference with some numbers, consider $|\mathcal{K}| = |\mathcal{X}| = 2^{128}$, we have $|\mathsf{Perm}(\mathcal{X})| \approx 2^{2^{135}}$, which is a lot larger than the key space!

3.3.2 Modes of Operation

Just like stream ciphers, block ciphers can be used to encrypt messages directly. However, it suffers from the same problems as stream ciphers. That is, repetition of message is leaked and the messages can be changed by tweaking the ciphertexts. To address the problems above, we need to make the blocks of ciphertexts depend on more than one plaintext block. At the same time, we need to have integrity check build into the encryption.

One of the first *mode of operation* [137] that achieves this is called cipher block chaining (CBC) [64]. Let $m_0, \ldots, m_l$ be the message blocks, the idea is to encrypt the i-th message block with the $i-1$-th ciphertext block. That way, even if two message blocks

are the same, since they are XOR-ed with seemingly random ciphertext blocks, there is an overwhelming chance that their ciphertexts will be different. As a slight complication, the message block m_0 does not have a ciphertext block to be XOR-ed with. To fix the problem, we randomly generate an initial value (IV) block, and it is send to the receiver together with the ciphertext blocks. Formally, we can write the encryption algorithm as:

1. $IV \xleftarrow{R} \mathcal{M}$
2. $c_0 \leftarrow E(\mathsf{sk}, m_0 \oplus IV)$
3. For $i \leftarrow 1, \ldots, l$: $c_j \leftarrow E(\mathsf{sk}, m_i \oplus c_{i-1})$
4. Output (IV, c)

We note that it is not enough to just have CBC to defend against attacks. For example, an attacker can append more ciphertext blocks to the ciphertext to potentially generate adversarial plaintexts. To solve this problem, the length of the plaintext is padded to the plaintext before encryption. Unfortunately, CBC mode is known to be vulnerable to padding oracle attacks.

Counter mode is another mode of operation [64]. On its own, it achieves a weaker security guarantee than CBC mode, but it can be made secure against all of the attacks discussed above. In counter mode, the user randomly picks an initial value (IV) as a counter, and for each message block, he increments the counter, encrypts it, and XOR it with the message block. In some sense, it is very similar to a stream cipher, and as expected, it is vulnerable to all attacks against a stream cipher. To fix the security vulnerabilities, one can use a message authentication algorithm to authenticate the ciphertexts. The most well-known mode that does this is called Galois/Counter mode (GCM). For the purpose of encrypting document identifiers and documents in a database, GCM suffices for the security needs.

3.3.3 Chosen-Plaintext Attack Security

We now know that we can use a secure block cipher to encrypt messages of arbitrary lengths. But this does not automatically imply security when the block cipher is used to encrypt multiple messages. Of course, one can generate a key each time before encryption, but this step is time consuming in practice and we want to avoid it as much as possible. This motivates the security notion known as semantically secure against chosen-plaintext attack where the security notion is almost identical to semantic security introduced in Section 3.1.2 except that the attacker can choose more than one pair of plaintexts *adaptively*:

Definition 3.12 (Chosen-plaintext Attack (CPA) Game)**.** Let $\mathcal{E} = (E, D)$ be a computational ciphertext defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Let $\mathcal{A}$ be an adversary. We define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we have

Experiment b:

- For $i = 1, 2, \ldots$, the adversary computes $(m_{i,0}, m_{i,1}) \in \mathcal{M}$ where $|m_0| = |m_1|$ and sends them to the challenger.
  The challenger computes $\mathsf{sk} \xleftarrow{R} \mathcal{K}, c \xleftarrow{R} E(\mathsf{sk}, m_{i,b})$, and sends c to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0,1\}$.

Definition 3.13 (CPA Advantage)**.** For the semantic security game above, for $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s semantic security advantage with respect to $\mathcal{E}$ as:

$$\mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{A},\mathcal{E}} := \left|\mathsf{Pr}\left[W_0\right] - \mathsf{Pr}\left[W_1\right]\right|.$$

3.4 Pseudo-random Functions (PRFs)

Another primitive which we will use later (in Chapter 7) is pseudo-random function. A PRF F is a deterministic algorithm that takes as input a key $\mathsf{sk} \in \mathcal{K}$, a message $x \in \mathcal{X}$, and outputs a sequence of bits $y \in \mathcal{Y}$. For a secure PRF, the outputs should look like outputs from a truly random function. Such a function F allows one to completely mask a message (provided that it is short enough as an input to the PRF) from an eavesdropper, yet for someone else with the key, he can recompute the output of the F to verify if the message has been tampered.

For structured encryption, PRFs can be used to generate search keys for the queries, so as to hide the identity of the queries themselves. We note that encryption algorithms can do the same job, but practical implementations of PRFs are much faster than encryption algorithms so the earlier is preferred for the application. In this section, we give a formal security definition of PRF.

Definition 3.14 (PRF Advantage)**.** For a given PRF F defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, and for an adversary $\mathcal{A}$, we define two experiments, Experiment b for $b = 0, 1$:

- If $b = 0$, the challenger picks: $\mathsf{sk} \xleftarrow{R} \mathcal{K}$, $f \leftarrow F(k, \cdot)$.
- If $b = 1$, the challenger picks: $f \xleftarrow{R} \mathsf{Funs}(\mathcal{X}, \mathcal{Y})$.
- The adversary repeatedly query the challenger with $x_i \in \mathcal{X}$; the challenger computes $y_i \leftarrow f(x_i)$ and return y_i to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0,1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s advantage with respect to G as

$$\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},F} := \left|\mathsf{Pr}\left[W_0\right] - \mathsf{Pr}\left[W_1\right]\right|.$$

Definition 3.15 (Secure PRF)**.** A PRF F is secure if for all PPT adversaries $\mathcal{A}$, the value $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},F}$ is negligible.

We note that in this definition, the adversary is only allowed to query the challenger once. There are other variants of the definition, including non-adaptively security where the attacker generates multiple queries at once; weak security where at every invocation the challenger returns the pair $(x, F(x))$ for a random $x \in \mathcal{X}$; and sequential security where the i-th call to the challenger is answered by the value $F(i)$. In the later chapters, we use $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{A},F}$ to mean non-adaptive PRF security, and the key is omitted from the PRF for readability.

3.5 Other Primitives

In this section, we introduce other primitives that appear in or are related to structured encryption.

Public-key Cryptography. The primitives we have listed above are all symmetric primitives. In other words, the encryption key and the decryption key are identical. However, there are scenarios for which this is not sufficient. For example, if someone wants to build a database which everyone can encrypt and upload documents to it but only certain people can decrypt, he needs a scheme that is built from a public-key primitive [65, 130, 110]. Public-key cryptography is also very helpful when one wants fine-grained access control on the database. For example, the data owner can issue search keys to the users such that each key can only be used for a subset of the search queries.

Oblivious Random-access Memory. Oblivious random-access memory (ORAM) [84] is a primitive which transforms a program into one such that the memory access pattern is independent of the original program. This is motivated by the fact that an adversary can obtain non-trivial information about the execution of a program and the underlying data just by observing the access pattern during the execution of the program even if the contents of the memory are encrypted. More recently, there are works to generalise ORAM to other data structures [190, 67]. These ideas can be applied to structured encryption as access-pattern leakage is known to be problematic (see Section 4.5 and Chapter 6). On the other hand, ORAM is known to be relatively inefficient. Larsen and Nielsen [118] have shown that for any online ORAM, the lower bound for the bandwidth overhead is $\Omega(\log(n))$, where n is the size of the memory.

Private Information Retrieval. A private information retrieval (PIR) protocol [49] is a protocol which allows a user to retrieve an item from a server without revealing which item is retrieved. This can be trivially achieved by retrieving the entire database. However, this is not an efficient solution. In practice, we are interested in computational PIRs where the server is computationally bounded, and multi-server PIRs where there are more than one server and they are assumed to be non-cooperating. There is a long line of work [9, 77, 179, 32] which focused on improving the communication efficiency of a PIR protocol. Finally, in 2015, Kiayias et al. [105] have shown a PIR protocol with optimal communication rate.

Fully-homomorphic Encryption. Fully-homomorphic Encryption (FHE) [76] is a primitive which allows a user to perform arbitrary computations on encrypted data. This leads to many works [124, 42, 3, 176, 95] on cloud computation. These techniques can be applied directly to an structured encryption scheme to allow for secure statistical queries. The major drawback of FHE is its ciphertext expansion. TFHE [48] for example, requires the ciphertext to be 10 thousand times longer than the plaintext for a reasonable level of security (100 bits).

Chapter 4

Background III: Structured Encryption

This chapter formally recalls the core ideas around *structured encryption* (STE). Section 4.1 provides the syntax of structured encryption. Section 4.2 uses a simple construction to demonstrate the core ideas behind structured encryption. Section 4.3 gives the standard security notion used in the literature. Section 4.4 offers an overview of the constructions in the literature. Section 4.5 highlights the typical information leakage associated to the use of structured encryption and lists the classes of known attacks on some of these leakages. Section 4.6 and 4.7 discuss the research challenges of structured encryption.

Contents

4.1 Background

In this section, we describe the basic objects used in structured encryption and the syntax of structured encryption.

4.1.1 Abstract Data Types

An abstract data type is a collection of data objects and a set of operations defined on those objects. For instance, set with an operation to initialise a set and the common set operations is an abstract data type. Operations on abstract data types can be broadly categorised into two groups: static operations which do not change the data objects; and dynamic operations which may change the data objects. We use **Query** to denote static operations and **Updt** to denote dynamic operations.

4.1.2 Syntax

Let $\mathcal{T}$ be an abstract data type supporting query operation **Query** and update operation **Updt**. Then, a private-key structured encryption scheme Σ for $\mathcal{T}$ is a tuple $\Sigma = (\mathbf{Setup}, \mathbf{Query}_e, \mathbf{Updt}_e)$ where:

- **Setup** is the setup algorithm which takes as input some data **D** of structure $\mathcal{T}$, and outputs a secret key sk and some encrypted data **ED**.
- $\mathbf{Query}_e$ is the query protocol between the client and server. The client takes as input a secret key sk and a query **q**, and the server takes as input some encrypted data **ED**; after the interaction between the client and server, the client obtains a response **rsp**.
- $\mathbf{Updt}_e$ is the update protocol between the client and server. The client takes as input a secret key sk and an update query **q**, and the server takes as input some encrypted data **ED**; after the interaction between the client and server, the client obtains a response **rsp**, and the server obtains updated encrypted data $\mathbf{ED}'$.

Here, we assume that the query protocol supports query types that do not change the underlying data, whereas the update protocol supports query types that do. If the scheme Σ does not support update queries, we say that scheme Σ is static, otherwise we say that scheme Σ is dynamic. For the purpose of this thesis, we assume that there is only one user and one server.

Scheme Σ is correct if for all data **D** and all sequences of polynomially many queries, an execution of scheme Σ on encrypted data $\mathbf{ED} \leftarrow \mathbf{Setup}(\mathsf{sk}, \mathbf{D})$ yields the same query responses as an execution of the same queries using query operation **Query** and update operation **Updt** on the plaintext data **D**.

4.1.3 Refinement for Specialised Schemes

This thesis considers two abstract data types, namely keyword-based databases and value-based databases. The corresponding structured encryptions for these abstract

data types are known as searchable encryption and encrypted range queries. In this section, we give a detailed description for each abstract data type.

Searchable Encryption. A keyword-based database $\mathbf{DB}$ consists of a set of documents doc_i, each associated to a set of keywords W_i, so $\mathbf{DB} = \{(doc_i, W_i)\}$. The database $\mathbf{DB}$ must support keyword search queries, where a keyword search query $\mathbf{q}$ on keyword kw returns the set of documents containing the keyword, i.e. $\{doc_i \mid kw \in W_i\}$. The database $\mathbf{DB}$ may support additional static query types such as boolean queries and keyword search queries with wildcards. If $\mathbf{DB}$ is dynamic, then it must support document insertions and deletions. A structured encryption scheme for keyword-based databases is known as searchable encryption. Here, we focus on the symmetric-key setting and searchable symmetric encryption is typically abbreviated as SSE.

Encrypted Range Queries. A value-based database $\mathbf{DB}$ consists of a set of documents doc_i, each associated to a value val_i, so $\mathbf{DB} = \{(doc_i, \mathsf{val}_i)\}$. The database $\mathbf{DB}$ must support range queries, where a range query $\mathbf{q}$ on range (a, b) returns all documents with associated values greater or equal to a and less than or equal to b, that is, $\{doc_i \mid a \leq \mathsf{val}_i \leq b\}$. The database $\mathbf{DB}$ may support additional static and dynamic queries. A structured encryption scheme for value based databases is known as encrypted range queries.

Notations. We use the following notations for the two types of structured encryption above. For a keyword-based database $\mathbf{DB} = \{(doc_i, W_i)\}$, we assume an implicit order and write $\mathbf{DB}[i]$ to mean the i-th document and associated keywords. We write $KW(doc)$ to mean the set of keywords associated to document doc. We write $KW(\mathbf{DB})$ to mean all of the keywords in the database. For a keyword search query $\mathbf{q}$, we write $KW(\mathbf{q})$ to mean the keyword associated to the query. We use $\mathbf{DB}(W)$ to mean the set of documents containing all of keywords in W.

For a value-based database $\mathbf{DB} = \{(doc_i, \mathsf{val}_i)\}$, we write $\mathsf{val}(doc)$ to mean the value associated to document doc. We write $\mathbf{DB}[a, b]$ to mean the set of documents with values between a and b.

4.2 A Simple Searchable Encryption Scheme

This section shows how to construct a simple keyword-based database which supports keyword search queries only. For simplicity, we use database to refer to keyword-based database in this section thereafter. We then modify the database with cryptographic primitives to achieve *some level* of privacy. This construction is then used to motivate the standard security notion for structured encryption and other structured encryption schemes in the literature.

Searching in An Unencrypted Database. Taking a step back, let us consider how searching can be done on an unencrypted database. For now, we ignore privacy concerns and focus on the efficiency of the search protocol. Suppose that the user wants to retrieve all documents containing keyword kw. There is not a lot he needs to do other than to send the keyword in plain to the server. A naive server (Algorithm 4.1) can then iteratively scan through the documents and grab the documents for which $kw \in kw(doc)$.

Algorithm 4.1 A naive server

input A keyword kw
output The set of documents $\{doc_i \mid (doc_i, W_i) \in \mathbf{DB}, kw \in W\}$ containing keyword kw

procedure SEARCH($kw, \mathbf{DB}$)
    rsp $\leftarrow \{\}$
    for $(doc, W) \in \mathbf{DB}$ **do**
        if $kw \in W$ **then**
            rsp $\leftarrow$ **rsp** $\cup\, doc$
    Return rsp

Careful readers should have already realised that the approach is not scalable: even if there is a single document containing keyword kw, the server still has to scan through the whole database to find the document in interest. As a solution to this problem, the server can create and maintain an *inverted index* (also known as a search index or lookup index) to make the search time linear in the response size.

An inverted index $\mathbf{I}$ is a map $KW(\mathbf{DB}) \to \mathcal{P}(\mathbf{DB})$, where $\mathcal{P}(\mathbf{DB})$ is the power set of database $\mathbf{DB}$, which is indexed by the keywords and $\mathbf{I}(kw)$ contains all the documents containing keyword kw. The server can build the inverted index when the user uploads the database to him, and for subsequent queries, he can simply use it to lookup for the query results. The pseudocode for the search algorithm on the server is shown in Algorithm 4.2. The pseudocode for building the inverted index is omitted as the server can simply invoke Algorithm 4.1 on all keywords.

Algorithm 4.2 An improved server with an inverted index

input A keyword kw
output The set of documents $\{doc_i \in \mathbf{DB} \mid kw \in kw(doc_i)\}$ containing kw

procedure SEARCH($kw, \mathbf{I}$)
    Return $\mathbf{I}(kw)$

In practice, the inverted index often operates on the document identifiers as opposed to the documents directly. This is because if the documents are used directly (not as pointers but as hard copies), the inverted index will be much larger than the original database as each document appears as many times as the number of keywords in it. The document identifiers on the other hand are short strings and duplication does not hurt storage a lot. To retrieve the actual documents, the client simply send the database to the server along with the inverted index; upon a search query, the server can look up for the document identifiers from the inverted index and send the documents corresponding to them back to the client. The pseudocode for the construction described above is shown in Algorithm 4.3.

A SIMPLE SEARCHABLE ENCRYPTION SCHEME. There are two major privacy concerns with the unencrypted database above. Firstly, the server can see all documents in clear. This is clearly a problem for sensitive documents such as medical or financial records. Secondly, the keywords of the queries are sent and processed in clear. This means that the server can learn the query pattern of the user, and potentially infer information about the documents even if they are encrypted.

Algorithm 4.3 A refined server with an inverted index on the document identifiers

```
1: procedure BUILD_INDEX(DB)
2:     I ← {}
3:     EDB ← {}
4:     for (doc_i, W_i) ∈ DB do
5:         for kw ∈ W_i do
6:             I(kw) → I(kw) ∪ i
7:     Return (I, DB)

8: procedure SEARCH(kw, I, DB)
9:     rsp ← {}
10:    I ← I(kw)
11:    for i ∈ I do
12:        rsp ← rsp ∪ DB[i]
13:    Return rsp
```

We make two modifications to the scheme above to address these security concerns. Firstly, the documents are encrypted with a semantically secure symmetric key encryption scheme before uploading to the server. Secondly, instead of letting the server generating the inverted index, the client generates it, with the keywords replaced with the outputs of a PRF, i.e. $F(kw)$ if kw was used originally. The key of the PRF is omitted from the notation for readability.

The document identifiers are also encrypted with a semantically secure symmetric key encryption scheme. The pseudocode of the modified scheme is shown in Algorithm 4.4 and 4.5, where **Enc**() and **Dec**() are the encryption and decryption algorithms of the semantically secure symmetric key encryption scheme the client uses. Just as before, the key used by the encryption scheme is omitted to not overload notation. The key management aspect of the protocol is not shown in the pseudocode. In practice, the client generates the key and keeps it.

Algorithm 4.4 Client of the Searchable Encryption Scheme

```
1: procedure ENC_DOCS(DB)
2:     EDB ← [ ]
3:     for doc_i ∈ DB do
4:         EDB → EDB + [Enc(doc_i)]
5:     Return EDB

6: procedure BUILD_INDEX(DB)
7:     I ← {}
8:     for doc_i ∈ DB do
9:         for kw ∈ kw(doc_i) do
10:            I(F(kw)) → I(F(kw)) ∪ Enc(i)
11:    Return I

12: procedure GEN_QUERY(kw)
13:    Return F(kw)
```

SECURITY OF THE SIMPLE SEARCHABLE ENCRYPTION SCHEME. Consider a passive

Algorithm 4.5 Server of the Searchable Encryption Scheme

```
1: procedure SEARCH(t, I, EDB)
2:     rsp ← {}
3:     for ei ∈ I(t) do
4:         Ask the client for i ← Dec(ei)
5:         rsp ← rsp ∪ EDB[i]
6:     Return rsp
```

attacker who can only observe the encrypted database and the transcripts of the queries. The natural question to ask is: *what can he learn about the database and the queries?* He can certainly learn something. For example, the documents are encrypted one by one, so by simply looking at the encrypted index, the attacker can recover the number of documents in the database. The attacker can also look at the number of documents returned and figure out the frequency of any queried keyword. We can carry on and list *everything* an attacker can learn, but that is not rigorous, as there is no way of arguing that everything an adversary can learn is enumerated. The approach is also not robust, in the sense that a construction depends on other primitives (e.g. PRFs) that are not perfectly secure, and it is important to establish a concrete cryptographic advantage to understand the limits of a construction (e.g. how many documents can be encrypted safely under the same key).

In the next section, we show how information leakage is formally captured in the literature via *leakage profile* and we prove the security of our simple construction as a demonstration.

4.3 Security of Structured Encryption

FORMAL DEFINITION. The standard security notion for structured encryption was first proposed by Curtmola et al. [53] as a security notion for searchable encryption. The notion is later generalised by Chase and Kamara [40] as a security notion for structured encryption. The idea of the security notion is to enumerate the information leakage of a scheme with respect to the underlying data just like what we have done in the last part of Section 4.2. However, instead of stopping there, we need to show that there is a *simulator* who has access to the leaked information only, can produce some encrypted data and transcripts of queries such that an attacker who generates the plaintext data and the queries, cannot distinguish it from a real execution of the scheme.

Definition 4.1 (CQA2-security)**.** Let $\Sigma = (\mathbf{Setup}, \mathbf{Query}_e, \mathbf{Updt}_e)$ be a private-key structured encryption scheme for abstract data type $\mathcal{T}$. Consider the following probabilistic experiments between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$:

- $\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda)$: the adversary $\mathcal{A}$ generates data $\mathbf{D}$ and sends it to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ runs the **Setup** algorithm to generate a secret key sk and some encrypted data **ED**. The encrypted data **ED** is sent to the adversary. After that, the adversary picks a polynomial number of queries adaptively and send them to the challenger. The challenger and adversary executes the $\mathbf{Query}_e$ protocol and $\mathbf{Updt}_e$ protocol on the queries where the challenger plays the client and the adversary plays the server. Finally, the adversary outputs a bit b that is output by the experiment.

- $\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda)$: the adversary $\mathcal{A}$ generates data $\mathbf{D}$ and $\mathcal{L}_{\mathbf{Setup}}(\mathbf{D})$ is sent to the simulator $\mathcal{S}$. The simulator $\mathcal{S}$ generates encrypted data $\mathbf{ED}$ using the leakage and sends it back to the adversary. The adversary picks a polynomial number of queries $\mathbf{q}_1, \ldots, \mathbf{q}_l$ adaptively, and for $\mathbf{q}_i$, the simulator is given either $\mathcal{L}_{\mathbf{Query}_e}(\mathbf{q}_i, \mathbf{D})$ or $\mathcal{L}_{\mathbf{Updt}_e}(\mathbf{q}_i, \mathbf{D})$ depending on the type of the query. The simulator and adversary executes the $\mathbf{Query}_e$ protocol or the $\mathbf{Updt}_e$ protocol on the queries where the simulator plays the client and the adversary plays the server. Finally, the adversary outputs a bit b that is output by the experiment.

We define the semantic advantage of adversary $\mathcal{A}$ against simulator $\mathcal{S}$ to be

$$\mathsf{Adv}^{\mathsf{CQA2}}_{\mathcal{C},\Sigma}(\lambda) = \left|\mathsf{Pr}\left[\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda) = 1\right] - \mathsf{Pr}\left[\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda) = 1\right]\right|.$$

We say that Σ is $(\mathcal{L}_{\mathbf{Setup}}, \mathcal{L}_{\mathbf{Query}_e}, \mathcal{L}_{\mathbf{Updt}_e})$-secure against adaptive chosen-query attacks if for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

$$\mathsf{Adv}^{\mathsf{CQA2}}_{\mathcal{C},\Sigma}(\lambda) \leq \mathtt{negl}(1^\lambda).$$

We say that Σ is non-adaptively secure if the adversary $\mathcal{A}$ chooses the queries before executing them. We say that Σ is adaptively secure if the adversary $\mathcal{A}$ can choose his queries based on the past transcripts.

Any scheme is secure if the leakage is *all* of the database and the queries – that is not the point of the security notion. The idea is to show the *minimal* amount of information (often called the *leakage profile*) required for a simulator to work, and if that piece of information is harmless, for example, number of documents in a keyword-based database, we can say that the scheme is *practically* secure. A caveat to this approach is that 'the minimal amount of information' is usually difficult to specify, and it is not trivial to argue about security with it. Therefore, cryptanalysis is a necessary step after proving the leakage profile of a scheme.

Leakage of the Simple Searchable Encryption Scheme. We present the leakage of the simple searchable encryption scheme. For simplicity, we assume that the documents have the same length. Let us call the simple searchable encryption scheme Σ. In terms of the setup stage, we claim that the adversary learns the number of documents in the database and the frequencies of the keywords (as a multiset) from the search index:

$$\mathcal{L}_{\mathbf{Setup}}(\mathbf{DB}) = (|\mathbf{DB}|, \{\{|\mathbf{DB}(kw)| \mid kw \in kw(\mathbf{DB})\}\}).$$

Whenever the user makes a query on keyword kw, we claim that the scheme leaks the document identifiers of the matching documents, known as the *access pattern*; and whether it is the same as one of the previous queries, known as the *search pattern*. Traditionally, the search pattern is represented as a q-by-q array, where q is the number of queries made so far, and the i, j-th entry of the array is 1 if the i-th query has the same keyword as the j-th query, and 0 otherwise. For readability, we will simply call it $\mathbf{SP}(kw)$ below.

$$\mathcal{L}_{\mathbf{Srch}}(kw) = (\{i \mid kw \in kw(doc_i)\}, \mathbf{SP}(kw)).$$

The leakage of a construction is often shown by presenting a black-box simulator (see Definition 4.1) which works for all adversaries. We demonstrate how it is done for the

simple construction. For the setup algorithm, the simulator is given

$$(|\mathbf{DB}|, \{\{|\mathbf{DB}(kw)| \mid kw \in kw(\mathbf{DB})\}\}).$$

From $|\mathbf{DB}|$, he can generate an encrypted database $\mathbf{EDB}$ by encrypting $|\mathbf{DB}|$ documents filled with strings of zeros (the length of the documents are fixed and publicly known) using a randomly generated key. To generate the inverted index, the simulator picks random non-repeating strings as the keywords, and applies the F used by the client to the strings to generate the keys in the inverted index. He fills the inverted index with encryptions of strings of zeros according to $\{\{|\mathbf{DB}(kw)| \mid kw \in kw(\mathbf{DB})\}\}$: an instance of k in the multiset indicates that one of the keywords has k matching documents, so k encryptions of strings of zeros is added to one of the empty keys of the inverted index.

To simulate a query on keyword kw, the simulator checks if the keyword has appeared in an earlier query before using the search pattern leakage $\mathbf{SP}(kw)$. If so, he replays his execution of the previous query. Otherwise, he picks a key in the inverted index such that the number of encrypted documents returned matches $|\{i \mid kw \in kw(doc_i)\}|$, and returns $\{i \mid kw \in kw(doc_i)\}$.

Security requires us to show that real and ideal games are indistinguishable. This is done using standard game-hopping technique. Let game G_0 be the real execution of the scheme, and game G_1 be the same as G_0 except that the encryptions of the documents are replaced with encryptions of zeros (of appropriate lengths). The advantage of an adversary $\mathcal{A}$ distinguishing G_0 and G_1 cannot be higher than $\mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{A},\mathcal{E}}$ where $\mathcal{E}$ is the underlying cipher used to encrypt the documents. To transform game G_1 to game G_2, we replace the PRF outputs used for the keys of the search index with random strings of the same length. The advantage of an adversary $\mathcal{B}$ in distinguishing G_1 and G_2 is clearly $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},F}$, where F is the underlying PRF used. Finally, the transcripts of the queries are replaced with the random strings generated for the documents and keys. We call this game G_3. Game G_3 and G_2 are indistinguishable as there is no cryptographic operations involved and the leakage is consistent. In all, the advantage of an adversary who tries to distinguish G_0 and G_3 is bounded by the sum of $\mathsf{Adv}^{\mathsf{CPA}}_{\mathcal{A},\mathcal{E}}$ and $\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B},F}$. So if $\mathcal{E}$ is CPA secure and F is a secure PRF, the semantic security advantage of the scheme $\mathsf{Adv}^{\mathsf{SS}}_{\mathcal{C},\Sigma}(\lambda)$ for any adversary $\mathcal{C}$ must be negligible.

4.4 Structured Encryption in the Literature

This section gives an overview of the constructions of structured encryption in the literature, with a focus on searchable encryption.

4.4.1 Early Constructions

The first construction of searchable encryption dates back to 2000 in a paper by Song, Wagner and Perrig [166]. The construction uses something similar to a stream cipher to encrypt the database, and each ciphertext block can be decrypted by the server with the right keyword-derived token. The major drawback of this approach is that the search process has to scan through the entire database regardless of the actual frequency of the queried keyword. The authors have also proposed to use a search index to tackle the problem above. They noted that a search index may leak a lot of statistical information and it makes it difficult to craft dynamic schemes, which are exactly the two major research challenges that the community are facing now.

Goh [82] proposed a construction based on Bloom filter. A Bloom filter [18] is a data structure with two operations: insertion and membership testing. An insertion operation inserts an element into the bloom filter. A membership testing operation takes as input an element and outputs *True* if the element was inserted into the Bloom filter before (false positives are possible); it outputs *False* if the element was not inserted into the Bloom filter before (false negatives are impossible). By creating a Bloom filter for each document, and inserting keyword-derived tokens into the Bloom filters, the user can search over the database by sending the search token of the keyword he wants to search for to the server, and the server uses the Bloom filters to check if the documents contain the keyword and return all the documents that do. As before, the amount of work the server has to do is proportional to the number of documents.

Chang and Mitzenmacher [39] devised a construction based on an inverted index. Unlike the inverted index introduced in Section 4.2, the inverted index in [39] is a $|kw(\mathbf{DB})|$-by-$|\mathbf{DB}|$ bit matrix where every row is a bit string of length $|\mathbf{DB}|$ to indicate if each of the documents contains the keyword indexed by the row. The search time of the construction is linear in the number of documents due to this representation.

4.4.2 Modern Security Notions and Constructions

The modern notion. The modern security definition in the searchable encryption setting was introduced by Curtmola et al. in the seminal paper [53]. It was the first security definition based on leakage functions, and it captures the security of the database and the security of the search tokens at the same time. The authors have proposed the first searchable encryption scheme with search time proportional to query response size, using a search index based on linked list. This has inspired many works, including [100, 36, 97, 29, 116, 56]. The notion was later generalised to structured encryption by Chase and Kamara [40].

The Universal Compostability (UC) Model. Kurosawa and Ohtaki [113] considered the setting where the adversary can be malicious, and formulated UC-security for searchable encryption. They proposed a static searchable encryption scheme that is secure under their security definition, and extended it to dynamic databases [114].

Forward and Backward Security. The first efficient dynamic searchable encryption scheme was proposed by Kamara, Papamanthou and Roeder in 2012 [100]. However, it and other index-based schemes such as [35] are vulnerable to file-injection attacks [200]. This has inspired the notion *forward security*, where a scheme that is secure with respect to this notion should not leak any information about an updated keyword in *the update query itself*. Schemes which satisfies this notion were proposed in [169, 25].

A dynamic searchable encryption scheme may also expose unwanted information about deleted documents in *future queries*. To prevent such leakage, *backward privacy* was considered by Bost, Minaud and Ohrimenko [29], who also proposed a forward and backward private searchable encryption scheme.

4.4.3 Richer Queries

Boolean Queries. Many works tried to expand on the query types supported by a searchable encryption scheme. In particular, Cash et al. [36] and Pappas et al. [146] focused on support for boolean queries, where the user can issue queries with 'and, or, or not' between/in front of the keywords. For example, the user can retrieve all documents with keyword A *and* not keyword B with a single query. This was later improved by Kamara and Moataz in [97] in terms of search complexity and Lai et al. [116] in terms of leakage.

Other Text-based Queries. Instead of the boring exact keyword matches, Faber et al. [69] have shown how to search with wildcards, substrings and phrases. Boldyreva and Chenette [19] demonstrated how to perform fuzzy search (approximate string matching).

SQL-like Queries. It is important to note that all of the aforementioned constructions support limited query types and it is nowhere close to unencrypted databases used in practice. To bridge the gap, legacy-compatible constructions such as CryptDB [152] and Arx [150] are proposed. CryptDB is built from primitives such as homomorphic encryption, deterministic encryption and order-preserving encryption; whereas Arx is built with common techniques found in the encrypted database literature.

Multi-user Searchable Encryption. Another line of works [182, 191, 183] focused on searchable encryption in the multi-user setting where there is one centralised server but many users. This setting allows the users to have different privileges in terms of their permitted operations, which keywords they can search/update, and which subset of the database they can retrieve from. The biggest security threat, other than the ones applied to searchable encryption in general, is the possibility of colluding users. For that reason, works in this direction use a different security notion to traditional searchable encryption and the constructions are often built from different primitives.

Other Abstract Data Types. Besides searchable encryption, some works focused on searching on other abstract data types. Chase and Kamara [40] proposed structured encryption as a generalization of searchable encryption to any data structure. Meng et al. [128] proposed a scheme supporting approximate shortest distance queries on graph databases. There are also works studying k-nearest neighbour queries on graph databases [66, 47] and location queries on geographical databases [201]. Finally, there is a line of research which focuses on encrypted range queries [163, 31, 58, 57].

4.5 Leakage Cryptanalysis in the Literature

As explained above, the security of structured encryption is parametrized by a leakage function. The leakage function captures what an (honest-but-curious) adversary can learn. The problem with this approach is that unless the leakage is well understood to be harmless, for example, the number of documents in a database, there is a good chance that the leakage can be *abused* to recover some information the scheme intends to hide.

The process of studying security impact of leakage is known as leakage analysis in

the literature. This section serves as an overview of leakage analysis, with a focus on searchable encryption and encrypted range queries.

4.5.1 Characterisation of Leakage Cryptanalysis

The leakage cryptanalysis in the literature can be characterised by the following factors:

- **Attack Model**: The adversary may be *snapshot* (with access to only the encrypted data) or *persistent* (with access to both the encrypted data and the "history" of query operations). We note that an encrypted database may contain the history of query operations as part of its cache. In that case, a snapshot adversary is equivalent to a persistent one.
- **Attack Target**: The adversary may target either *data recovery* or *query recovery*. For data recovery, there exists weaker versions of it such as *approximate data recovery* and *distribution recovery*.
- **Attack Assumptions**: The adversary is assumed to have some *auxiliary data* to facilitate his attack. In a *known-data* attack, the adversary is assumed to know a subset of the original data. In an *inference* attack, the adversary is assumed to have a (potentially noisy) statistical representation of the original database. In a *known-query* attack, the adversary is assumed to know a subset of the queries.
- **Adversarial Power**: The adversary may either passively observe the leakage (referred to as *leakage-abuse attacks*) or actively create leakage by modifying the client's data (referred to as *injection attacks*).

4.5.2 Typical Leakage Functions

Works in leakage analysis typically focus on a single leakage function. The commonly studied leakage functions are as follows:

- **Search Pattern**: For a given query $\mathbf{q}$, the search pattern leakage reveals if the query is identical to any previous queries.
- **Access Pattern**: For a given search query $\mathbf{q}$, the access pattern leakage reveals the set of encrypted data elements retrieved by query $\mathbf{q}$.
- **Volume**: For a given search query $\mathbf{q}$, the volume leakage reveals the number of data elements returned.
- **Co-occurrence Pattern**: for given search queries $\mathbf{q}_i$ and $\mathbf{q}_j$, the co-occurrence count leakage reveals the number of encrypted data elements retrieved by queries $\mathbf{q}_i$ and $\mathbf{q}_j$ at the same time.

These leakage functions are allowed to be different from that of unencrypted data. For instance, a query $\mathbf{q}$ may retrieve n documents in a keyword-based database $\mathbf{DB}$, but retrieves $2n$ documents from an encrypted database $\mathbf{EDB}$ built for $\mathbf{DB}$ using a searchable encryption scheme Σ. The latter leakage is still called *volume leakage*.

We note that access pattern leakage can be used to construct volume and co-occurrence pattern leakage. As a result, attacks in the literature using the latter leakages are typically referred to as access pattern leakage attacks as well.

4.5.3 Leakage Cryptanalysis on Searchable Encryption

Search Pattern Leakage Attacks. Search pattern is the information whether two queries have the same underlying keyword. Many constructions [53, 100, 29] use fixed tokens to query the database, which means that search pattern is leaked. With some background information on how the queries are distributed, an adversary can use simple frequency matching to guess the underlying keywords of the queries with high accuracy [122, 126, 143].

Access-pattern Leakage Attacks. Access pattern is the information which set of document (or document identifiers) are accessed by a query. This piece of information is extremely useful as the access patterns of different queries overlaps only if the underlying keywords all appear in the same document. This allows an adversary to build a *co-occurrence* pattern from the queries. If he has some auxiliary information on the distribution, he can match the two and recover the underlying keywords of the queries, as demonstrated in [94, 102, 155, 17]. These attacks are devastating as a successful query recovery attack implies a data recovery attack (recovering the keywords in the documents).

File-injection Attacks. File-injection attacks are a class of attacks where the adversary can insert malicious documents with keywords of his choice into the database. By observing when those documents are retrieved by future queries (with unknown underlying keywords), the adversary can guess the underlying keywords of the queries easily [200, 17]. If the targeted scheme leaks access pattern, a successful file-injection attack also means that the adversary can recover all keywords in the other documents.

4.5.4 Leakage Cryptanalysis on Encrypted Range Queries

Access-pattern Leakage Attacks. Unlike access-pattern leakage attacks on searchable encryption, access-pattern leakage attacks on encrypted range queries do not require auxiliary information. This makes access-pattern leakage attacks on encrypted range queries much more devastating than those on searchable encryption.

For a dense database with N labels, Kellaris et al. [102] have proposed an attack with $\mathcal{O}(N^2 \log(N))$ query complexity assuming uniform queries. The result is later improved by Lacharité et al. [115] to $\mathcal{O}(N \log(N))$ for full database reconstruction and $\mathcal{O}(N)$ for approximate reconstruction. Grubbs et al. [88] extended the attacks with statistical learning theory and proved the lower bound on the query complexity of access-pattern leakage attacks.

Volume-leakage Attacks. The more surprising result is perhaps that the benign volume leakage is problematic for encrypted range queries. Volume leakage is defined to be the numbers of documents returned from the queries. From these volumes, Kellaris et al. [102] have shown that the distribution of the database (how many documents are associated to each label) can be recovered with $\mathcal{O}(N^4 \log(N))$ uniform queries. Grubbs et al. [87] improved the query complexity of the attack to $\mathcal{O}(N^2 \log(N))$ with the same assumptions.

4.6 The Index Retrieval Problem

A major caveat is that the schemes in the literature often use expensive techniques to achieve security. For instance, many searchable encryption schemes [53, 100, 36, 97, 29, 116, 56] use duplication to suppress leakage. The general idea of duplication is shown in Figure 4.1. Here, every *instance* of a keyword generates a key in the search index and each document generates k encrypted documents in the document array where k is the number of keywords in that document. This represents a huge storage overhead on the search index and the document array, especially when each document contains a large number of keywords or when the documents are large.

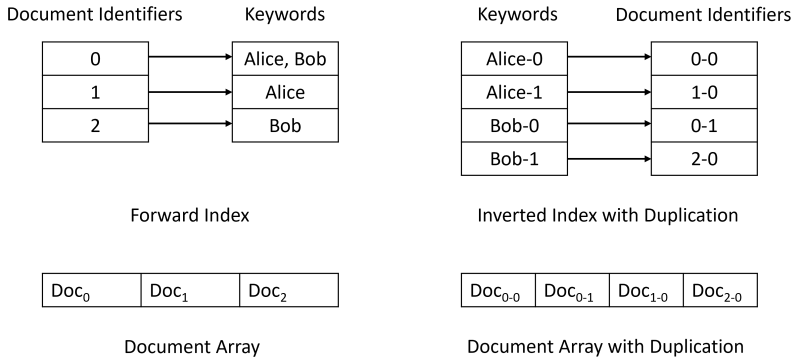


Figure 4.1: Demonstration of the duplication technique. The left side shows the documents and the keywords they contain. The right side shows the inverted index and document array with duplication. Note that the number of entries in the inverted index and document array is equal to the number of keyword-document pairs which can be extremely large for certain applications.

This has lead to three schools of thoughts on how those schemes should be used:

1. Security should never be compromised and the expensive techniques should be applied to the search index and document array at the same time.
2. Although large, the overhead induced by the schemes on the search index is still manageable as entries of the index are short (document identifiers). So one can use those schemes to retrieve document identifiers and use another space-efficient primitive (e.g. ORAM) to retrieve actual documents.
3. Efficiency is important and only encrypted indices should be built with expensive techniques. The documents should be left naively encrypted.

None of the proposed solutions is satisfactory. The first solution is, by definition, not efficient. The second solution suffers from time-efficiency problems as pointed out by Naveed [134] (see Section 6.1 for a detailed discussion). The last solution is insecure as pointed out by our results in Chapter 6.

4.7 A Conundrum

Given all of the attacks and constructions, a natural question to ask is: *is there a structured encryption scheme (for a particular abstract data type) that is secure against all of the known attacks and is practically efficient*? The answer to the question is a definite '*no*', even for simple types of structured encryption such as searchable encryption. While there are secure searchable encryption schemes proposed in the literature, they often focus on the *index* retrieval problem (a keyword search query returns the identifiers of the matching documents) and completely ignoring the *document* retrieval problem (a keyword search query returns the actual matching documents).

There is not a lot we can borrow from related primitives to make more efficient constructions too. For example, we know access-pattern leakage is problematic, but to hide it fully requires ORAM or something with a similar overhead [134, 27, 148], which is not going to be efficient for database applications in their current forms. We refer the readers to Section 6.1 for a detailed discussion.

In this thesis, we address the conundrum by creating a new paradigm: instead of the all-or-nothing approach, we try to construct schemes with *partial leakage* which allows for much better efficiency. In order to achieve the goal, we need to further our understandings on the leakages of the constructions (Chapter 5 and 6), design more efficient schemes (Chapter 7), and refine the security notion so that a security proof is meaningful with respect to leakage-abuse attacks (Chapter 8).

Chapter 5

Cryptanalysis I: Encrypted Range Queries

Range queries are one of the fundamental query types of a database system. In the context of structured encryption, the majority of the constructions consider one-dimensional range query only, that is, there is only one attribute that can be queried by the user. Despite the simplification, we are still very far from a secure and efficient construction.

In this chapter, we explore the different ways a structured encryption scheme that supports range queries can be broken, and learn which of the leakages to avoid. Section 5.1 provides an introduction to the techniques used in the previous constructions and the attacks on some of them. In Section 5.2, we demonstrate how some of the previous constructions which are secure against previous access-pattern leakage attacks can be broken. Section 5.3 improves on the previous volume leakage attacks. Unlike previous attacks in this direction, we are interested in imperfect volume leakages, meaning that not all volumes from the possible queries have been observed by the attacker as a result of a particular query distribution or active suppression of the leakage. Finally, we discuss in Section 5.4 what we have learnt from the attacks and highlight the leakages to avoid.

Contents

5.1 Introduction

In this section, we give a brief introduction to constructions of encrypted range queries and known attacks on them. We also discuss shortcomings of previous attacks and outline how we improve them. The syntax of encrypted range queries can be found in Section 4.1.

5.1.1 Previous Constructions

Previous constructions of structured encryption supporting range queries can be categorised into three groups.

The first group of constructions rely on property-preserving encryption schemes. This includes constructions based on order-preserving encryption [20, 21, 152, 177, 103] and order-revealing encryption [120, 46, 121]. Order-preserving encryption preserves the order of plaintext upon encryption and a search operation on the plaintext corresponds to a search operation on the ciphertext with the end-points of the query rescaled. Order-revealing encryption does not reveal the order of the ciphertexts until they are queried. However, with enough queries, it reveals as much information as an order-preserving encryption scheme would. Schemes in this group leak two pieces of information which an attacker can abuse, namely the search-pattern and access-pattern information. The search-pattern refers to whether two queries are the same, and the access-pattern refers to which query accesses which document.

The second group of constructions are tree-based constructions, including [186, 69, 58, 187, 203]. The idea of these constructions is to use different nodes of the tree to answer different queries, so as to avoid the access-pattern leakage.

The last group of constructions are custom constructions that do not fall into the other two categories. This includes [188, 163, 75].

5.1.2 Previous Attacks

For an attack on encrypted range queries, there are three main attack goals, namely query reconstruction, database reconstruction and distribution reconstruction. In a query reconstruction attack, the goal of the attacker is to recover the end-points of the range queries. In a database reconstruction attack, the goal of the attacker is to reconstruct the values of the encrypted documents. Finally, in a distribution reconstruction attack, the goal of the attacker is to reconstruct the shape of the database, i.e. how many documents there are for each value.

A scheme that is vulnerable to one of the attacks can leak sensitive information. Just as an example, a distribution reconstruction attack may look innocuous, but such an attack on the distribution of diseases[1] can lead to regional discrimination by the employers.

In terms of the attack techniques and the types of leakages used, there are three main categories of attacks, namely search-pattern leakage attacks, access-pattern leakage attacks and volume-leakage attacks.

[1] Diseases in medical databases are coded with International Classification of Diseases [193] which can be sorted and searched by range queries.

A scheme that leaks the search pattern is vulnerable to a search-pattern leakage attack. The OyaKer20 attack [142] is one of the attacks that uses search pattern to reconstruct the queries. All of the schemes described above are vulnerable to this attack.

If a scheme leaks the access pattern like all of the schemes in the first group do, then it is vulnerable to an access-pattern leakage attack. In such an attack, the adversary is able to reconstruct everything about the database – the queries, the database and the distribution – if enough queries have been made. The KKNO16 attack, LMP18 attack and GLMP19 attack [102, 115, 88] are some of the examples.

Finally, if a scheme leaks the query response sizes or *volumes* like all the schemes in the first group do, then the scheme is vulnerable to the volume-leakage attacks in the literature [102, 87, 111]. However, those attacks assume that *all* possible volumes are observed so they do not apply to some of the schemes in the second and third group.

It is worth to note that these attacks are closely related to traffic analysis [72, 70, 144, 133]. The key difference between the two is that for encrypted range queries, the server is assumed to be honest-but-curious, so the attack surface is larger.

5.1.3 Discussion

Before moving on, we briefly discuss three aspects of the above attacks. Although they expose a clear source of insecurity, most of the attacks rely on strong assumptions on the information the adversary learns. For instance, the KKNO16 attack and LMP18 attack [102, 115] assume a uniform query distribution and will only work if all queries have been observed, and the OyaKer20 attack [142] assumes that the attacker has some background information on the query distribution. Indeed, since at the moment we have little understanding of what is the query distribution in practical applications, it is difficult to confirm the assumptions required by the attacks. Is it unclear how reasonable it is to assume that the adversary gets to observe answers to all possible queries. The independence and uniformity assumptions on the query distribution are perhaps even harder to justify.

Furthermore, it seems easy to defend against these attacks on the schemes they are applicable to, since they do not seem robust under small departures from the assumptions on which they rely. Put differently, do the attacks fail if a few queries are missing? What if the adversary never gets to see the answer to $\mathbf{q}(1, N)$ or queries of the form $\mathbf{q}(i, i)$, both of which are crucial for some of the attacks to work? These are not merely rhetorical questions. One tempting proposition is to deploy protocols with a modicum of protection which invalidate the assumptions required by the attacks. For example, one could simply ensure that certain volumes are never returned [58], prevent queries for large ranges, or introduce noise in the database or communication [57]. Each of these countermeasures thwarts existing attacks, with no obvious way to bypass them.

Finally, the attacks only applies to a subset of the schemes in the literature, and a natural question to ask is if the other schemes are secure against leakage-abuse attacks.

5.1.4 Chapter Outline

This chapter is an attempt to address the questions above. There are two main sections.

In Section 5.2, we show how to abuse access-pattern leakage on schemes with countermeasures to achieve database reconstruction. Section 5.2.2 presents an attack on [203, 187] using access-pattern leakage from search queries, and Section 5.2.3 presents an attack on the same constructions if access pattern leakage from update queries is available too. We show that the latter attack requires an order of magnitude less queries (assuming a uniform distribution of search and update queries) for it to succeed.

In Section 5.3, we demonstrate viability of volume leakage attacks with much less information than what is required by the previous attacks. We also develop a new volume leakage attack for schemes with non-standard volume leakage. Section 5.3.1 presents a distribution reconstruction attack where only queries from 'small windows' are observed. Section 5.3.2 demonstrates the viability of distribution reconstruction attacks even if there are further variations to the small-window assumption. Section 5.3.3 shows that elementary volumes which are critical in previous attacks [102, 87, 111] are not needed for distribution reconstruction attacks. Section 5.3.4 shows how to reconstruct a segment of a database uniquely if full distribution reconstruction is not possible. Section 5.3.5 shows how side information can be used to improve the attacks above. Section 5.3.6 generalises the basic attack in Section 5.3.1 to constructions [58, 57] which try to defend against volume-leakage attacks.

A summary of the results is shown in Table 5.1.

Section	**Leakage**	**Goal**	**Query Complexity**	**Targets**
5.2.2	AP from search queries	QR & DBR	$\mathcal{O}(N^2 \log(N))$ SQ	[203, 187]
5.2.3	AP from search and update queries	QR & DBR	$\mathcal{O}(N \log(N))$ SQ and $\mathcal{O}(N \log(N))$ UQ	[203, 187]
5.3.1	V from windows smaller than b	DR	$\mathcal{O}(b \log(N))$ SQ	-
5.3.2	Noisy V from windows smaller than b	DR	$\mathcal{O}(b \log(N))$ SQ	-
5.3.3	V from windows sizes between a and b	DR	$\mathcal{O}(b \log(N))$ SQ	-
5.3.4	V from windows smaller than b	Partial DR	$\mathcal{O}(b \log(N))$ SQ	-
5.3.5*	Noisy V from windows smaller than b	DR	$\mathcal{O}(b \log(N))$ SQ	-
5.3.6	V from search queries	DR & Approx. DR	$\mathcal{O}(N^2 \log(N))$ SQ	[58, 57]

Table 5.1: Acronyms in the table: access pattern (AP); volume (V); query recovery (QR); database reconstruction (DBR); distribution reconstruction (DR); search query (SQ); update query (UQ). The query complexity (i.e. the number of queries required to meet the requirements of the attack) is computed based on uniformly distributed queries. *: The key difference between the attacks in Section 5.3.5 and 5.3.2 is that the attacker in Section 5.3.5 is assumed to have some side information.

Finally, in Section 5.4, we discuss what we have learnt from the attacks and highlight the leakages to avoid when designing a structured encryption scheme that supports range queries.

5.2 Access-pattern Leakage Attacks

This section shows how access-pattern leakage from range queries can be exploited. Earlier literatures [102, 115, 88] have shown how this can be done on a naive construction where the original access pattern is leaked. We extend this class of attacks to constructions with countermeasures.

5.2.1 Background

Full Database Reconstruction Attack. In an access-pattern leakage attack, we assume there is an honest-but-curious adversary who has access to the transcripts of the queries but is unable to decrypt the documents. Without loss of generality, we can write the encrypted database as a set of encrypted documents: $\mathbf{EDB} = \{edoc_i\}$, and there is a function $\mathsf{val}(\cdot)$ which associates a *value* between 1 and N to each document. Upon observing a sequence of queries, that is, for each query, the adversary gets to see a set of encrypted documents returned by the query, the goal of the adversary is to recover the values of the documents. We call this attack *full database reconstruction attack*.

Target leakage profile. In previous attacks [102, 115, 88], it was shown that such an attack is viable on a naive construction, where for a search query $(\mathbf{Srch}, (i, j))$ on values from i to j results in leakage

$$\mathcal{L}_0(\mathbf{Srch}, (i, j)) = \{edoc \in \mathbf{EDB} \mid \mathsf{val}(edoc) \in [i, j]\}\,.$$

These attacks have motivated binary-tree-based constructions [203, 187] which tried to hide the leakage. These constructions work as follows:

1. A leaf node $T_{i,i}$ is constructed by the encryptions of documents with value i.
2. Nodes above are created by iteratively merging the nodes below. The documents are encrypted under fresh randomness so the edges of the tree are not directly leaked.

Figure 5.1 gives a visual representation of the constructions.

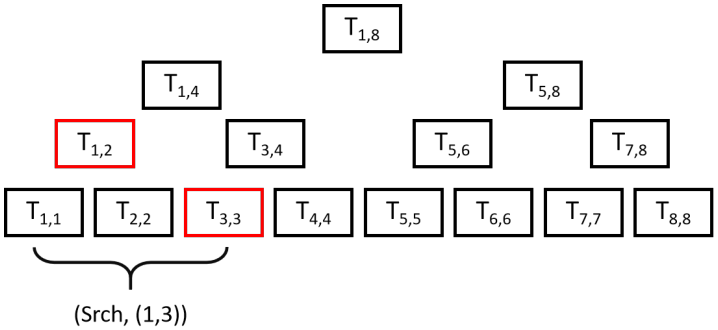


Figure 5.1: The binary tree built for answering range query. $T_{i,j}$ contains all encrypted documents with value between i and j. To answer a search query $(\mathbf{Srch}, (i, j))$, the minimum set of nodes that covers the range is returned as the query response. As an example, to retrieve all documents with values between 1 and 3, nodes $T_{1,2}$ and $T_{3,3}$ are returned.

To answer a query, a technique called best range cover (Algorithm 5.1) is used. This technique finds the minimum number of nodes required to answer a query, and hence, achieves optimal communication complexity in this case.

Algorithm 5.1 Best range cover

```
1: procedure BRC(i, j)
2:     if i > j then
3:         return ∅
4:     w = 2^⌊log2(j−i+1)⌋
5:     i' = ⌈i/w⌉ ∗ w + 1
6:     return (i', i' + w) ∪ BRC(i, i' − 1) ∪ BRC(i' + w + 1, j)
```

There are two complications to the description above. Firstly, instead of encryptions of the actual documents, only the document identifiers are stored in this search index. This is because duplication is expensive in terms of storage and people want to avoid it. On the other hand, if only the document identifiers were used in the search index, and the actual (encrypted) documents were retrieved from the server, the adversary effectively has access to leakage profile $\mathcal{L}_0$ which we know is insecure for any construction. In this section, we show that even if the scheme above has been applied on the documents directly, an attacker can achieve query reconstruction.

Secondly, although the constructions are motivated by a binary tree, the search index does not take shape of one. In particular, the edges are not part of the search index as otherwise it defeats the purpose of the constructions. In [203, 187], the tree is wrapped in a single-keyword search index, that is, the nodes are used as the keywords and the contents of the nodes are used as the documents (identifiers) associated to the keywords. Nonetheless, the leakage of the constructions with a wrapper is exactly the same as that of a binary tree, which allows us to use the latter in our leakage analysis.

These constructions also support document insertions and deletions. As one might expect, an insertion (or deletion) affects all nodes containing the value of the document. For example, in Figure 5.1, if we want to insert a document with value 1, we have to update nodes $T_{1,1}, T_{1,2}, T_{1,4}$ and $T_{1,8}$. [203, 187] using forward and backward-secure searchable encryption schemes as the wrappers so the identity of the nodes will not be immediately leaked after a update query, but a future query on the nodes will reveal this piece of information. For simplicity, we assume that the identity of the nodes are leaked immediately after an update query.

Formally, we can write the leakage $\mathcal{L}_1$ of the binary-tree-based constructions as:

$$\begin{aligned}
\mathcal{L}_1(\mathbf{Srch}, (i,j)) &= \{(T_{i',j'}, |T_{i',j'}|) \mid (i',j') \in \mathbf{BRC}(i,j)\}, \\
\mathcal{L}_1(\mathbf{Insert}, (i, doc)) &= \{(T_{i',j'}, |T_{i',j'}|) \mid 0 \le k \le \log(N), \\
&\qquad i' = \lfloor (i-1) \cdot 2^{-k} \rfloor \cdot 2^k + 1, j' = i' + 2^k\}, \\
\mathcal{L}_1(\mathbf{Delete}, (i, doc)) &= \{(T_{i',j'}, |T_{i',j'}|) \mid 0 \le k \le \log(N), \\
&\qquad i' = \lfloor (i-1) \cdot 2^{-k} \rfloor \cdot 2^k + 1, j' = i' + 2^k\}.
\end{aligned}$$

The type of operation is leaked by the constructions but we omit them in the description of the leakage function for readability.

Density of the database. We say that a database $\mathbf{DB}$ is *dense* if $|\mathbf{DB}[i]| \neq 0$ for all i. We say that a database is *sparse* if it is not dense. If a database is sparse, then

some of the search queries may return no result. [203] and [187] leak this explicitly and our access-pattern leakage attack is compatible with sparse databases.

Access-pattern Leakage Attacks. We present two full database reconstruction attacks in this section. The first attack uses access-pattern leakage from search query only and the number of uniformly distributed queries required for the attack to succeed with high probability is $\mathcal{O}(N^2 \log(N))$, where N is the maximum value of the database. The second attack uses access-pattern leakage from search query and update query, and the attack is expected to succeed with $\mathcal{O}(N \log(N))$ uniformly distributed update queries and $\mathcal{O}(N \log(N))$ uniformly distributed search queries.

As the search pattern leakage is not obscured in [203, 187], a successful full database reconstruction attack implies a full query recovery attack. In particular, for the second attack, this means that the attacker can recover the update history of the database.

5.2.2 Full Database Reconstruction Attack using Access-pattern Leakage from Search Query

Attack Setting. The attack we describe in this section works on leakage profile $\mathcal{L}_1$. The adversary only uses the leakage from the search queries so the attack is applicable to the static and dynamic constructions at the same time.

We assume that an honest client uses one of the schemes that satisfies leakage profile $\mathcal{L}_1$ to make a sequence of k randomly generated search queries $\mathbf{q}_1, \ldots, \mathbf{q}_k$. The attacker records the leakages of the queries as $\mathcal{L}_1(\mathbf{q}_1), \ldots, \mathcal{L}_1(\mathbf{q}_k)$, and his goal is to reconstruct the database. More specifically, by the end of the attack, the adversary outputs an array A of length N where N is the maximum value in the database, such that $A[i]$ contains all documents with value i up to reflection (except $A[1]$ and $A[N]$ which are empty for reasons we will discuss later). Here, the documents in array A must only come from the leaf nodes of the binary tree.

The reconstruction of the leaf nodes allows an attacker to trivially reconstruct the values of the other nodes, by exploiting appropriate query leakages. We omit the details of how this can be done. We also note that a database reconstruction attack for leakage profile $\mathcal{L}_1$ allows for trivial query reconstruction. Details of this is omitted from the thesis for the same reason.

If the underlying scheme supports update queries, then two queries on the same range may not give the same leakage. For simplicity, we assume all queries issued by the client are search queries. The attack can be easily generalised to the dynamic setting where some of the queries are insertion or deletion queries. To demonstrate that, consider an insertion query with a document that has value 1. This translates to document insertions on nodes of the shape $T_{1,2^i}$ for $i = 0, \ldots, \log(N)$ which means that the attacker can always undo the insertion and use a canonical version of the leakage.

We note that the query response volumes are not obscured in leakage profile $\mathcal{L}_1$ which means all previous volume-leakage attacks [102, 115, 88] apply immediately. However, our proposed attack uses access-pattern leakage of the search queries only which means that even if fake documents are introduced, as long as access-pattern is not obscured, our attack applies to that scheme.

To simplify our notation in the description of the attack, we write $\mathcal{L}_1(\mathbf{Srch}, (i,j)) = \{T_{i',j'} \mid (i',j') \in \mathbf{BRC}(i,j)\}$.

INTUITION. Let $r_{i,j}$ be the number of distinct queries that retrieve node $T_{i,j}$. We show that $r_{i,j}$ differs for different choices of i and j, which means if the adversary has observed all queries at least once, he can infer the nature of the nodes by simply computing $r_{i,j}$ for each node. As an abuse of notation, we write $r(T_{i,j}) = r_{i,j}$.

We categorises the nodes into three groups, namely left leaf node, right leaf node and non-leaf node. The left leaf nodes are nodes of the shape $T_{i,i}$ where $i = 1 \bmod 2$ if $i \leq \frac{N}{2}$ and $i = 0 \bmod 2$ if $i > \frac{N}{2}$. The right nodes are the leaf nodes that are not left leaf nodes. The non-leaf nodes are the remaining nodes. The node categorisation is illustrated in Figure 5.2.

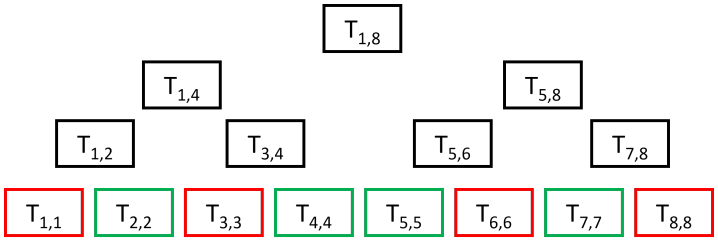


Figure 5.2: In the figure, red nodes are left nodes, green nodes are right nodes and black nodes are non-leaf nodes.

For a left leaf node $T_{i,i}$ with $i \leq \frac{N}{2}$, the node will only be covered by query ranges of the shape (j,i) where $1 \leq j \leq i$. This is because for any query with right endpoint greater than i, a node above $T_{i,i}$ will be retrieved by the query instead. Hence, the number of distinct queries that retrieve a left leaf node $T_{i,i}$ is simply

$$r_{i,i} = i \text{ if } i = 1 \bmod 2.$$

Using a similar argument on the right leaf nodes, we have that

$$r_{i,i} = N - i + 1 \text{ if } i = 1 \bmod 2.$$

This can be seen easily from Figure 5.3.

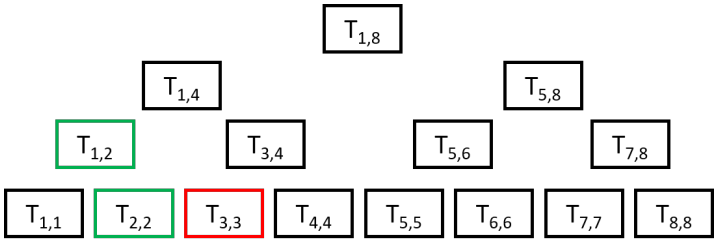


Figure 5.3: Node $T_{3,3}$ is a left leaf node. It appears in search queries with endpoints $(1,3), (2,3)$ and $(3,3)$. The other nodes that can appear together with node $T_{3,3}$ are highlighted in green.

We can use the same counting argument on the non-leaf nodes and get $r_{i,j}$ for $i \neq j$ as:

$$r_{i,j} = \begin{cases} N - i + 1 & \text{if } \frac{j}{j-i+1} = 0 \bmod 2, \\ j & \text{if } \frac{j}{j-i+1} = 1 \bmod 2 \text{ and } j - i + 1 \neq N, \\ 1 & \text{otehrwise.} \end{cases}$$

This is demonstrated in Figure 5.4.

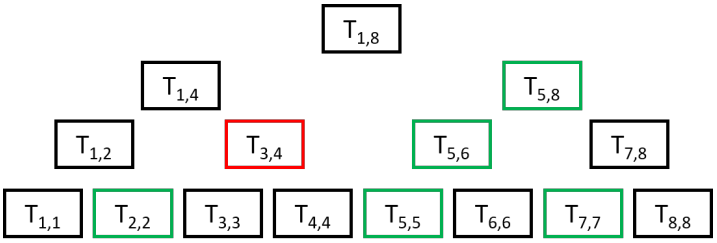


Figure 5.4: Node $T_{3,4}$ is a non-leaf node. It appears in search queries with endpoints $(2,4),(3,4),(3,5),(3,6),(3,7)$ and $(3,8)$. The other nodes that can appear together with node $T_{3,4}$ are highlighted in green.

Observe that all $r_{i,i}$ are all odd whereas all $r_{i,j}$ for $i \neq j$ are even except $r_{1,N}$, this means that the adversary can immediately differentiate the leaf nodes from the non-leaf ones except the root node. Furthermore, $r_{i,i}$ is unique up to reflection, i.e. $r_{i,i} = r_{j,j}$ if and only if $i + j = N + 1$, so the leaf nodes can be re-identified up to reflection using $r_{*,*}$ only. Finally, to disambiguate the reflection, we can fix $T_{2,2}$ and $T_{N-1,N-1}$ and check if there is a query where $T_{2,2}$ appears together with the node in interest. For example, $T_{3,3}$ appears together with $T_{2,2}$ in the search query $(\mathbf{Srch}, (2,3))$, but $T_{N-2,N-2}$ does not.

The procedure above allows us to re-identify all the leaf nodes except $T_{1,1}$ and $T_{N,N}$. To identify the other nodes, we can simply use the leakage from some of the queries. Without loss of generality, let $T_{i,j}$ where $i \neq 1$ and $j \neq N$ be the node the adversary is trying to re-identify. As we assume that all the queries have been issued once, there must be a search query of the shape $(\mathbf{Srch}, (i-1, j+1))$. The leakage of the query is $\mathcal{L}_0(\mathbf{Srch}, (i-1, j+1)) = \{T_{i-1,i-1}, T_{i,j}, T_{j+1,j+1}\}$. This is the only query containing $T_{i-1,i-1}$, $T_{j+1,j+1}$ and another node, so $T_{i,j}$ can be easily re-identified.

Attack Overview. The attack algorithm $\mathcal{A}_1$ is given inputs a sequence of query leakages $L = (\ell_1, \ldots, \ell_k)$ and the maximum value of the database N. The output of the algorithm is an array A of leaf nodes with values from 2 to $N-1$. The first and last positions in A are marked by $\perp$ as the two leaf nodes $T_{1,1}$ and $T_{N,N}$ cannot be uniquely re-identified. The algorithm proceeds in two steps as shown in Algorithm 5.2. In the first step, the algorithm removes duplicate queries and identifies all the nodes of the encrypted database. In the second step, the r-value of each node is computed and the leaf nodes are re-identified. Query reconstruction and database reconstruction from A are trivial and they are omitted from the pseudocode.

Theoretical Analysis. The attack described requires all queries involving the leaf nodes as part of the leakage to be observed at least once. Any missing query would change the values of $r_{*,*}$ computed by the attacker, hence leading to the failure of

Algorithm 5.2 Access-pattern Leakage Attack using Search Query Leakage

```
1:  procedure A_1(L, N)
2:      /* Convert L into a set */
3:      L ← {ℓ ∈ L}
4:      /* Set T to be the union of the query tokens */
5:      T ← ∪_{ℓ∈L} ℓ

6:      /* Initialise an array of size N for the leaf nodes */
7:      A ← [⊥ for i ∈ 1, ..., N]
8:      /* Fix the reflection with two of the leaf nodes */
9:      {t_1, t_2} ← {t ∈ T | r(t) = N − 1}
10:     A[2] ← t_1
11:     A[N − 1] ← t_2
12:     /* Fill the other leaf nodes */
13:     for i ← 3, ..., N/2 do
14:         {t_1, t_2} ← {t ∈ T | r(t) = i}
15:         if (i mod2 = 0) ∧ (∃ℓ ∈ L, {t_1, A[2]} ∈ ℓ) then
16:             A[i] ← t_2, A[N − i + 1] ← t_1
17:         else if (i mod2 = 0) then
18:             A[i] ← t_1, A[N − i + 1] ← t_2
19:         else if (i mod2 = 1) ∧ (∃ℓ ∈ L, {t_1, A[2]} ∈ ℓ) then
20:             A[i] ← t_1, A[N − i + 1] ← t_2
21:         else
22:             A[i] ← t_2, A[N − i + 1] ← t_1
23:     return A
```

the reconstruction algorithm. More concretely, the number of queries required for the algorithm to succeed is $\mathcal{O}(N^2 \log(N))$ and we state it as a theorem below.

Theorem 5.1 (Number of queries required for Attack $\mathcal{A}_1$ to succeed)**.** Suppose that the queries follow a uniform distribution. Then the number of queries required for attack $\mathcal{A}_1$ to succeed with high probability is $\mathcal{O}(N^2 \log(N))$.

Proof. We know that the number of distinct queries is $\frac{N(N+1)}{2}$. As the queries are uniformly distributed, the number of queries required to observe all distinct queries at least once can be formulated as a coupon collector's problem [71] with $\frac{N(N+1)}{2}$ coupons. A standard argument shows that the number of queries required is $\frac{N(N+1)}{2} H_{\frac{N(N+1)}{2}} = \mathcal{O}(N^2 \log(N))$, where H_n is the n-th harmonic number. □

5.2.3 Full Database Reconstruction Attack using Access-pattern Leakage from Search Query and Update Query

Attack Setting The attack we describe in this section works on leakage profile $\mathcal{L}_1$ in the dynamic setting. Just like the previous attack $\mathcal{A}_1$, we assume that the adversary who is honest-but-curious observes random queries generated by the honest client. These queries are a mixture of search queries, insertion queries and deletion queries. The goal of the adversary is to reconstruct the database just like before. More specifically, by the end of the attack, the adversary outputs an array A of length N where N is the

maximum value in the database, such that $A[i]$ contains all documents with value i up to reflection (with $A[1]$ and $A[N]$ as the leakage from update queries allows for it). Here, the documents in array A must only come from the leaf nodes of the binary tree.

There are two key differences between this attack and the previous one. Firstly, this attack will only work if there is at least one update query on each value, which means it is not applicable to static schemes. Secondly, this attack is capable to re-identify the two endpoints ($A[1]$ and $A[N]$) which attack $\mathcal{A}_1$ cannot identify.

As before, we omit the volume leakage from the leakage profile and use the access pattern leakage only. For simplicity, we assume that the nodes in the queries are not changed by the insertion or deletion queries.

INTUITION To motivate the attack, consider a database with $N = 8$. Imagine that the client inserts a document doc_1 with value 1 and a document doc_2 with value 2. Then the leakages of the two insertion queries are

$$\mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (1, doc_1)) = \{T_{1,1}, T_{1,2}, T_{1,4}, T_{1,8}\},$$
$$\mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (2, doc_2)) = \{T_{2,2}, T_{1,2}, T_{1,4}, T_{1,8}\}.$$

So $|\mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (1, doc_1)) \cap \mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (2, doc_2))| = 3$.

On the other hand, suppose that the client also inserts a document doc_3 with value 3, then

$$\mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (3, doc_3)) = \{T_{3,3}, T_{3,4}, T_{1,4}, T_{1,8}\}.$$

So

$$\begin{aligned} &|\mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (1, doc_1)) \cap \mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (3, doc_3))| \\ =&|\mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (2, doc_2)) \cap \mathcal{L}_{\mathbf{Insert}}(\mathbf{Insert}, (3, doc_3))| \\ =&2. \end{aligned}$$

Straightforwardly, the insertions are nothing but the common ancestries between the leaf nodes touched by the insertion queries, and by computing the cardinalities of the insertions, the adjacencies between the leaf nodes can be learnt immediately.

The nature of the other nodes are revealed too. For example, node $T_{1,8}$ is the only node covered by all insertion queries, node $T_{1,4}$ and $T_{5,8}$ are the only two nodes covered by four insertion queries.

It remains to recover the order of the leaf nodes up to reflection. This can be done by fixing a pair of adjacent leaf nodes (T_1, T_2) and extend it iteratively using the leakages from the search queries. Without loss of generality, assume that node T_3 and T_4 are adjacent nodes and there is a search query with leakage $\{T_2, T_5, T_4\}$. Then we know that the order of nodes T_1, T_2, T_3, T_4 must be T_1, T_2, T_4, T_3, with a gap between node T_2 and T_4 as the query width of node T_5. As we can recover the query width of node T_5 by looking at how many times it is covered by unique insertion queries, we have recovered the order of nodes T_1, T_2, T_3, T_4 completely. The process can then be repeated until the order of all leaf nodes are fixed. The non-leaf nodes can be re-identified easily by using the ancestry information of the leaf nodes.

ATTACK OVERVIEW The attack algorithm $\mathcal{A}_2$ is presented in Algorithm 5.3. The attack takes as input a sequence of leakages $L = (\mathcal{L}_1(\mathbf{q}_1), \ldots, \mathcal{L}_1(\mathbf{q}_k))$ and the maximum value

N, and outputs an array B that contains the leaf nodes of the binary tree in order (up to reflection). The attack is broken down into three steps. Firstly, the leakages are separated by their query types. Secondly, the adjacent leaf nodes are identified. Thirdly, the leaf nodes are ordered using the access-pattern leakage from the search queries. For readability, the detailed pseudocode for the third step is omitted. Query reconstruction and database reconstruction from B are trivial and they are omitted from the pseudocode.

Algorithm 5.3 Access-pattern Leakage Attack using Search Query and Update Query Leakage

```
1:  procedure A_2(L, N)
2:      L_1 ← access-pattern leakage from the update queries
3:      L_2 ← access-pattern leakage from the search queries

4:      /* Identify adjacent leaf nodes */
5:      A ← {}
6:      for ℓ_1 ∈ L_1 do
7:          for ℓ_2 ∈ L_1, ℓ_2 ≠ L_1 do
8:              if ℓ_1 ∩ ℓ_2 = log(N) then
9:                  A ← A ∪ {{ℓ_1 − ℓ_2, ℓ_2 − ℓ_1}}

10:     B ← [⊥ for i ∈ 1, ..., N]
11:     B ← ordered leaf nodes from A using L_2
12:     return B
```

Theoretical Analysis. The attack described requires an update query on every value and a few more search queries. We assume that both types of queries follow uniform distributions and we provide a theorem on the number of queries of each type required below.

Theorem 5.2 (Number of queries required for Attack $\mathcal{A}_2$ to succeed)**.** Suppose that the search queries and update queries follow uniform distributions. Then the number of search queries required for attack $\mathcal{A}_2$ to succeed with high probability is $\mathcal{O}(N \log(N))$ and the number of update queries required for that is $\mathcal{O}(N \log(N))$.

Proof. The proof is separated into two parts. In the first part, we show that the number of update queries required is $\mathcal{O}(N \log(N))$. Then, we show that the number of search queries required is $\mathcal{O}(N \log(N))$ too.

(**# update queries**) The number of distinct update queries is N. By assuming that the update queries are uniformly distributed and apply coupon collector's problem, we have that the number of queries required to observe all update queries at least once is $\mathcal{O}(N \log(N))$.

(**# search queries**) The search queries are used to fix the orientation of the adjacent nodes recovered from update query leakage. There are $\frac{N}{2}$ pairs of adjacent nodes, and for every two pairs of adjacent nodes, there is a single query that fixes the orientation between them. In that light, the problem of finding the orientation of the pairs of adjacent nodes is equivalent to making a connected graph where the nodes are the pairs of adjacent nodes and the edges are the queries that fixes the respective orientation between the pairs of adjacent nodes.

Let T_i be the number of queries required to reduce the number of connected components of the graph from i to $i-1$. Then the number of queries required to fix the orientation of all pairs of adjacent nodes can be expressed as $\sum_{i=2}^{\frac{N}{2}} T_i$. We will give an upper bound on the expectation of $\mathbf{E}[T_i]$ for each i and then combine the expectations to give an upper bound of the expectation of T.

Suppose that there are i connected components in the graph, and the number of nodes in the connected components are $x_1, \ldots, x_i$, then there are

$$\sum_{j=1}^{i} x_j \cdot \left(\frac{N}{2} - x_j\right)$$

queries that can reduce the number of connected components to $i-1$. With $\sum_j x_j = \frac{N}{2}$, the expression can be simplified to

$$\left(\frac{N}{2}\right)^2 - \sum_{j=1}^{i} x_j.$$

This expression can be bounded from below by

$$\begin{aligned}
&\left(\frac{N}{2}\right)^2 - \left(\frac{N}{2} - i + 1\right)^2 - (i-1) \\
=&N(i-1) - (i-1)(i-2) \\
=&(i-1)(N-i+2).
\end{aligned}$$

As i runs from 2 to $\frac{N}{2}$ in the summation, $N - i + 2 > \frac{N}{2}$ for all $i \in \left\{2, \ldots, \frac{N}{2}\right\}$, so

$$\sum_{j=1}^{i} x_j \cdot \left(\frac{N}{2} - x_j\right) > (i-1) \cdot \frac{N}{2}.$$

As there are $\frac{N(N+1)}{2}$ possible queries in total and the queries are uniformly randomly distributed, the probability that a query reduced i connected components to $i-1$ of them is at least

$$\begin{aligned}
&\frac{(i-1) \cdot \frac{N}{2}}{\frac{N(N+1)}{2}} \\
=&\frac{i-1}{N+1}.
\end{aligned}$$

This implies that the expectation of T_i is at most $\frac{N+1}{i-1}$. Thus,

$$\begin{aligned}
\mathbf{E}[T] &= \sum_{i=2}^{\frac{N}{2}} \mathbf{E}[T_i] \\
&< \sum_{i=2}^{\frac{N}{2}} \frac{N+1}{i-1} \\
&< (N+1) H_{\frac{N}{2}} \\
&= \mathcal{O}(N \log(N)).
\end{aligned}$$

□

5.3 Volume Leakage Attacks

In this section, we describe and analyse our volume leakage attacks. In Section 5.3.1, we begin with a simple attack where the attacker is assumed to be able to observe all volumes from *small* windows. In Section 5.3.2, we apply additional variations on top of the small-window assumption, and show that the attacker can still reconstruct a significant amount of databases uniquely. In Section 5.3.3, we show that the *elementary* volumes are not required for the reconstruction attack. In Section 5.3.4, we devise a *partial* reconstruction attack for databases that are not fully reconstructable. We show how side information can help in improving the success rate of the attacks above in Section 5.3.5. Finally, we demonstrate in Section 5.3.6 how the ideas in our attacks can be used on similar but different volume leakage profiles that have not been broken in the previous attacks.

5.3.1 Basic Attack

In this section, we describe and analyse our basic reconstruction attack. We assume that only queries of small windows are observed by the attacker. More formally, we define the leakage function as

$$\mathcal{L}_b(\mathbf{DB}) = \left\{ \sum_{i=x}^{y} v_i \mid \ y - x + 1 \leq b \right\}. \tag{5.1}$$

The adversary then learns $W = \mathcal{L}_b(\mathbf{DB})$ for some database $\mathbf{DB} = (v_1, \ldots, v_N)$; his goal is to recover $\mathbf{DB}$. By setting $b = N$, we recover the leakage function in the previous attacks [102, 87]. It is the only case for which previous attacks work: both attacks require volumes generated by *all* possible queries. On the other hand, our attack works with considerably smaller b.

5.3.1.1 Reconstruction Algorithm

Intuition. Given a tuple of observed volumes, say $(w_1, \ldots, w_k)$, we can tell if it looks like a subarray of $\mathbf{DB}$, since we know a set of conditions need to be satisfied: all the sums of two consecutive volumes need to be present in W; similarly, the sums of three to b consecutive volumes need to be present in W. Given a large number of constraints, it is unlikely that an arbitrary choice of $(w_1, \ldots, w_k)$ meets all the constraints. Hence, a partial solution of length k is likely to be a continuous segment of $\mathbf{DB}$.

Our basic attack is a variant of breadth-first search with pruning which extends partial solutions iteratively. There are several choices for the initial partial solutions. For certain databases, the minimum observed volume suffices as an initial partial solution. This choice may not work on some databases as the number of partial solutions grows too quickly. In that case, we opt for a clique-finding subroutine inspired by GLMP18 [87] to generate partial solutions of length b. In each iteration, our attack tries to extend the partial solutions found in the previous iteration to the left and right by checking the new constraints introduced by the new volume. Partial solutions that cannot be extended by any observed volume are discarded. After obtaining the partial solutions of length N, we check if the solutions generate the expected leakage, and the solutions that

do not are discarded. The remaining solutions are returned by our attack. Pseudocode of our attack is shown in Algorithm 5.5.

Initial Solution via Clique-finding. An obvious starting partial solution is the smallest observed volume: since all elementary volumes are observed the smallest volume is necessarily an elementary volume. Looking ahead, this strategy may not work (e.g. when the smallest volume is suppressed). We therefore employ a more robust mechanism inspired by the GLMP18 attack and which shares some ideas with a heuristic which complements the KKNO16 attack. Below, we describe how an initial solution can be identified by finding a clique in a certain graph. Let W be the set of observed volumes. We define the set of complemented volumes C to be $\{v \mid v \in W \wedge \max(W) - v \in W\} \cup \{\max(W)\}$. Nodes of our graph are defined by the elements of C. For $v_1, v_2 \in C$, there is an edge between nodes v_1 and v_2 if $|v_1 - v_2| \in W$. We know $\max(W)$ is the sum of b elementary volumes, so a clique of size b with one of the volumes as $\max(W)$ describes a partial solution of length b.

In GLMP18 the reconstruction of the entire database is reduced to finding a clique (with N nodes). Here we only use clique finding to initialize our search, so we only need to recover a small segment of size b of the database.

Detailed pseudocode of our clique-finding algorithm can be found in Algorithm 5.4. In line 13, we use $g\langle i\rangle$ to mean i-th smallest volume in g. It is also worth pointing out that our algorithm employs a similar pruning strategy suggested in Appendix E of the KKNO16 paper [102]. Without going into too many details, that algorithm verifies similar constraints with our pruning mechanism to identify a plausible segment of the hidden database.

Algorithm 5.4 Finding initial solution set

```
1:  input W = {∑_{i=x}^{y} v_i | y − x + 1 ≤ b}, b
2:  output {(w_1, ..., w_b) | w ∈ W ∧ L_b(w_1, ..., w_b) ⊂ W ∧ ∑_i w_i = max(W)}

3:  procedure Initial Solution(W, b)
4:      C = {v | v ∈ W, max(W) − v ∈ W}
5:      G_2 = {{v, max(W)} | v ∈ C}
6:      for i ← 3, b do
7:          G_i = {}
8:          for g ∈ G_{i−1}, v ∈ C do
9:              if {|h − v| | h ∈ g} ⊂ W then
10:                 G_i = G_i ∪ {g ∪ {v}}
11:     S = {}
12:     for g ∈ G_b do
13:         s = (g⟨1⟩, g⟨2⟩ − g⟨1⟩, ..., g⟨b⟩ − g⟨b − 1⟩)
14:         S = S ∪ {s}
15:     return S
```

A Running Example. Let $\mathbf{DB} = (11, 5, 12, 20, 19)$ and $b = 3$, we get

$$W = \{5, 11, 12, 16, 17, 19, 20, 28, 32, 37, 39, 51\}.$$

According to Algorithm 5.4, we start by identifying a set C where for every volume v in C, we can also find another volume $v' \in W$ such that $v + v' = \max W$. For the database

in the example, $C = \{12, 19, 32, 39\}$.

The algorithm then proceeds to find $G_2 = \{\{12, 51\}, \{19, 51\}, \{32, 51\}, \{39, 51\}\}$ which is essentially a set of sets where each set contains one of the volumes in C and the maximum volume. These sets are used as representations of part of the database. For example, $\{12, 51\}$ is a representation of $(v_3, v_4 + v_5)$ in the form $\{v_3, v_3 + v_4 + v_5\}$.

Our goal in the next step is to refine G_2 so that we find a set $\{v_3, v_3 + v_4, v_3 + v_4 + v_5\}$. This is done by testing every volume in C, and check if it fits in the sets in G_2. Using the set $\{v_3, v_3 + v_4 + v_5\}$ as an example, the absolute differences between the new volume and other volumes in the set have to be observed volumes. For $\{12, 51\}$, the volume that satisfies the conditions is 32, so one of the sets in G_3 is $\{12, 32, 51\}$. For this example, we get two sets in G_3 where the other set is $\{19, 30, 51\}$. If b is larger, we need to carry this step iteratively until the sets have size b.

Finally, to extract the partial solutions from G_b, we order the volumes in the sets and take differences in the way that is shown in line 13 of Algorithm 5.4. This step produces two partial solutions in the example which are $(12, 20, 19)$ and $(19, 20, 12)$. These two solutions are reflections of each other and we will always get them from Algorithm 5.4.

Attack Overview. Pseudocode of our basic attack is shown in Algorithm 5.5. Our clique-finding algorithm (Algorithm 5.4) is used as the initial solution finding algorithm but other choices are possible. In line 4, the set of initial solutions are computed. The partial solutions are then iteratively extended by running procedures Extend Left and Extend Right. In these two procedures, only the sums involving the new volume w has to be checked, as all the other constraints are checked in previous iterations. After obtaining the partial solutions of length N, we recompute the leakage associated to each one and discard those for which the result is different from the input to the algorithm.

A Running Example (continued). Recall from above that we have $\mathbf{DB} = (11, 5, 12, 20, 19)$ and $b = 3$, and the set of observed volumes is

$$W = \{5, 11, 12, 16, 17, 19, 20, 28, 32, 37, 39, 51\}.$$

We have obtained two initial solutions $(12, 20, 19)$ and $(19, 20, 12)$ from the initial solution finding algorithm. Our basic attack takes these partial solutions and tries to extend them. Using $(12, 20, 19)$ as an example, the algorithm will check if $(w, 12, 20, 19)$ is a valid partial solution for all $w \in W$, so does all partial solutions of the shape $(12, 20, 19, w)$. For instance, $(11, 12, 20, 19)$ cannot be a partial solution of length 4 as $11 + 12 \notin W$, so it is eliminated by the algorithm. Only $(5, 12, 20, 19)$ and $(19, 20, 12, 5)$ are a plausible partial solutions of length 4 after this iteration. The process is continued until solutions of length N are found. For the example, only one more iteration is required to produce solutions of length $N = 5$.

Before outputting these solutions, the algorithm checks if the observed volumes generate the exact set of observed volumes. In the example, the only two solutions of length 5 are $(11, 5, 12, 20, 19)$ and $(19, 20, 12, 5, 11)$ and they pass the test expectedly. As a result, $(11, 5, 12, 20, 19)$ and $(19, 20, 12, 5, 11)$ are the solutions output by Algorithm 5.5.

Algorithm 5.5 Basic attack

1: **input** $W = \{\sum_{i=x}^{y} v_i \mid y - x + 1 \leq b\}, b, N$
2: **output** $\{(w_1, \ldots, w_N) \mid w_i \in W\}$

3: **procedure** ATTACK(W, b, N)
4: $S_b = \text{INITIAL SOLUTION}(W, b)$
5: **for** $i \leftarrow b+1, N$ **do**
6: $S_i \leftarrow \text{EXTENDLEFT}(S_{i-1}, W, b) \cup$ $\text{EXTENDRIGHT}(S_{i-1}, W, b)$
7: $S_N \leftarrow \{s \mid s \in S_N \wedge \mathcal{L}_b(S_N) = W\}$
8: **return** S_N

9: **procedure** EXTEND LEFT(S_i, W, b)
10: $S' \leftarrow \{\}$
11: **for all** $(w_1, \ldots, w_i) \in S_i, w \in W$ **do**
12: **if** $\{w + \sum_{j=1}^{k} w_j \mid k < b\} \subseteq W$ **then**
13: $S' \leftarrow S' \cup \{(w, w_1, \ldots, w_i)\}$
14: **return** S'

15: **procedure** EXTEND RIGHT(S_i, W, b)
16: $S' \leftarrow \{\}$
17: **for all** $(w_1, \ldots, w_i) \in S_i, w \in W$ **do**
18: **if** $\{w + \sum_{j=1}^{k} w_{i-j+1} \mid k < b\} \subseteq W$ **then**
19: $S' \leftarrow S' \cup \{(w_1, \ldots, w_i, w)\}$
20: **return** S'

5.3.1.2 Correctness

For convenience, we denote our basic attack algorithm as $\mathcal{A}_1$. It takes as input the set of observed volumes $W = \mathcal{L}_b(\mathbf{DB})$, bound b and maximum label N. We say that $\mathcal{A}_1$ is correct if the output of the attack is precisely the set of databases that generates the given leakage. Formally, the correctness of our attack is established by the following theorem.

Theorem 5.3 (Correctness of the basic attack)**.** *Let $\boldsymbol{DB}$ be a database, $N = |\boldsymbol{DB}|$ and b be any natural number less or equal to N. Let S_N be the output of $\mathcal{A}_1$, i.e. $S_N = \mathcal{A}_1(\mathcal{L}_b(\boldsymbol{DB}), b, N)$. Then*

1. $\forall \boldsymbol{DB}' \in \mathcal{L}_b(\boldsymbol{DB})^N, \mathcal{L}_b(\boldsymbol{DB}') = \mathcal{L}_b(\boldsymbol{DB}) \Leftrightarrow \boldsymbol{DB}' \in S_N$,
2. $\boldsymbol{DB} \in S_N$.

Proof of Theorem 5.3. Statement (2) follows from (1), so it suffices to prove (1).

($\Leftarrow$) We obtain this implication for free as line 7 of the algorithm ensures that $\mathcal{L}_b(\mathbf{DB}') = \mathcal{L}_b(\mathbf{DB})$.

($\Rightarrow$) Assume that $\mathcal{L}_b(\mathbf{DB}') = \mathcal{L}_b(\mathbf{DB})$. We show that at iteration i of the algorithm in line 6, we have the invariant that $\forall \mathbf{DB}' \in \mathcal{L}_b(\mathbf{DB})^N$, $\mathcal{L}_b(\mathbf{DB}') = \mathcal{L}_b(\mathbf{DB}) \implies$

$\exists(x_1, \ldots, x_i) \in \mathbf{DB}'$ s.t. $(x_1, \ldots, x_i) \in S_i$. In particular, this implies that for solutions of length N, $\mathbf{DB}' \in S_N$. We prove this by induction.

The correctness of the base case relies on the correctness of the initial solution finding algorithm. We prove that Algorithm 5.4 is correct. Assume for the sake of contradiction that there exists some clique of size b, $(w_1, \ldots, w_b)$ which is not in the output of the algorithm. Then $\left\{w_1, w_1 + w_2, \ldots, \sum_{j=1}^{b} w_j\right\}$ must not be in G_b. This is impossible as $\left\{w_1, \sum_{j=1}^{b} w_j\right\} \in G_2$ and for i between 3 and b, adding $\sum_{j=1}^{i} w_j$ to the previous set of volumes generates a valid element in G_i. Therefore, our assumption must be false and the correctness of the clique finding algorithm follows.

In the inductive step, we assume that at iteration k, there exists $(x_1, \ldots, x_k) \in \mathbf{DB}'$ such that $(x_1, \ldots, x_k) \in S_k$. Without loss of generality, we can assume x_k is not the last volume to the right of $\mathbf{DB}'$, so there is some x_{k+1} such that $(x_1, \ldots, x_k, x_{k+1}) \in \mathbf{DB}'$. Running the EXTEND RIGHT procedure from line 15 to 20, we check if

$$\left\{x_{k+1} + \sum_{i=0}^{j} x_{k-i} \mid \ 0 \leq j \leq \min\{b-1, k\}\right\} \subseteq \mathcal{L}_b(\mathbf{DB}).$$

This is indeed the case as $\mathcal{L}_b(\mathbf{DB}') = \mathcal{L}_b(\mathbf{DB})$. So there exists $(x_1, \ldots, x_k, x_{k+1}) \in \mathbf{DB}'$ such that $(x_1, \ldots, x_k, x_{k+1}) \in S_{i+1}$.

Hence, we conclude that after the final iteration, $\forall \mathbf{DB}' \in \mathcal{L}_b(\mathbf{DB})^N$, $\mathcal{L}_b(\mathbf{DB}') = \mathcal{L}_b(\mathbf{DB}) \implies \exists(x_1, \ldots, x_N) \in \mathbf{DB}'$ s.t. $(x_1, \ldots, x_N) \in S_i$, and the desired result follows. □

The correctness property of the attack implies that our attack is optimal in terms of database recovery, meaning that there is no algorithm that can eliminate more partial solutions from the set of solutions, if there is no further information.

5.3.1.3 Uniqueness

Correctness of our algorithm only ensures that we recover all the valid solutions from the leakage. It is possible to have multiple databases (up to reflection) generating the same leakage. For example, $\mathcal{L}_3((1,1,1,2,1,1)) = \mathcal{L}_3((2,1,1,1,1,1)) = \{1,2,3,4\}$, so the databases $(1,1,1,2,1,1)$ and $(2,1,1,1,1,1)$ are indistinguishable given the leakage. Yet, our experiments show that our attacks recover unique solutions for an overwhelming amount of real-world databases. In this section we investigate why this is the case. We assume that the databases are randomly sampled from some underlying distribution $\mathcal{V}$ and we are therefore interested in the probability that the database can be reconstructed uniquely from the leakage:

$$p_{\text{unique}} = \Pr_{\mathbf{DB} \leftarrow \mathcal{V}}\left[S \leftarrow \mathcal{A}_1(\mathcal{L}_b(\mathbf{DB}), b, N), \exists s, S = \left\{s, s^R\right\}\right].$$

It becomes immediately clear that the probability is computationally infeasible to compute from its analytical expression. We outline how to use a series of bounding and approximation techniques to find an estimation of the probability. We use a very rough approximation where we are interested in the event that all of the partial solutions discovered/maintained by our attack are *genuine*, that is they are solutions of the form

$(v_i, \ldots, v_j)$ for some i and j or their reflections. First we bound the probability that at iteration b the set of partial solutions contains *only* genuine solutions. Here, we use the Chen-Stein method [44, 43] to estimate the distribution of the observed volumes and derive the probability that an arbitrary choice of b volumes from the set of observed volumes satisfies all the constraints. This allows us to derive the probability that all the partial solutions of length b are genuine solutions (Lemma 5.3). Then, we bound the probability that the solution set contains only genuine solutions at iteration $i+1$ given that all solutions are genuine at iteration i. A bound on p_{unique} follows by observing that the only genuine solutions of length N are $\mathbf{DB}$ and $\mathbf{DB}^R$ (Theorem 5.5).

Our derivation uses the underlying distribution on the database as an abstract parameter. We can then computationally determine concrete values for the bound by instantiating with concrete distributions.

Distribution of observed volumes. We write $\mathcal{W}$ to denote the distribution of observed volumes. Unlike a simple random variable, a draw from $\mathcal{W}$ generates a set of volumes. Given the distribution of elementary volumes as $\mathcal{V} = (V_1, \ldots, V_N)$, and bound b, a draw from $\mathcal{W}$ is equivalent to $\left\{\sum_i^j v_i \mid \ v_i \leftarrow V_i, j - i + 1 \leq b\right\}$. Volumes in $\mathcal{W}$ have significant dependencies amongst each other, and it is not trivial to compute $\mathcal{W}$. We appeal to the works of Stein [172] and Chen [44, 43], later known as the Chen-Stein method. The main result we use is proven by Arratia, Goldstein and Gordon [8].

Theorem 5.4. *Let $U = \sum_{\alpha \in \mathcal{I}} X_\alpha$ be the number of occurrences of dependent events, and let Z be a Poisson random variable with $\boldsymbol{E}[Z] = \boldsymbol{E}[U] = \lambda < \infty$. For each $\alpha \in \mathcal{I}$, we define $B_\alpha \subset \mathcal{I}$ with $\alpha \in B_\alpha$ as the neighbourhood of X_α. The neighbourhood of X_α is the set of random variables that have dependency with X_α. Define*

$$b_1 = \sum_{\alpha \in \mathcal{I}} \sum_{\beta \in B_\alpha} p_\alpha p_\beta,$$

$$b_2 = \sum_{\alpha \in \mathcal{I}} \sum_{\alpha \neq \beta \in B_\alpha} p_{\alpha\beta}, \ \text{where } p_{\alpha\beta} = \boldsymbol{E}[X_\alpha X_\beta],$$

$$b_3 = \sum_{\alpha \in \mathcal{I}} \boldsymbol{E}\left[\boldsymbol{E}[X_\alpha - p_\alpha \mid \ \sigma(X_\beta : \beta \neq B_\alpha)]\right].$$

Then

$$\left|\Pr[U = 0] - e^{-\lambda}\right| < \min(1, \lambda^{-1})(b_1 + b_2 + b_3).$$

For our purpose, X_α's are indicator random variables that tell us if the sum of adjacent elementary volumes associated with the index α takes a particular value u or not. These events usually happen with small probability so it is reasonable to use Poisson approximation.

Let $\mathcal{V} = (V_1, \ldots, V_N)$ be the distribution of elementary volumes and b be the maximum window size. We define the index set as $\mathcal{I} = \{(i,j) \mid \ j - i + 1 \leq b\}$. We define the neighbourhood set of index (i,j) as $\beta_{(i,j)} = \{(k,l) \mid i < k < j, l > j, l - k + 1 \leq b\} \cup \{(k,l) \mid k < i, i < l < j, l - k + 1 \leq b\}$. We define $X_{(i,j)}(u) = \mathbb{1}\left\{\sum_{k=i}^j V_k = u\right\}$ to be the indicator random variables to check if the sum of i-th elementary volume through j-th elementary volume is equal to some volume u. Then $U(u) = \sum_{\alpha \in \mathcal{I}} X_\alpha(u)$ is the random variable for the number of times we see u in the set of observed volumes. Using Theorem 5.4, we find an upper bound on the probability that we do see u. The result is summarized in the following lemma. See the full version for the proof.

Lemma 5.1 (Distribution of Observed Volumes)**.** *Let $\mathcal{V} = (V_1, \ldots, V_N)$ be the distribution of elementary volumes and b be the maximum window size. Let $\mathcal{I}$ be the index set, $\beta_{(i,j)}$ be the neighbourhood set, and $X_{(i,j)}(u)$ be the indicator random variables defined above. Let $U(u) = \sum_{\alpha \in \mathcal{I}} X_\alpha(u)$. Then*

$$\Pr[U(u) > 0] \leq 1 - e^{-\lambda(u)} + \min(1, \lambda^{-1})(b_1 + b_2),$$

where

$$\begin{aligned}
\lambda(u) &= \sum_{\substack{(i,j) \\ j-i+1 \leq b}} \Pr\left(\sum_{k=i}^{j} V_k = u\right), \\
b_1(u) &= \sum_{\substack{(i,j) \\ j-i+1 \leq b}} p_{(i,j)}(u)^2 \\
&\quad + \sum_{\substack{(i,j) \\ j-i+1 \leq b}} \sum_{\substack{(k,l) \\ i<k<j, l>j \\ l-k+1 \leq b}} p_{(i,j)}(u) p_{(k,l)}(u) \\
&\quad + \sum_{\substack{(i,j) \\ j-i+1 \leq b}} \sum_{\substack{(k,l) \\ k<i, i<l<j \\ l-k+1 \leq b}} p_{(i,j)}(u) p_{(k,l)}(u), \\
b_2(u) &= \sum_{\substack{(i,j) \\ j-i+1 \leq b}} \sum_{\substack{(k,l) \in \beta_{(i,j)} \\ (k,l) \neq (i,j)}} \sum_{x} \Pr\left(\sum_{m=i}^{k-1} V_m = x\right) \cdot \\
&\quad \Pr\left(\sum_{m=k}^{j} V_m = u - x\right) \cdot \Pr\left[\sum_{m=j+1}^{l} V_m = x\right].
\end{aligned}$$

Distribution of out-of-order sum of observed volumes. An out-of-order sum of the observed volumes is a summation of volumes picked from the set of observed volumes in random order. So the sum does not necessarily correspond to a sum of adjacent elementary volumes. We need the distribution of out-of-order sum of the observed volumes because it is key to derive the probability that a random sum of the observed volumes is equal to one of the observed volumes.

To derive the distribution, we do not find the Chen-Stein method as useful as the neighbourhood set is always everything, and b_1 and b_2 are too large to provide any meaningful bound. Instead, we simply use union bound for this purpose. The results are summarized in Lemma 5.2. The proof is presented in the full version of the paper. As of notation, we write U_i for the distribution of the summation of i volumes from the set of observed volumes, where the database is distributed with $\mathcal{V} = (V_1, \ldots, V_N)$. By abusing notation, we write U to mean the distribution of the observed volumes.

Lemma 5.2 (Distribution of out-of-order sum of observed volumes)**.** *Let $\mathcal{V} = (V_1, \ldots, V_N)$ be the distribution of elementary volumes and b be the maximum window size. Write U for the distribution of observed volumes and U_k for the distribution of summations of k volumes from the set of observed volumes. We have*

$$\Pr[u \in U_k] \approx \Pr[U * \ldots * U = u] \ / \ i!,$$

where $U * \ldots * U$ is the convolution of U k times.

COLLISION PROBABILITY AND UNIQUENESS OF SOLUTION. Suppose we have more than one solution by the end of $\mathcal{A}_1$. This means that there is at least one non-genuine solution satisfying all the constraints. The solution has to be different from the original database by at least one volume. With some simplification, we assume that this volume is in the middle of the tuple, then the solution must satisfy two two-way out-of-order sum of observed volumes, three three-way out-of-order sum of observed volumes and so on. In other words, there are two volumes drawn from U_2 that are in U by chance, and so on.

We define k-way collision probability by

$$\Pr[k\text{-way collision}] = \max_x \left[\Pr[U(x) > 0] \Pr[x \in U_k]\right].$$

This is the maximum probability that a random sum of k out-of-order observed volumes equal to one of the observed volumes. This allows us to derive the probability that the solutions of length b contains only genuine solutions in Lemma 5.3. The proof can be found in the full version of the paper.

Lemma 5.3. *Let $\mathcal{V} = (V_1, \ldots, V_N)$ be the distribution of elementary volumes and b be the maximum window size. Let S_b be the solutions of length b. Then S_b contains only genuine solutions with probability approximately*

$$1 - \sum_{j=1}^{i-1} \binom{i-1}{j} |W|^j \prod_{k=2}^{i} \Pr[k\text{-way collision}]^{\min\{j\cdot k, i-k+1\}}.$$

The next Theorem establishes an estimation for p_{unique}.

Theorem 5.5 (Estimation of p_{unique})**.** *Let $\mathcal{V} = (V_1, \ldots, V_N)$ be the distribution of elementary volumes and b be the maximum window size. We can estimate p_{unique} as*

$$\Pr[S_b \textit{ contains only genuine solutions}] - (N-b)q,$$

where $q = 2|W| \prod_{k=2}^{b+1} \Pr[k\text{-way collision}]$.

Proof. We know from Lemma 5.3 there is a certain probability that the solutions of length b contains only genuine solutions. We seek the probability that the solution set remains genuine for all further iterations. There are $|W|$ to be tested on the left and right of the genuine solutions in each iteration. If our algorithm finds a solution by chance, there must be a k-way collision for all $2 \le k \le b$. Hence, using union bound, we get

$$\begin{aligned}
&\Pr[{}^{S_{j+1} \text{ contains}}_{\text{only genuine solutions}} \mid {}^{S_j \text{ contains}}_{\text{only genuine solutions}}] \\
\approx &1 - 2|W| \prod_{k=2}^{b} \Pr[k\text{-way collision}] \\
= &1 - q.
\end{aligned}$$

Applying union bound again, we find

$$\begin{aligned}
&p_{\text{unique}} \\
\approx &\Pr[S_b \text{ contains only genuine solutions}] \\
&- \sum_{j=b+1}^{N} 1 - \Pr[{}^{S_{j+1} \text{ contains}}_{\text{only genuine solutions}} \mid {}^{S_j \text{ contains}}_{\text{only genuine solutions}}] \\
= &\Pr[S_b \text{ contains only genuine solutions}] - (N-b)q.
\end{aligned}$$

□

5.3.1.4 Complexity Analysis

We find worst-case complexity analysis uninformative, as there are cases where the solution set is exponentially large throughout the execution of the algorithm. Furthermore, the worst case scenarios would impact even an average-case analysis. Instead, we concentrate on bounding the probability that the number of partial solutions maintained does not exceed a certain size. In turn, this bound implies a bound on the total runtime of the attack.

It turns out that we can reuse the analysis of uniqueness discussed above. We are interested in the same event as our analysis of uniqueness, that is, we bound the probability that the set of partial solutions of length b to N contain only genuine solutions. As there are at most $2(N-i+1)$ genuine solutions of length i, we know that if the event happens, the number of solutions of length b to N cannot exceed $2N$. On the other hand, the number of solutions before iteration b cannot exceed $|W|^{b-1}$. Therefore, we conclude that the number of partial solutions never exceed $|W|^{b-1}$ with probability p_{unique}.

As a consequence, the space complexity of the attack is $\mathcal{O}(|W|^{b-1})$ with probability p_{unique}. In each iteration of the attack, every partial solution is appended with all volumes in W to the left and to the right, and at most b conditions are checked. Therefore, extending a partial solution takes $\mathcal{O}(2b\,|W|)$ operations, and the total time complexity is $\max_i \mathcal{O}(bN\,|S_i|) = \mathcal{O}(bN\,|W|^b)$ with probability p_{unique}.

5.3.1.5 Experimental Data

In order to validate our basic attack and the other attacks we will present in this chapter, we chose to use the Healthcare Cost and Utilization Project (HCUP) dataset as the attack target. The HCUP dataset is maintained and published by the Agency for Healthcare Research and Quality (AHRQ) who is the lead Federal agency in US with a mission to improve the safety and quality of America's healthcare system. One of the datasets in the HCUP is the National (nationwide) Inpatient Sample (NIS), which is the largest publicly available all-payer inpatient healthcare database in US. It contains data from over 7 million hospital stays each year.

The NIS has been anonymised to protect patient privacy. **We did not attempt to deanonymise any of the data, nor are our attacks designed to deanonymise medical data.** I and the other authors involved in the original paper have completed the HCUP Data Use Agreement training and submitted signed Data Use Agreements to the HCUP Central Distributor.

We use the HCUP dataset from year 2004, 2008 and 2009 for our experiments. For each year the dataset contains data for about one thousand hospitals. The median number of inpatients is around 3 to 4 thousand depending on the year. Most of the data fields are for discharge information and hence, not suitable for our attacks. We choose admission month (`AMONTH`), number of diagnoses (`NDX`), number of procedures (`NPR`), age (`AGE`), and length of stay (`LOS`) as the target attributes for our attacks as these attributes are suitable for range queries and they have diversified distributions. For the attributes `NDX` and `NPR`, there are 16 categories for year 2004 and 2008 and these attributes are labelled

as `NDX-16` and `NPR-16` respectively. For the same attributes, there are 26 categories for year 2009 and these attributes are labeled as `NDX-26` and `NPR-26` respectively. Summary statistics of the attributes can be found in Table 5.2. We also offer a comparison of the number of queries required to observe all volumes in the leakage for the GLMP18 attack and our basic attack with a typical window size, assuming an uniform query distribution.

Attribute	Max. N	Avg. N	Dense	b	GLMP	Basic
AMONTH	12	11.9	97.8%	3	451.2	149.1
NDX-16	16	12.0	4.3%	3	598.4	175.0
NDX-26	26	17.2	8.0%	4	1221.3	341.6
NPR-16	16	9.2	15.3%	3	343.2	113.2
NPR-26	26	10.7	1.0%	4	463.8	168.7
AGE	83	71.1	39.2%	6	16666.3	1714.5
LOS	366	45.9	0.0%	6	6671.6	805.9

Table 5.2: Summary statistics on the attributes used in the attacks.

5.3.1.6 Experimental Validation

We present an experimental evaluation of our basic attack. We perform attacks on the attributes `AMONTH`, `NDX`, `NPR` and `AGE` as these attributes have a variety of N ranges and different distributions, allowing us to understand the effectiveness of our attack on different databases. We excluded experimental results on the attribute `LOS`, as most of the databases on the attribute cannot be recovered uniquely with a relatively small b. We discuss how to relax the adversarial goal to recover a part of the database uniquely in Section 5.3.4. We also give numerical examples and simulations to the analysis of uniqueness of solutions presented in Section 5.3.1.3.

Experimental results. For efficiency of implementation, we abort as soon as the size of the solution set exceeds $|W|^2$. Although our earlier analysis suggests that the solution size can grow exponentially, our experiments show a small threshold is sufficient for most attacks. We test our attack on the attributes and parameters shown in Table 5.3. For each attribute in turn, we report on the fraction of databases we recover uniquely and the fraction of databases for which more than one solution exist. We also indicate the fraction of databases for which our clique-finding initialization procedure fails and the the fraction of databases where our attack runs out of space.

For the attributes with moderate N, our attack works sufficiently well with bounds as small as 3 to 5. Our attack is less effective on the attribute `AGE` as the databases on the attribute often contain segments of small volumes that cannot be recovered uniquely given the bounds. Nonetheless, even here our attack recovers uniquely (up to reflection) one in five databases.

Theoretical analysis of uniqueness of solutions and simulation. We compare the quality of the theoretical bound on uniqueness with experimental simulation, using the theoretical bound described in Section 5.3.1.3. We perform experiments on distributions of databases where the elementary volumes are modelled by independent and identically distributed binomial distributions, for $b = 5$ to 8, and $N = 40$. There

Attribute	b	Unique	Ambiguous	Clique fail	Abort	Avg. time (s)
AMONTH	3	78.5%	3.2%	0.0%	18.3%	0.0087
NDX-16	3	87.3%	2.1%	0.0%	10.7%	0.0083
NDX-26	4	82.7%	4.1%	0.3%	12.9%	0.0494
NDX-26	5	88.3%	40.0%	1.2%	6.4%	0.0498
NPR-16	3	89.1%	10.2%	0.0%	0.7%	0.0011
NPR-26	4	81.2%	15.2%	0.0%	3.7%	0.0124
NPR-26	5	84.6%	14.2%	0.0%	1.2%	0.0131
AGE	6	22.4%	0.9%	41.1%	35.7%	17.60
AGE	8	32.1%	0.6%	52.6%	14.7%	15.99

Table 5.3: Experimental data for the basic attack.

are two types of synthetic distributions we have considered, namely databases with an increasing/decreasing elementary volumes and those with an inverted-U shape.

For the first type of database, we sample the elementary volumes as $\mathbf{DB}_i \sim \mathbf{Binom}(20i + 200, 0.5)$. For the second type of database, we sample the elementary volumes as $\mathbf{DB}_i \sim \mathbf{Binom}(400 - 20i, 0.5)$ for $N \leq 20$ and $\mathbf{DB}_i \sim \mathbf{Binom}(20i - 200, 0.5)$ for $N > 20$. We compute the theoretical estimations of the uniqueness rates for the two types of databases and compare those to the simulations.

For the simulations, we generate 1000 databases from the type of database in interest and execute our basic attack. We report the uniqueness rates with out the final leakage check procedure (line 7 of Algorithm 5.5) so that the comparison with our theoretical estimations is more direct. The experimental results are shown in Table 5.4 respectively.

Our theoretical analysis of the uniqueness of solutions can be overly pessimistic with small b, but it is fairly accurate for larger b. Our experiments provide some initial evidence that databases with large variations of elementary volumes are susceptible to volume leakage attacks.

b	Experimental	Theoretical
5	0.0%	98.1%
6	73.8%	98.1%
7	95.8%	99.8%
8	99.5%	99.7%

b	Experimental	Theoretical
5	0.0%	85.9%
6	0.0%	98.7%
7	40.4%	99.7%
8	88.0%	99.6%

Table 5.4: Experiments on the databases with a decreasing shape (left) and inverted-U shape (right).

5.3.2 Simple Variations on the Leakage

In this section, we study the robustness of our basic attack with variations on the basic leakage function in Equation (5.1). These variations correspond to more realistic query distributions, the use of countermeasures, or both. For example, we look at the case where the user may issue some queries with window size larger than b, or decide not to query some of the small ranges. Additionally, the server can pad some fake records to the query responses to invalidate our basic attack. Recall that our basic attack incrementally extends solutions using volumes which need to verify some constraints so additional volumes, or missing ones, will impact our algorithm. Nonetheless, we show that many of the databases can still be reconstructed uniquely under these variations.

5.3.2.1 Spurious Volumes

First, we consider two types of spurious volumes in our attack, namely volumes from queries with larger windows and random volumes. We show that with slight modification to our basic attack, a significant proportion of the databases from the HCUP dataset can still be recovered uniquely.

Attack Overview. To model additional volumes we use some distribution $\mathcal{N}$. We write $n \leftarrow \mathcal{N}(\mathbf{DB})$ for sampling a volume from this distribution. Notice that we allow the distribution to depend on the real database counts: this is useful to model both the case when noise consists of real volumes (due to real queries outside of unexpected window size) and noise which is calibrated with respect to the real data. The leakage function in the presence of such noise is

$$\mathcal{L}_{b,\mathcal{N},I}(\mathbf{DB}) = \mathcal{L}_b(\mathbf{DB}) \cup \{n_i \mid i \leq I, n_i \leftarrow \mathcal{N}(\mathbf{DB})\}. \tag{5.2}$$

for some I which itself may depend on $\mathbf{DB}$. The goal of the adversary is to reconstruct database $\mathbf{DB}$ given a sample from $\mathcal{L}_{b,\mathcal{N}}(\mathbf{DB})$.

For this type of scenarios, we argue that the same strategy (with a minimal change) works. That is, we use the same way to incrementally build solutions by checking the same constraints as before. While all solutions can still be found (if the attack does not abort) the presence of noise may lead to additional solutions which include fake volumes. Worse, since the adversary learns $W \leftarrow \mathcal{L}_{b,\mathcal{N}}(\mathbf{DB})$ we cannot sift potential solutions by checking that $\mathcal{L}_b(\mathbf{DB}) = W$. We experimentally confirm that the success of the adversary does not drastically degrade.

We do need to change the way we initialize our attack: due to noise our initial solution finding algorithm may fail to return a genuine partial solution. For example, if the maximum observed volume is a fake volume, our clique-finding algorithm will certainly no generate proper partial solutions. To overcome this problem, we run the initial solution finding algorithm, iteratively, starting with different volumes and use the union of all initial solutions found this way as the starting point for the iterative part of the attack.

Our attack in the presence of noise is the same as Algorithm 5.5, except that we omit the final check on the leakage (line 7 of Algorithm 5.5) and employ a slightly more elaborate algorithm for identifying starting partial solutions. We refer to the resulting attack as $\mathcal{A}_2$.

Theoretical Analysis. Since we can no longer check for equality of leakages in $\mathcal{A}_2$, we cannot ensure that any solution $\mathbf{DB}'$ output by the attack satisfies $\mathcal{L}_b(\mathbf{DB}') = W$, though it must be the case that $\mathcal{L}_b(\mathbf{DB}') \subseteq W$. The modified correctness property is established by the following theorem. The proof is identical to that of the basic attack so it is omitted.

Theorem 5.6 (Correctness of the attack in the presence of noise)**.** *Let $\boldsymbol{DB}$ be a database, $N = |\boldsymbol{DB}|$ and b be any natural number less or equal to N. For any possible sample from the leakage function $W \leftarrow \mathcal{L}_{b,\mathcal{N}}(\boldsymbol{DB})$, let S_N be the output of $\mathcal{A}_2$, i.e. $S_N = \mathcal{A}_2(W, b, N)$. Then*

1. $\forall \boldsymbol{DB}' \in W^N, \mathcal{L}_b(\boldsymbol{DB}') \subseteq W \Rightarrow \boldsymbol{DB}' \in S_N$,

2. $\boldsymbol{DB} \in S_N$.

Experimental Results. We use `NDX-26` and `NPR-26` as the attributes to perform experiments: since our basic attack succeeded with high probability for these attributes, they offer a good starting point to understand the effects of the noise.

For uniformly distributed random noise, the fake volumes are drawn without replacement from a discrete uniform distribution with lower limit as the minimum observed volume, and upper limit as the maximum observed volume. The minimum observed volume is used as the initial solution to the attacks. We compare recovery rate of our attack with $b = 5$ and noise levels 0.5 and 1, where with noise level α, $\alpha \cdot |\mathcal{L}_b(\mathbf{DB})|$ fake volumes are added to $\mathcal{L}_b(\mathbf{DB})$. We abort an attack as soon as the solution set is larger than $|W|^2$. The results are summarized in Table 5.5.

Attribute	α	Unique	Ambiguous	Abort	Avg. Time (s)
NDX-26	0.5	62.3%	15.9%	21.9%	0.72
NDX-26	1	58.1%	16.4%	25.5%	1.70
NPR-26	0.5	61.4%	16.7%	22.0%	0.61
NPR-26	1	60.1%	18.1%	21.7%	1.37

Table 5.5: Experimental data for the attack with uniform random noise.

For volumes from larger windows, we test our attack on $b = 5$ and larger window of size 8. The minimum observed volume is used as the initial solution. We use noise levels $\alpha = 0.5$ and 0.75 in our experiments, where with noise level α, $\alpha \cdot |\mathcal{L}_8(\mathbf{DB}) \backslash \mathcal{L}_5(\mathbf{DB})|$ volumes from larger windows are added to $\mathcal{L}_5(\mathbf{DB})$. We abort an attack as soon as the solution set is larger than $|W|^2$. The results are summarized in Table 5.6.

Attribute	α	Unique	Ambiguous	Abort	Avg. Time (s)
NDX-26	0.5	64.5%	14.3%	21.2%	0.46
NDX-26	0.75	60.4%	16.1%	23.5%	1.00
NPR-26	0.5	43.6%	20.1%	36.4%	0.45
NPR-26	0.75	41.8%	20.9%	37.3%	0.61

Table 5.6: Experimental data for the attack with noise from larger windows.

5.3.2.2 Missing Queries

In our basic attack, every volume within a given window size is assumed to be observed by the adversary, so all the constraints within the window size b can be checked. In practice, it is possible that some of the queries are not issued by the user as they are uninteresting. It is also possible that the user/server actively blocks some of the queries in an attempt to defend against volume leakage attacks. We demonstrate that it is still possible to reconstruct the database uniquely, even if some of the volumes from the small windows are missing. For simplicity, within each window of size b, we assume the adversary does not have access to k randomly chosen volumes. Furthermore, we assume that the elementary volumes and all two-way sums of the elementary volumes are always part of the leakage. Our assumptions are somewhat arbitrary but our attack demonstrate that there is a lot of redundancy in the leakage function to cope with missing volumes.

Attack Overview. We start with a description of the leakage function one may observe when some queries are suppressed. Let k be a natural number that is less or equal to $b-2$, let $\mathcal{I}$ be an index set of pairs of values as follows:

1. $\forall i \leq N, (i,i) \in \mathcal{I}$,
2. $\forall i \leq N-1, (i,i+1) \in \mathcal{I}$,
3. $\forall i \leq N-b, |\{(x,y) \in \mathcal{I} \mid i \leq x \leq y \leq i+b-1\}| \geq \frac{b(b+1)}{2} - k$.

Indexes (x,y) that appear in $\mathcal{I}$ are the queries the adversary observes. Notice that condition (3) allows for a certain number of queries to be missing (more specifically k queries for each individual window of size b.

For a fixed $\mathcal{I}$, the adversary learns the leakage function

$$\mathcal{L}_{b,\mathcal{I}}(\mathbf{DB}) = \left\{ \sum_{i=x}^{y} v_i \mid (x,y) \in \mathcal{I}, y-x+1 \leq b \right\}. \tag{5.3}$$

The adversary is given the leakage of some database $\mathcal{L}_{b,\mathcal{I}}(\mathbf{DB})$ and k, and his goal is to reconstruct all $\mathbf{DB}$ that generates the set of observed volumes, potentially with a different index set that satisfies all the constraints.

We can no longer check all the additive constraints within the windows, but given our assumption on $\mathcal{I}$, we know that a solution is not plausible if in any window, there are more than k missing constraints. Therefore, we extend a solution by some new volume only if at most k of the constraints associated to the new volume are missing. Our attack is described by Algorithm 5.6, and we call the attack $\mathcal{A}_3$.

Algorithm 5.6 Attack with missing queries

input $W = \{\sum_{i=x}^{y} \mathbf{DB}_i \mid (x,y) \in \mathcal{I}, y-x+1 \leq b\}, b, k, N$
output $\{(w_1, \ldots, w_N) \mid w_i \in W\}$

procedure Attack(W, b, k, N)
    $S_1 = \{(\min(W))\}$
    for $i \leftarrow 2, N$ **do**
        $S_i \leftarrow$ ExtendLeft$(S_{i-1}, W, b, k) \cup$ ExtendRight(S_{i-1}, W, b, k)
    return S_N

procedure Extend Left(S_i, W, b, k)
    $S' \leftarrow \{\}$
    for all $(w_1, \ldots, w_i) \in S_i$ **do**
        for all $w_0 \in W$ **do**
            $m \leftarrow 0$
            for all $(x,y), x < y, y \leq b-1$ **do**
                if $\left(\sum_{j=x}^{y} w_j\right) \notin W$ **then**
                $m \leftarrow m+1$
            if $m \leq k$ **then**
                $\mathcal{S}' \leftarrow S' \cup \{(w_0, w_1, \ldots, w_i)\}$
    return S'

Theoretical Analysis. We say that $\mathcal{A}_3$ is correct if for all solutions found by the algorithm, there exists an index set satisfying all the constraints and the resultant leakage is the same as the leakage from the input. The following theorem establishes that $\mathcal{A}_3$ is correct. The proof is similar to that of the basic attack with the invariant changed to $\mathcal{L}_{b,\mathcal{I}'}(s) \subseteq W$ for some index set $\mathcal{I}'$ and partial solution s, so it is omitted.

Theorem 5.7 (Correctness of the attack with missing queries)**.** Let **DB** be a database, $N = |\mathbf{DB}|$ and b be any natural number less or equal to N. Let $\mathcal{I}$ be an index set described above. Let $W = \mathcal{L}_{b,\mathcal{I}}(\mathbf{DB})$ and S_N be the output of $\mathcal{A}_3$, i.e. $S_N = \mathcal{A}_3(W, b, k, N)$. Then

1. $\forall \mathbf{DB}' \in W^N, \forall \mathcal{I}' \in ([N] \times [N])^*, \mathcal{L}_{b,\mathcal{I}'}(\mathbf{DB}') = W \Rightarrow \mathbf{DB}' \in S_N$,
2. $\mathbf{DB} \in S_N$.

Experimental Results. We test our attack on the attributes `NDX-26` and `NPR-26`, with the missing queries generated uniformly, and the results are shown in Table 5.7. A significant proportion of the databases can still be reconstructed uniquely in the setting $b = 6$ and $k = 2$, indicating that banning some of the queries is not an efficacious countermeasure.

Attribute	b	k	Unique	Ambiguous	Abort	Avg. Time (s)
NDX-26	5	1	42.8%	23.7%	33.4%	2.20
NDX-26	6	1	59.4%	16.7%	23.9%	21.31
NDX-26	6	2	40.1%	27.4%	32.5%	40.33
NPR-26	5	1	44.6%	20.5%	34.9%	1.18
NPR-26	6	1	56.1%	15.9%	27.9%	5.97
NPR-26	6	2	40.4%	24.0%	35.5%	11.87

Table 5.7: Experimental data for attack with missing queries.

5.3.2.3 Adding Fake Records

To a large extent, our attacks rely on checking equality of volumes, i.e. if v_1 and v_2 are adjacent volumes, then their sum should be in the set of observed volumes. An obvious defense strategy is to pad the responses with fake records. An entire spectrum of instantiations of this idea is possible. We discuss some considerations in padding strategies, and present an attack which bypasses a plausible instantiation of this countermeasure.

The best defense is to pad *all* answers with a large number of fake records. While there is clearly no effective database reconstruction attack, the scheme is highly inefficient. For efficiency one may choose to pad a small fraction of the queries with a small number of fake records. If the parameters of the padding strategy are chosen inappropriately, the attacker may be able to recover the true observed volume and use our basic attack to reconstruct the underlying database. Interestingly, the padding strategy plays an important role in the security too. For example, if the size of the padding is generated uniformly at random for each individual query then an attacker may be able to learn the true volumes from the perturbed volumes for some databases. On the other hand, if the size of the padding stays is an (unknown) constant r for all queries, then it is easy to see that the optimal guess of any elementary volume always has an uncertainty of r.

ATTACK OVERVIEW. Let $N = \{N_{x,y}\}$ be some noise distribution (which can potentially depend on the indices of the queries and the volumes of the query responses). We formally define the leakage function as

$$\mathcal{L}_{b,N}(\mathbf{DB}) = \left\{ \sum_{i=x}^{y} v_i + n_{x,y} \mid y - x + 1 \leq b, n_{x,y} \leftarrow N_{x,y} \right\}. \tag{5.4}$$

Given the leakage of some database $\mathcal{L}_{b,N}(\mathbf{DB})$ and a description of N, the adversary is asked to find all databases that can generate the set of observed volumes. We study the case where each query is padded with up to r fake entries selected uniformly and independently. That is, i.e. for all x, y, $N_{x,y} = \text{Uniform}(1, r)$.

Our attack $\mathcal{A}_4$ begins by guessing the ranges for the observed volumes. A guess is of the form (w_1, w_2) where w_1 is the lower bound and w_2 is the upper bound (inclusive) of the guess. A partial solution is a tuple just like that in the basic attack $\mathcal{A}_1$, except that the entries are ranges of the form (w_1, w_2) as described above. The solution extension procedure uses the same idea as the basic attack, but the constraints checked are changed to if the partial solution can generate the padded volumes. We present the pseudocode of the attack in Algorithm 5.7.

Algorithm 5.7 Attack with padded queries

input $R, W = \{\sum_{i=x}^{y} \mathbf{DB}_i + r \mid y - x + 1 \leq b, r \leftarrow N\}, b, N, r = \max(N)$
output $\{(w_1, w_2), \ldots, (w_{2N-1}, w_{2N}))\}$

procedure ATTACK(R, W, b, N, r)
    $S_1 = \{(\min(R))\}$
    for $i \leftarrow 2, N$ **do**
        $S_i \leftarrow$ EXTENDLEFT(R, S_{i-1}, W, b, r)$\cup$ EXTENDRIGHT(R, S_{i-1}, W, b, r)
    for all $s \in S_N$ **do**
        if $\exists w \in W, \forall (x, y), \sum_{i=x}^{y} s[i][1] < w - r$ and $\sum_{i=x}^{y} s[i][2] > w - 1$ **then**
            $S_N \leftarrow S_N \setminus \{s\}$
    return S_N

procedure EXTEND LEFT(S_i, R, W, b, r)
    $S' \leftarrow \{\}$
    for all $(w_1, \ldots, w_i) \in S_i$ **do**
        for all $w_0 \in R$ **do**
            if $\forall j < b, \exists w \in W, \sum_{k=0}^{j} w_k[1] \geq w - r$ and $\sum k = 0^j w_k[2] \leq w - 1$ **then**
                $\mathcal{S}' \leftarrow S' \cup \{(w_0, w_1, \ldots, w_i)\}$
    return S'

We note that our notion of approximate reconstruction is different from that of the GLMP19 attack [88]. In their notion, the adversary is given the access pattern leakage of *all* queries and his goal is to guess the value of every record within a certain threshold of error. For our attack, the approximation is on the elementary volumes, and this is the best the adversary can do as the volumes are perturbed.

As the goal of the adversary is to approximately recover the database, he can give up some accuracy in his guess to allow for more databases to be uniquely reconstructed. By that, we mean that if there are two solutions $((w_1, w_2), (w_3, w_4))$ and $((w_1, w_5), (w_3, w_4))$ and $w_5 > w_2$, the adversary can merge the guesses as $((w_1, w_5), (w_3, w_4))$ at a loss of accuracy. In our attack, we achieve this trade-off by allowing relaxed guesses on the observed volumes.

Theoretical Analysis. The solutions in the final solution set S_N are of the form $((w_1, w_2), \ldots, (w_{2N-1}, w_{2N}))$. We say that the attack is correct if given any solution of that form, there is a database $(\mathbf{DB}_1, \ldots, \mathbf{DB}_N)$ that can generate the given leakage W and the solution contains the database, i.e. $w_{2i-1} \leq \mathbf{DB}_i \leq w_{2i}$ for all $i = 1, \ldots, N$. We formalize the correctness of $\mathcal{A}_4$ as follows. The proof is similar to that of the basic attack except that the invariant is changed to that there is a realisation of $\mathcal{L}_{b,N}(s)$ that generates a subset of W. The proof is omitted due to space limitation.

Theorem 5.8 (Correctness of attack with padded queries)**.** Let $\mathbf{DB}$ be a database, $N = |\mathbf{DB}|$ and b be any natural number less or equal to N. Let $N = \{N_{x,y}\}$ be distributions of noises with $N_{x,y} = \text{Uniform}(1, r)$ for some natural number r. Let $W \leftarrow \mathcal{L}_{b,N}(\mathbf{DB})$, R be some estimations of the true volumes and S_N be the output of $\mathcal{A}_4$, i.e. $S_N = \mathcal{A}_4(R, W, b, N, r)$. Then

1. $\forall \mathbf{DB}' \in \mathbb{N}^N, W \in \text{supp}(\mathcal{L}_{b,N}(\mathbf{DB}')) \Rightarrow \exists s \in S_N, \forall s_i, s_i[1] \leq \mathbf{DB}'[i] \leq s_i[2]$,
2. $\mathbf{DB} \in S_N$.

Experimental Results. We study the effectiveness of our attack with two sets of attacks on the attributes `NDX` and `NPR` with $b = 5$ and $r = 10$. Our choice of r is arguably small but that is due to the fact that the elementary volumes themselves are small. For instance, for the attribute `NPR-26`, over 54% of the elementary volumes are below 100. As before, we abort if the number of partial solutions is over $|W|^2$.

For the first set of attacks, we aim to reconstruct all elementary volumes with the best possible precision. To do that, we compute R as $R = \{(v - r, v - 1) \mid v \in W\}$ and execute our attack (Algorithm 5.7). The experimental results are shown in Table 5.8.

Attribute	Unique	Ambiguous	Abort	Avg. Time (s)
NDX-16	4.5%	16.2%	79.3%	0.92
NPR-16	10.0%	13.1%	76.9%	0.64
NDX-26	1.4%	6.0%	92.6%	2.82
NPR-26	6.7%	8.4%	84.9%	1.25

Table 5.8: Experimental data for the attack with perturbed volumes, the unique solutions are the most information-theoretically precise solutions.

For the second set of attacks, we trade precision for more unique solutions. We compute R just as before, and iteratively merge ranges $(w_1, w_2), (w_3, w_4)$ in R into (w_1, w_4) if $w_2 \geq w_3$. The experimental results are shown in Table 5.9.

An overwhelming proportion of the attacks abort as the databases often contain similar elementary volumes. After perturbing, these volumes can often be swapped without violating the constraints. Overall, adding fake records is a better countermeasure than the other ones we have considered.

Attribute	Unique	Ambiguous	Abort	Avg. Time (s)
NDX-16	19.1%	10.8%	70.1%	0.84
NPR-16	19.7%	9.8%	70.5%	0.50
NDX-26	11.1%	6.7%	82.1%	2.10
NPR-26	12.1%	9.2%	78.7%	0.74

Table 5.9: Experimental data for the attack with perturbed volumes, the guesses on the elementary volumes are relaxed.

5.3.3 Attack on Observed Volumes with Bounded Window Size

All attacks in the literature and our attacks described above require the set of elementary volumes to be part of the observed volumes, so one may suspect that these are absolutely necessary. In this section, we show that this is not the case. We construct a successful adversary which only observes volumes for queries of medium window sizes, i.e. there is a lower bound and an upper bound on the window size. More formally, we define the leakage function as:

$$\mathcal{L}_{a,b}(\mathbf{DB}) = \left\{ \sum_{i=x}^{y} v_i \mid \ a \leq y - x + 1 \leq b \right\} \tag{5.5}$$

for some $0 < a < b \leq N$. For simplicity, we assume b and N are multiples of a. The assumptions are not necessary requirements for our attack to work, though they make our attack simpler to present and understand. We do need however that b is somewhat large compared with a, more specifically that $b > k \cdot a$, for some k. We explain below the role played by this restriction.

5.3.3.1 Reconstruction Algorithm

Intuition. We provide an overview of the ideas that go into our attack. For concreteness, let's assume that $a = 3, b = 9, N = 12$. The database counts are $\mathbf{DB} = (v_1, \ldots, v_{12})$. Here, and throughout this section, we write $\mathbf{DB}[i]$ for v_i, and we write $\mathbf{DB}[i, j]$ for $\sum_{k=i}^{j} v_k$. Clearly, $\mathbf{DB}[i, j] + \mathbf{DB}[j + 1, k] = \mathbf{DB}[i, k]$.

Our attack proceeds in two stages. In the first stage we recover the sequence $\tilde{\mathbf{DB}}_0 = (\mathbf{DB}[1, 3], \mathbf{DB}[4, 6], \mathbf{DB}[7, 9], \mathbf{DB}[10, 12])$ (i.e. disjoint queries with window size a which cover the entire database). To piece together $\tilde{\mathbf{DB}}_0$ we observe that the leaked information allows us to check constraints on neighbouring entries in $\tilde{\mathbf{DB}}$. For example, we have that $\mathbf{DB}[4, 6] + \mathbf{DB}[7, 9] = \mathbf{DB}[4, 9]$ and $\mathbf{DB}[4, 9]$ occurs in the leakage (it corresponds to a query with window size 6). Similarly, $\mathbf{DB}[4, 6] + \mathbf{DB}[7, 9] + \mathbf{DB}[10, 12] = \mathbf{DB}[4, 12]$ also occurs in the leakage (query with window size 9).

Based on the above observations, we construct $\tilde{\mathbf{DB}}_0$ by viewing its entry as elementary volumes and applying our basic search strategy. Every entry needs to satisfy between $k - 1$ and $2k - 2$ constraints, where $k = b/a$[2]. Importantly, notice that from $\tilde{\mathbf{DB}}_0$ we can also recover all volumes of the form $\mathbf{DB}[1, 3i]$.

Next, we determine volumes of the form $\mathbf{DB}[1, 3i + 1]$ by reconstructing the sequence $\tilde{\mathbf{DB}}_1 = (\mathbf{DB}[1, 4], \mathbf{DB}[5, 7], \mathbf{DB}[8, 12])$. One can think of $\tilde{\mathbf{DB}}_1$ as a variant of $\tilde{\mathbf{DB}}_0$

[2] k is the "window size" for $\tilde{\mathbf{DB}}_0$ – the larger the ration of b to a the more constraints we have available

shifted by 1. The first query has window size $a+1$, and all but last of the subsequent ones have window size a. The last query has size $2a-1$. Here, again we use the strategy to incrementally build solutions from shorter ones. In addition to the additive constraints which have to be satisfied by neighbouring entries (e.g. that $\mathbf{DB}[1,4]+\mathbf{DB}[5,7]$ occurs in the leakage) we also use two other types of constraints. Taking $w=\mathbf{DB}[1,4]$ as an example, notice that it must be the case that $v[1,3] \leq w \leq v[1,6]$. It also must be the case that the values $\mathbf{DB}[1,9]-w=\mathbf{DB}[5,9]$ and $\mathbf{DB}[1,12]-w=\mathbf{DB}[5,12]$ occur in the leakage (since these correspond to queries of window size 5 and 8 respectively whereas the maximum size of a query window size is 9). Similar conditions on size and relation to the entries in the leakage hold for the rest of the entries in $\tilde{\mathbf{DB}}_1$. Figure 5.5 depicts the key ideas in the attack.

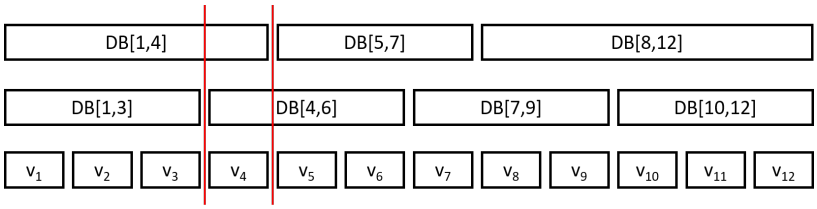


Figure 5.5: The bottom row of the figure represents the elementary volumes in the database. The row on top represents a solution for $\tilde{\mathbf{DB}}_0$. By finding the correct solution $\tilde{\mathbf{DB}}_1$ which 'shifts' all volumes by one position, we can recover all volumes of the shape v_{3i+1} as indicated by the red lines.

Finally, we also recover $\mathbf{DB}[1,3i+2]$ by reconstructing the sequence $\tilde{\mathbf{DB}}_2 = (\mathbf{DB}[1,5], \mathbf{DB}[6,8], \mathbf{DB}[9,12])$ using a similar strategy.

At this point, since we have recovered all volumes of the form $\mathbf{DB}[1,3i]$, $\mathbf{DB}[1,3i+1]$ and $\mathbf{DB}[1,3i+2]$ we can recover almost all entries in $\mathbf{DB}$ since $\mathbf{DB}[t+1]=\mathbf{DB}[1,t+1]-\mathbf{DB}[1,t]$. In our example, we can recover the values $(\mathbf{DB}[4],\mathbf{DB}[5],\ldots,\mathbf{DB}[9])$.

To complete the attack we recover the elementary volumes in the windows of size 3 at the start and end of $\mathbf{DB}$. This can be done by simply extending $(\mathbf{DB}[4],\mathbf{DB}[5],\ldots,\mathbf{DB}[9])$ to the left and right: since all volumes with window size between 3 and 9 are part of the leakage, we can check 6 constraints on $\mathbf{DB}[3]$ and $\mathbf{DB}[10]$, and so on.

One difficulty which in the above description is not apparent, is that when reconstructing $\tilde{\mathbf{DB}}_0, \tilde{\mathbf{DB}}_1, \tilde{\mathbf{DB}}_2$ we may obtain more than one solution for each. In brief, we overcome this difficulty by relying on constraints that need to hold between the entries in the three different sequences.

Attack overview. Our attack begins by finding all plausible solutions which consists of consecutive volumes corresponding to queries of window size a. That is, we find all valid $\tilde{\mathbf{DB}}_0$ from the example above. Call this set S_0. Next, for every solution in S_0, we find sequences of consecutive volumes: the first one belonging to the interval $\mathbf{DB}[1,a]$ and subsequent ones being elementary volumes – this step subsumes roughly identifying $\tilde{\mathbf{DB}}_1, \tilde{\mathbf{DB}}_2, \ldots$ and (most) of the elementary volumes in $\mathbf{DB}$. The result is a set S' of sequences where the first and last entry correspond to compound entries (i.e .queries of window size greater than 1) whereas all other entries are actual elementary volumes. Finally, for each sequence in S' we recover the elementary volumes on the sides (and remove the compound entries). All solutions are added to a set of tentative solutions S. Finally, we sift through S and only keep those entries that generate the leakage observed. Full pseudocode of the attack can be found in Algorithm 5.8.

Algorithm 5.8 Attack with bounded window size

input $W = \{\sum_{i=x}^{y} v_i \mid a \leq y - x + 1 \leq b\}, a, b, N$
output $\{(w_1, \ldots, w_N) \mid w_i \in W\}$

procedure ATTACK(W, b, N)
  $X \leftarrow \emptyset$
  $S \leftarrow \mathcal{A}_2(W, b/a, N/a)$
  for all $s \in S$ **do**
    $S' \leftarrow$ OFFSET SOLUTIONS($s, W, b/a, N/a$)
    $S' \leftarrow$ MERGE SOLUTIONS(S', s, W, a, b, N)
    for all $s' \in S'$ **do**
      $s' \leftarrow$ FINALISE SOLUTION(s', W, a, b, N)
      $X \leftarrow X \cup \{s'\}$
  $X \leftarrow \{x \mid x \in X, \mathcal{L}_{a,b}(x) = W\}$
  return X

  procedure OFFSET SOLUTIONS(s, W, b, N)
    $S_1 \leftarrow \emptyset$
    for all $w \in W$ **do**
      if CHECK $1(w, s, W)$ **then**
        $S_1 \leftarrow S_1 \cup \{(w)\}$
    for all $i \leftarrow 2, N - 2$ **do**
      $S_i = \emptyset$
      for all $s' \in S_{i-1}, w \in W$ **do**
        if CHECK $2(w, s', s, W)$ **then**
          $S_i \leftarrow S_i \cup \{s' + (w)\}$
    $S_{N-1} = \emptyset$
    for all $s' \in S_{N-1}$ **do**
      if $\left\{\sum_i s[i] - \sum_{i=1}^{j} s'[i] \mid \ N - b \leq j \leq N - 2\right\} \subset W$ **then**
        $S_{N-1} \leftarrow S_{N-1} \cup \{s' + (\sum_i s[i] - \sum_i s'[i])\}$
    return S_{N-1}

5.3.3.2 Correctness

We say our algorithm is correct if it identifies the set of solutions such that every solution in the set generates the same set of observed volumes as the one given at the start of the attack. For convenience, we call our attack on observed volumes with bounded window size $\mathcal{A}_5$. We establish that the algorithm works as expected, under some further assumptions on the parameters a and b.

Theorem 5.9 (Correctness of the attack with bounded window sizes)**.** *Let $\boldsymbol{DB}$ be a database, $N = |\boldsymbol{DB}|$ and a, b be natural numbers less or equal to N with $b > 2a$. Let S be the output of the attack, i.e. $S = \mathcal{A}_5(\mathcal{L}_{a,b}(\boldsymbol{DB}), a, b, N)$. Then*

1. $\forall \boldsymbol{DB}' \in \mathcal{L}_{a,b}(\boldsymbol{DB})^N, \mathcal{L}_{a,b}(\boldsymbol{DB}') = \mathcal{L}_{a,b}(\boldsymbol{DB}) \Leftrightarrow \boldsymbol{DB}' \in S$,
2. $\boldsymbol{DB} \in S$.

PROOF OF THEOREM 5.9. We sketch the proof of the correctness of the attack. This

procedure MERGE SOLUTIONS(S, s, W, a, b, N)
  $S_0 \leftarrow \{()\}$
  for all $i \leftarrow 1, N/a - 2$ **do**
    $S_i \leftarrow \left\{ s' + \left(\sum_{j=1}^{i} s[j] \right) \mid s' \in S_{i-1} \right\}$
    $X \leftarrow \left\{ \sum_{j=1}^{i} s'[j] \mid s' \in S \right\}$
    for all $x \in X, s' \in S_i$ **do**
      if $\left\{ x + \sum_{j=a-1}^{b} s'[j] \right\} \subset W \wedge$
      $\left\{ \left(\sum_{j=1}^{k} s[j] \right) - x \mid i+2 \leq k \leq i + b/a + 1, k < |s| \right\} \subset W$ **then**
        $S_i \leftarrow S_i \cup \{s' + (x)\}$
    $S_i \leftarrow \{s' \mid s' \in S', |s'| = i \cdot a\}$
  $S' \leftarrow \{(s'[0], s'[1] - s'[0], \ldots, s'[N-a] - s'[N-a-1], \sum_i s[i] - s[N/a-1] - s'[N-a], s[N/a-1] - s[N/a]) \mid s' \in S_{N/a-2}\}$
  return S'

procedure FINALISE SOLUTION(s, W, a, b, N)
  if $s = ()$ **then**
    return $\emptyset$
  $S' = \{s[2 : |s| - 1]\}$
  for all $i \leftarrow 1, a$ **do**
    $S_{tmp} \leftarrow \emptyset$
    for all $x \in S', w \in W$ **do**
      if $\left\{ w + \sum_{j=a}^{k} x[j] \mid \ k = a, \ldots, b-1 \right\} \subset W$ **then**
        $S_{tmp} \leftarrow S_{tmp} \cup \left\{ \left(w - \sum_{j=1}^{a-1} x[j] \right) + x \right\}$
    $S' = S_{tmp}$
  $S' \leftarrow \{x^R \mid \ x \in S'\}$
  for all $i \leftarrow 1, a$ **do**
    $S_{tmp} \leftarrow \emptyset$
    for all $x \in S', w \in W$ **do**
      if $\left\{ w + \sum_{j=k}^{|x|-a+1} x[j] \mid \ |x| - b \leq k \leq |x| - a + 1 \right\} \subset W$ **then**
        $S_{tmp} \leftarrow S_{tmp} \cup \left\{ x + \left(w - \sum_{j=0}^{a-2} x[|x| - j] \right) \right\}$
    $S' = S_{tmp}$
  return S'

is done through a sequence of correctness proofs of the sub-routines. The correctness proofs are all simple inductions and they are omitted from the paper. At a high level, the correctness of the sub-routines states that if we begin with a database that has the same leakage profile as what is given, and run the sub-routines with appropriate inputs including some transformation(s) of the database, then the outputs contain some other transformation(s) of the database.

Lemma 5.4 (Correctness of the Initial Solutions)**.** *Let $\boldsymbol{DB}$ be a database, $N = |\boldsymbol{DB}|$ and a, b be natural numbers less or equal to N with $b > 2a$. Let $S = \mathcal{A}_2(\mathcal{L}_{a,b}(\boldsymbol{DB}), b/a, N/a)$. Let $s = (\boldsymbol{DB}[1, a], \boldsymbol{DB}[a+1, 2a], \ldots, \boldsymbol{DB}[N-a+1, N])$. Then*

$$(s \in S) \vee \left(s^R \in S\right).$$

Lemma 5.5 (Correctness of procedure Offset Solutions)**.** *Let $\boldsymbol{DB}$ be a database, $N =$*

$|\boldsymbol{DB}|$ and a, b be natural numbers less or equal to N with $b > 2a$. Let $s = (\boldsymbol{DB}[1, a], \boldsymbol{DB}[a+1, 2a], \ldots, \boldsymbol{DB}[N-a+1, N])$. Let $S =$ Offset Solutions$(s, \mathcal{L}_{a,b}(\boldsymbol{DB}), b/a, N/a)$. Then for all $s_k := (\boldsymbol{DB}[1, a+k], \boldsymbol{DB}[a+k+1, 2a+k], \ldots, \boldsymbol{DB}[N-2a+k+1, N])$ with $1 \leq k < a$, we have

$$s_k \in S.$$

Lemma 5.6 (Correctness of procedure Merge Solutions)**.** *Let $\boldsymbol{DB}$ be a database, $N = |\boldsymbol{DB}|$ and a, b be natural numbers less or equal to N with $b > 2a$. Let $s = (\boldsymbol{DB}[1, a], \boldsymbol{DB}[a+1, 2a], \ldots, \boldsymbol{DB}[N-a+1, N])$. Define $s_k := (\boldsymbol{DB}[1, a+k], \boldsymbol{DB}[a+k+1, 2a+k], \ldots, \boldsymbol{DB}[N-2a+k+1, N])$ with $1 \leq k < a$. Let $S =$ Merge Solutions$(S', s, \mathcal{L}_{a,b}(\boldsymbol{DB}), a, b, N)$. If $s_k \in S'$ for all k, then*

$$(\boldsymbol{DB}[1, a], \boldsymbol{DB}[a+1], \ldots, \boldsymbol{DB}[N-a], \boldsymbol{DB}[N-a+1, N]) \in S.$$

Lemma 5.7 (Correctness of procedure Finalise Solution)**.** *Let $\boldsymbol{DB}$ be a database, $N = |\boldsymbol{DB}|$ and a, b be natural numbers less or equal to N with $b > 2a$. Let $s = (\boldsymbol{DB}[1, a], \boldsymbol{DB}[a+1], \ldots, \boldsymbol{DB}[N-a], \boldsymbol{DB}[N-a+1, N])$. Let $S =$ Finalise Solutions$(s, \mathcal{L}_{a,b}(\boldsymbol{DB}), a, b, N)$. Then*

$$\boldsymbol{DB} \in S.$$

We are ready to prove our main theorem.

Proof of Theorem 5.9. We get statement (2) for free if we can prove statement (1). Backward implication of statement (1) is trivial, as equality of leakage is checked in line 12 of the attack. It remains to prove the forward implication.

($\Rightarrow$) Let $\mathbf{DB}'$ be any database with $\mathcal{L}_{a,b}(\mathbf{DB}') = \mathcal{L}_{a,b}(\mathbf{DB})$, we show that $\mathbf{DB}'$ is one of the solutions to $\mathcal{A}_5(\mathcal{L}_{a,b}(\mathbf{DB}), a, b, N)$. By the correctness of the procedure Initial Solution, we know that $s = (\mathbf{DB}'[1, a], \mathbf{DB}'[a+1, 2a+1], \ldots, \mathbf{DB}'[N-a+1, N])$ or its reflection is one of the solutions returned by procedure Initial Solution. Without loss of generality, we assume s is one of the solutions. By the correctness of the procedure Offset Solutions, the offset solutions s_k associated to $\mathbf{DB}'$ are contained in the set of solutions returned by Offset Solutions. This means the procedure Merge Solutions returns solutions including $\mathbf{DB}'$ with volumes in range 1 to a and $N-a+1$ to N merged. Finally, with the correctness of the procedure Finalise Solution, the solution found in the previous step is restored to $\mathbf{DB}'$ and potentially some other solutions. Check of equality of leakage in line 12 does not affect $\mathbf{DB}'$ as we assumed $\mathcal{L}_{a,b}(\mathbf{DB}') = \mathcal{L}_{a,b}(\mathbf{DB})$ from the beginning. Therefore, we conclude that $\mathbf{DB}' \in \mathcal{A}_5(\mathcal{L}_{a,b}(\mathbf{DB}), a, b, N)$ and the proof is complete. □

5.3.3.3 Experimental Results

We test our attack on the attributes `NDX-26` and `NPR-26` of the HCUP database extensively, as our basic attack recovers the databases on the attributes uniquely with high success rate so attacking the databases with leakage function $\mathcal{L}_{a,b}$ is informative. At the same time, N for the attributes are large enough so that we can study effectiveness of our attack under a variety of choices of a and b. The threshold before aborting is set as $|W|^2$. The experimental results are shown in Table 5.10.

Our attack on the attribute `NDX-26` with $a = 3$ and $b = 9$ has recovered over 78% of the databases uniquely. This suggests that banning queries from small windows alone is

Attribute	a	b	Unique	Ambiguous	Abort	Avg. Time (s)
NDX-26	3	9	78.1%	1.2%	20.7%	0.49
NDX-26	4	12	80.2%	1.1%	18.7%	1.02
NDX-26	4	16	85.0%	2.5%	12.5%	1.90
NPR-26	3	9	79.8%	0.5%	19.7%	0.30
NPR-26	4	12	84.9%	0.3%	14.8%	0.68
NPR-26	4	16	91.6%	0.4%	8.0%	1.92

Table 5.10: Experimental results for the attack on bounded window size.

ineffective as a countermeasure. Furthermore, our attack on the attribute `NPR-26` with $a = 4$ and $b = 16$ is able to reconstruct over 91% of the databases uniquely, indicating that the use of larger b makes the databases more vulnerable to volume leakage attacks.

5.3.4 Partial Reconstruction

Not all databases are uniquely reconstructable as shown by our previous attacks. However, the information in the observed volumes often allows unique reconstruction of a segment of the database. Consider database $\mathbf{DB} = (100, 2, 1, 1, 1, 2, 1)$. The database is not uniquely reconstructable from $\mathcal{L}_3(\mathbf{DB})$. However, by setting 100 as the initial solution and run our basic attack, we find $(100, 2, 1)$ (and its reflection) as the only solution of length 3. This means we have uniquely reconstructed $(100, 2, 1)$ as a segment of the database.

We introduce the partial reconstruction problem as follows. The adversary obtains leakage $\mathcal{L}(\mathbf{DB})$ for some database $\mathbf{DB}$, and his goal is to output a segment of the database $s \in \mathbf{DB}$ (up to reflection). There is more than one choice on how to measure the effectiveness of an adversary with respect to this kind of attacks. Of the several different choices on how to quantify the success of the adversary we consider two. The first simply counts the fraction of values for which the attack recovers the correct counts; the second, also accounts for the size of the counts themselves. For example, an adversary may correctly identify counts for a small fraction of the values, but the total number of records associated to these values can be overwhelming.

5.3.4.1 Partial Reconstruction Algorithm

Intuition. Our partial reconstruction attack can be viewed as a special case of our basic attack where instead of reporting all solutions of length N, we return the longest solution that can be uniquely identified. Databases that can only be reconstructed partially usually have a segment of small volumes. For example, the length of stay of the patients in hospitals is usually less than 10 days, with occasional longer stays. If we use the minimum observed volume as the initial solution, it is very likely for the number of solutions to grow out of control. Hence, we initialise our partial reconstruction attack with the clique-finding algorithm described by Algorithm 5.4. Unlike our basic attack, solutions for the partial reconstruction attack have to be extended in one direction at a time. This is because extending the solutions in both directions inherently introduces ambiguity (and we aim to identify a unique common subsequence).

Attack Overview. Our partial reconstruction attack is described by Algorithm 5.9.

We begin by running Algorithm 5.4 as the initial solution finding algorithm. Reflections are removed from the initial solutions. The set of partial solutions is then extended iteratively by procedures Extend Left and Extend Right of the basic attack. Only the longest partial solution that is unique is kept. For practical purposes, the set of partial solutions is extended as much as possible until some threshold on its size has reached, and the longest unique solution amongst those is kept by the attacker. If a unique solution is found with line 15 of the algorithm, we return the solution in line 16.

Algorithm 5.9 Partial reconstruction attack

1: **input** $W = \{\sum_{i=x}^{y} v_i \mid y - x + 1 \leq b\}, b, N$
2: **output** $(w_1, \ldots, w_m)$ with $m \leq N$ and for all i, $w_i \in W$

3: **procedure** Attack(W, b, N)
4: $S_b \leftarrow$ Initial Solution(W, b)
5: **for** $s \in S_b$ **do**
6: **if** $s^R \in S_b$ **then**
7: $S_b \leftarrow S_b \setminus \{s\}$
8: **for** $i \leftarrow b+1, N$ **do**
9: $S_i \leftarrow$ ExtendLeft(S_{i-1}, W, b)
10: $j \leftarrow \max_i\{i \mid |S_i| = 1\}$
11: **for** $i \leftarrow j+1, N$ **do**
12: $S_i \leftarrow$ ExtendRight(S_{i-1}, W, b)
13: **if** $\min\{|S_i|\} > 1$ **then**
14: **return** ()
15: $j \leftarrow \max_i\{i \mid |S_i| = 1, i \geq j\}$
16: **return** $S_j.pop()$

5.3.4.2 Correctness and Complexity Analysis

Correctness. Correctness of the partial reconstruction attack needs to be formalized differently from other attacks as we do not recover the whole database most of the time. However, whenever we recover a unique solution, it must be a segment of the original database (up to reflection). We call our partial reconstruction attack $\mathcal{A}_6$. The following Theorem establishes the correctness of partial reconstruction attack. The proof is similar to that of the basic attack with the additional constraint that the partial solution is unique so it is omitted.

Theorem 5.10 (Correctness of the partial reconstruction attack)**.** *Let* $\boldsymbol{DB}$ *be a database,* $N = |V|$ *and* b *be any natural number less or equal to* N*. Let* $s = \mathcal{A}_6(\mathcal{L}_b(\boldsymbol{DB}), b, N)$*. Then* $s \in \boldsymbol{DB}$ *or* $s^R \in \boldsymbol{DB}$*.*

Complexity. Algorithm 5.4 takes at most $\mathcal{O}(|W|^b)$ time and space following a standard argument for brute-force clique finding. The solution extension procedures takes at most $\mathcal{O}(|W|^{b-1})$ space and $\mathcal{O}(b \cdot N \cdot |W|^b)$ time as the clique finding step generates at most $\mathcal{O}(|W|^{b-1})$ solutions. Therefore, the overall space complexity of the attack is $\mathcal{O}(|W|^b)$ and the overall time complexity of the attack is $\mathcal{O}(b \cdot N \cdot |W|^b)$.

In practice, we stop the attack as soon as there are more than $|W|^2$ solutions (including the clique finding step). So the space complexity in practice is $\mathcal{O}(|W|^2)$ and the time complexity is $\mathcal{O}(b \cdot N \cdot |W|^3)$.

5.3.4.3 Experimental Results

We choose `LOS` as the attribute to attack from the HCUP dataset since our basic attack performs poorly due to presence of small elementary volumes. In addition to the performance parameters introduced at the start of the section, we measure the fraction of databases for which our attack fails due to the computational threshold of $|W|^2$ or inability to identify a unique solution.

Attribute	b	Length	Volume	Failed	Time (s)
LOS	4	37.5%	95.6%	14.7%	0.31
LOS	6	48.1%	97.3%	14.2%	0.99
LOS	8	54.5%	98.2%	15.3%	2.24
LOS	10	59.0%	98.7%	16.5%	4.54

Table 5.11: Experimental results for the partial reconstruction attack.

Our experimental results are summarised in Table 5.11 where we report both the fraction of values correctly identified (column "Length") and the fraction of correctly identified values but weighted by their value (column "Volume"). Our partial reconstruction attack works on over 85% of the databases on `LOS` for bound b as small as 4. Although only 37.5% of the values can be recovered uniquely on average, it corresponds to 95.6% of the inpatients. This means that the partial reconstruction attack has effectively recovered the majority of the entries of the databases uniquely. The attack is very efficient despite the expensive clique-finding procedure.

5.3.5 Use of Side Information

Side information on the underlying databases can be used to boost our attacks. In this section, we discuss and experiment with two general ways of using side information.

Post-processing. The most straightforward use of side information is to post-process the results of our attacks to further sift the possible solutions output by our attacks. We note that the solutions output by our attacks are naturally ambiguous as it is information-theoretically impossible to distinguish the database from its reflection. With the help of minimal knowledge on the database, the attacker may be able to distinguish the database from its reflection. For instance, over 92% of the databases in HCUP, for the attribute `AGE` it holds that that $\mathbf{DB}[2,5] < \mathbf{DB}[N-4, N-1]$ so almost all databases can be distinguished from their reflections. More interestingly, if the adversary has access to more concrete side information, he may be able to identify the real database from a large set of solutions. For example, if the attacker knows some true volumes of some segments of the database, he can use that information to disambiguate the solutions.

Dynamic. Side information can also be used dynamically to prune the search space more effectively and reduce the number of ambiguous solutions. For example, over 50% of the databases on the attribute `NPR-26` satisfies the condition $\mathbf{DB}[i] > \mathbf{DB}[i+1]$ for $2 \leq i \leq N-1$. This means if the attacker knows the target database has this

property, the partial solutions must be decreasing in the middle, and the volumes used to extend the partial solutions to the left must be larger than the first volume of the partial solution and vice versa.

We experimentally verify the effect of side information on two of our attacks. We attack the attribute `NDX-26` in the setting of missing queries, with the same parameters as those in Table 5.7. In addition to the observed volumes and relevant parameters to the attack, the adversary is given the volume of the answer to up to 3 queries of window size within b. The queries are generated uniformly at random. For each hospital, we run the attack 10 times with freshly selected known queries, and report the average rate of success. We observe a considerable increment in the uniqueness rate of solutions as compared to the case where the attacker has received no additional information (Table 5.7), as shown by Table 5.12.

		Unique			
b	k	$m = 1$	$m = 2$	$m = 3$	Abort
5	1	51.8%	55.7%	57.7%	34.0%
6	1	64.7%	67.5%	68.9%	24.9%
6	2	50.2%	53.6%	55.9%	34.3%

Table 5.12: Experimental data for attack with missing queries on the attribute `NDX-26`. The attacker is given $m = 1, 2, 3$ random known queries.

On the same attribute, we have also tested known $\mathbf{DB}[1]$ and $\mathbf{DB}[N]$ as the side information, and the results are shown in Table 5.13. The uniqueness rates of solutions are higher than those in the previous setting. This hints that the ambiguity in the solutions often comes from the initial and final segments of the databases.

b	k	Unique	Ambiguous	Abort
5	1	63.3%	2.4%	34.3%
6	1	72.9%	1.8%	25.3%
6	2	64.7%	1.6%	33.7%

Table 5.13: Experimental data for attack with missing queries on the attribute `NDX-26`. The attacker is given $\mathbf{DB}[1]$ and $\mathbf{DB}[N]$.

We attack the attribute `NPR-26` in the setting of perturbed volumes, with $b = 5$ and $r = 10$. The attack is conducted on the databases with the property $\mathbf{DB}[i] > \mathbf{DB}[i+1]$ for $2 \leq i \leq N - 1$, and this information is given to the attacker. The side information improves the fraction of the databases that can be reconstructed uniquely from 12.1% (Table 5.9) to 24.8%.

5.3.6 Attacks on Binary-tree-based Constructions

This section extends our basic attack to constructions that try to hide volume leakage by only permitting certain queries [58] and using padding [57].

5.3.6.1 Leakages of the Constructions

Leakage profile $\mathcal{L}_1$. Recall from Section 5.2.1 that in a naive construction, the partition of the documents is leaked through access pattern leakage. To suppress this leakage, Demertzis et al. [58] proposed the Logarithmic-SRC construction which uses

a data structure called *tree-like Directed Acyclic Graph* (TDAG). TDAG can be built from the binary tree used in leakage profile $\mathcal{L}_1$ by adding intermediate nodes to the internal nodes as shown in Figure 5.6. We hereafter refer to the data structure as the Logarithmic-SRC data structure. To perform a query, the client simply retrieves the smallest node that covers the range. The algorithm to find the endpoints of the query is shown in Algorithm 5.10.

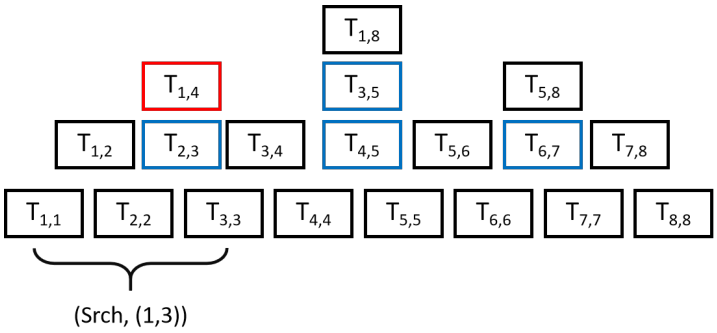


Figure 5.6: The Logarithmic-SRC data structure built for answering range query. The blue nodes are the additional nodes the data structure uses so that the search queries can be covered by a single node with bounded overhead. As an example, to retrieve all documents with values between 1 and 3, node $T_{1,4}$ is returned as the query response.

Algorithm 5.10 Single Range Cover

1: **procedure SRC**(i, j)
2: $\quad w = 2^{\lfloor \log_2(j-i+1) \rfloor}$
3: $\quad i' = \lceil i/w \rceil * w$
4: $\quad$ **return** $(i', i' + w)$

The Logarithmic-SRC construction supports update operations by batching the queries and creating new Logarithmic-SRC data structures periodically. This means that the server is able to learn the information from all the data structures simultaneously through search queries. Let the Logarithmic-SRC data structures stored by the server be $T^1, \ldots, T^t$, we can formally define the leakage profile $\mathcal{L}_1$ be:

$$\mathcal{L}_1(\mathbf{Srch}, (i,j)) = \left(\left(T^1_{\mathbf{SRC}(i,j)}, \left| T^1_{\mathbf{SRC}(i,j)} \right| \right), \ldots, \left(T^t_{\mathbf{SRC}(i,j)}, \left| T^t_{\mathbf{SRC}(i,j)} \right| \right) \right).$$

We abuse the notation $T^*_{\mathbf{SRC}(i,j)}$ to mean the search token(s) associated to the node. It is important to note that the trees are ordered as the adversary can learn that from the insertion operations.

An immediate consequence of this construction is that an attacker can no longer observe query response volumes from all queries. For instance, a search query on $(1, 3)$ will be turned into a search query on $(1, 4)$ by the construction. So previous volume leakage attacks [102, 115, 88] and our basic attack do not apply to this construction. Our attack presented in Section 5.3.2.2 does apply, but it is sub-optimal as the assumed leakage is slightly different.

Leakage profile $\mathcal{L}_2$. To reduce the leakage further, Demertzis et al. [57] proposed a scheme called RANGE-ADJ-SE which combines the Logarithmic-SRC construction

with a static searchable encryption scheme called *searchable encryption with adjustable leakage* (SEAL).

SEAL is built from an oblivious RAM (ORAM) and a padding algorithm. The number of ORAMS maintained by SEAL is 2^α where α is an adjustable parameter. Let N be the number of keywords in the database, then each ORAM stores documents associated to $\frac{N}{2^\alpha}$ keywords.

To build an encrypted database, RANGE-ADJ-SE starts by constructing a Logarithmic-SRC tree with the plaintext database. It then uses the values of the nodes of the Logarithmic-SRC tree as the keywords of SEAL and the contents of the nodes as documents associated to the keywords. The documents are then padded with fake ones such that the number of documents associated to each keyword is the smallest power of x above the current number of documents associated to that keyword, where x is a parameter picked by the user.

To achieve dynamism, the authors have proposed to batch the update queries and build a new encrypted database once in a while. As a terminology, we call the encrypted databases *sub-databases*. As a result of this construction, for each search query, the client has to interact with all sub-databases to retrieve the query responses. In the process, the server learns a set of volumes associated to the search query.

The leakage of the search queries can be described formally as follows. Define $\mathbf{pad}_x(n) = x^{\lceil \log_x(n) \rceil}$. Assume without loss of generality that there are t sub-databases at the point of a search query $(\mathbf{Srch}, (i,j))$. By labelling the Logarithmic-SRC data structures built as $T^1, \ldots, T^t$, the leakage from RANGE-ADJ-SE can be expressed as:

$$\mathcal{L}_2(\mathbf{Srch}, (i,j)) = \left(\mathbf{pad}_x(T^1_{\mathbf{SRC}(i,j)}), \ldots, \mathbf{pad}_x(T^t_{\mathbf{SRC}(i,j)})\right).$$

Note that the volumes in the leakage are ordered as the adversary can tell which sub-database these volumes come from and in which order were these sub-databases added.

5.3.6.2 Volume-leakage Attack against leakage profile $\mathcal{L}_1$

Attack Setting. The volume leakage attack presented in this section works on leakage profile $\mathcal{L}_1$ in the static setting. We assume an honest client who issues uniformly randomly distributed search queries and that the adversary learns the number of documents retrieved for each query. In addition, the adversary learns the query equality, meaning that if $\mathbf{q}_1$ and $\mathbf{q}_2$ are the same search query, the adversary knows that they are equal. The adversary is persistent and passive, meaning that it has access to the query leakage over a long period of time and he does not actively interfere with the client or the server.

To simplify the notation we re-define $\mathcal{L}_1$ as a map between a database and a multiset of natural numbers, that is,

$$\mathcal{L}_1(\mathbf{DB}) = \{\{v[i,j] \mid (i,j) \in \mathbf{Im}(\mathbf{SRC})\}\},$$

where $\mathbf{Im}(\mathbf{SRC})$ is the image of $\mathbf{SRC}$. This leakage function reflects all the volume information the adversary can extract from the queries. Given the leaked volumes $W = \mathcal{L}_1(\mathbf{DB})$, the goal of the adversary is to output a tuple of tuples $((w^1_1, \ldots, w^t_1), \ldots, (w^1_N, \ldots, w^t_N))$ such that for all i, j, $w^j_i = v^j_i$ or for all i, j, $w^j_i = v^j_{N-i+1}$.

Intuition. The attack works in a similar way to the basic attack in Section 5.3.1 where the attack begins with a set of short partial solutions and iteratively extend them until the partial solutions have length N.

The iterative step we use here is different from that of the basic attack as the leakages are very different. Consider a partial solution of length 4, say (w_1, w_2, w_3, w_4). In our basic attack, the adversary can take an observed volume w and check if $(w_1, w_2, w_3, w_4, w_5)$ is a plausible partial solution with 4 additive constraints if the window size is larger than 4. On the other hand, for leakage $\mathcal{L}_1$, we are only able to check if $w + w_4$ is an observed volume as the other additive constraints are not present in the leakage function to begin with. As a result of this, we will get a lot of false positive partial solutions even when the partial solutions are already very long.

To tackle this problem, we design the attack algorithm as follows. The attack initialises the partial solution with all observed volumes. For the subsequent iterations, let S be the set of partial solutions, the attack algorithm checks all pairs of $s_1, s_2 \in S_2$ if $\mathcal{L}_1(s_1 \parallel s_2) \subseteq \mathcal{L}_1(\mathbf{DB})$. Partial solutions $s_1 \parallel s_2$ that do are used as the new partial solutions. The process is repeated until the partial solutions have length N. Finally, the partial solutions that do not generate the exact set of observed volumes are eliminated and the remaining solutions are returned. We note that this approach works for our basic attack and all attacks in Section 5.3.2 too but it is not necessary as those attacks are already efficient enough.

Straightforwardly, this means that the attacker achieves the goal of the attack if the attacker returns one solution that is self-reflecting, or two solutions that are reflections of each other.

Attack Overview. The attack algorithm $\mathcal{A}_7$ takes as input a multiset of tuples W and the maximum value of the database N, and outputs a set of tuples of tuples $\{((w_1^1, \ldots, w_1^t), \ldots, (w_N^1, \ldots, w_N^t))\}$ where each tuple of tuples represents a guess of the distribution of the database $\mathbf{DB}$. The pseudocode of the algorithm can be found in Algorithm 5.11. The algorithm proceeds in three steps. In the first step, the algorithm generates a set of partial solutions of length 1 with all observed volumes. Then, the partial solutions are iteratively extended to produce longer partial solutions. Finally, after the partial solutions hit length N, the solutions that do not generate the exact observed volumes as W are eliminated and the remaining solutions are returned. In the pseudocode, we write $s_1 \parallel s_2$ to mean concatenation of tuples s and w.

Correctness of the Attack. The attack algorithm does not always return a unique solution (up to reflection) as there may be other solutions generating the same leaked volumes as the underlying database. However, the attack algorithm returns all solutions s such that $\mathcal{L}_1(s) = W$. We state this as a theorem and give a proof of it below.

Theorem 5.11 (Correctness of Attack $\mathcal{A}_7$)**.** Let $\mathbf{DB}$ be any database and $W = \mathcal{L}_1(\mathbf{DB})$. Let $S = \mathcal{A}_7(W, |\mathbf{DB}|)$. Then for all databases $\mathbf{DB}'$, $\mathcal{L}_1(\mathbf{DB}') = W$ if and only if $\mathbf{DB}' \in S$.

Proof. ($\Leftarrow$) The proof in the direction is trivial as the databases that do not generate W are eliminated by the last step of the attack (line 13 to 16 of Algorithm 5.11).

($\Rightarrow$) Suppose $\mathcal{L}_2(\mathbf{DB}') = W$. By denoting the segment of the database from value i to j as $\mathbf{DB}'[i, j]$, it is straightforward to see that $\mathcal{L}_2(\mathbf{DB}'[1, i]) \subseteq W$. We are ready to give

Algorithm 5.11 Full Distribution Reconstruction Attack

```
1:  procedure A_7(W, N)
2:      /* Generate the set of initial solutions */
3:      S_1 ← {w | w ∈ W}
4:      /* Iteratively extend the partial solutions */
5:      i ← 1
6:      while i < N do
7:          S_2i ← {}
8:          for s_1, s_2 ∈ S_i × S_i do
9:              if L_1(s_1 ‖ s_2) ⊆ W then
10:                 S_i ← S_i ∪ {s_1 ‖ s_2}
11:         i ← 2 · i
12:     /* Finalise the solutions */
13:     S ← {}
14:     for s ∈ S_N do
15:         if L_1(s) = W then
16:             S ← S ∪ {s}
17:     return S
```

a proof using induction. The attack is initialised with all $w \in W$ as the initial solutions. We know that $\mathbf{DB}'[i,i] \in W$ so $\mathbf{DB}'[i,i] \in S_1$ for all $\mathbf{DB}'$. Assume that $\mathbf{DB}'[ni+1, ni+n] \in S_1$ for all $i \in \{0, \ldots, \frac{N}{n} - 1\}$, we want to show that $\mathbf{DB}'[2ni+1, 2ni+2n] \in S_1 2n$ for all $i \in \{0, \ldots, \frac{N}{2n} - 1\}$. This is indeed the case as $\mathbf{DB}'[2ni+1, 2ni+n]$ and $\mathbf{DB}[2ni+n+1, 2ni+2n]$ are in S_1, and the check in line 9 of Algorithm 5.11 passes for $\mathbf{DB}'[2ni+1, 2ni+2n]$. As n grows to N, we get $\mathbf{DB}'[1, N] \in S_1$. Finally, these solutions will not be eliminated by the last step of the algorithm ((line 13 to 16 of Algorithm 5.11)) as $\mathcal{L}_2(\mathbf{DB}') = W$ and the proof is complete. □

A Variant of the Attack. We show how attack $\mathcal{A}_7$ can be modified to attack a related scheme called RANGE-SRC-SE [57] with certain parameters. RANGE-SRC-SE is a construction built from the Logarithmic-SRC construction and SEAL, where SEAL is a searchable encryption scheme with two adjustable parameters α and x. SEAL uses 2^α oblivious RAMs to store the nodes of the Logarithmic-SRC data structure, and the volume of each node is padded to a power of x. As ORAM is costly, the authors recommend $\alpha = \log(N)$ for their RANGE-SRC-SE construction. With regards to the choice of x, the authors recommended $x = 16$ for their dataset but the slow-down factor from the Logarithmic-SRC scheme they reported in the paper is of order 10^5. This renders this scheme impractical for some applications under these choices of parameters.

Some practitioners may be willing to pay the price in terms of oblivious RAM but not in terms of padding. This is because the use of oblivious RAM only increases the latency but not the storage. On the other hand, if the nodes of the Logarithmic-SRC data structure are padded to a power of x, then the storage expands by a factor of x too. The storage cost may be too high with padding as the Logarithmic-SRC data structure already introduces an expansion factor of $\mathcal{O}(\log(N))$.

In effect, this means some practitioner may instantiate RANGE-SRC-SE with a large α and $x = 1$. This of course invalidates our previous attack $\mathcal{A}_7$ as the search pattern

is hidden by the oblivious RAMs, but with a simple adjustment, our attack can still be applied to RANGE-SRC-SE.

Formally, let P_α be a partition of the nodes of the Logarithmic-SRC data structure such that there are 2^α equally distributed subsets. Then after all queries have been observed by the adversary, the leakage of the database can be expressed as:

$$\mathcal{L}_2'((v_1, \ldots, v_N)) = \{\{v[i,j] \mid (i,j) \in P\} \mid P \in P_{2^\alpha}\}.$$

Unsurprisingly, with a choice of large α, the ORAMs contain only a handful of the nodes, which means there is a good chance that the observed volumes from the ORAMs are unique. If that is indeed the case, then we can build the multiset of observed volumes from the smaller sets directly and run attack $\mathcal{A}_7$. If that is not the case, then the adversary can at least learn the upper bound of the multiplicities of the observed volumes. By representing the upper bound as a multiset as shown in Algorithm 5.12, the adversary can attempt to run attack $\mathcal{A}_7$ anyway to generate all the plausible solutions. However, the attacker should not run the final part of Algorithm 5.11 as **Merge_Vols**($\mathcal{L}_1'$(**DB**)) is not equal to $\mathcal{L}_1$(**DB**). We call the modified attack $\mathcal{A}_8$.

Algorithm 5.12 Multiset Generation

```
1: /* M is the maximum number of unique volumes in an ORAM */
2: procedure MULTISET_GEN(W, M)
3:     W' ← {{}}
4:     for w ∈ W do
5:         m ← M − |w| + 1
6:         for i ∈ {1, ..., m} do
7:             W' ← W' ∪ w
8:     return W'
```

5.3.6.3 Volume-leakage Attack against leakage profile $\mathcal{L}_2$

ATTACK SETTING. The volume leakage attack works on leakage profile $\mathcal{L}_2$ in the dynamic setting where there are sufficient number of update operations. We assume an honest client who generates uniform search queries and update the database regularly. The adversary on the other hand, is honest-but-curious. He learns the volume information of the queries as the client issues the queries. . For simplicity, we assume that the adversary is interested in the distribution of a snapshot of the database **DB**. As **DB** is made up of sub-databases generated by update queries, we can write $\mathbf{DB} = ((v_1^1, \ldots, v_N^1), \ldots, (v_1^t, \ldots, v_N^t))$. From the observed volumes, the goal of the adversary is to reconstruct the distribution of the database. However, due to the use of padding, the distribution reconstructed by the adversary is only approximate. For example, if 5 is a volume in the database, and all volumes are padded to a power of 2, then the adversary is going to observe 8 instead of 5, and the best guess it can have on the true volume is a range between 5 and 8.

INTUITION. As the volumes are padded, the attacker can no longer use attack $\mathcal{A}_9$ to reconstruct the distribution of the database, however, as similar strategy can be used to identify adjacent volumes. To demonstrate the idea, consider a database $\mathbf{DB} = ((v_1, \ldots, v_N))$ such that $v_1 = 3$ and $v_2 = 10$, and the volumes are padded to a power of

2. Due to the padding, the two volumes will become 4 and 16 respectively. Certainly, the attacker cannot verify if $4 + 16$ is an observed volume as all observed volumes are padded, but if 4 and 16 are two adjacent volumes, then the volumes before padding must be between 3 and 4, and 9 and 16 respectively. This implies that the sum of the volumes must be between 12 and 20, and hence, the observed volume must be 16 or 32.

An attack with this approach can certainly recover the original database, but there is likely going to be multiple databases generating the same leakage as padding is removing a lot of distributional information. However, this does not mean all databases cannot be reconstructed uniquely given the padded volumes. Recall that updates on the database is done by batching the queries and creating new sub-databases once in a while. In particular, if the database has been updated several times, then the adversary can learn a set of volumes for each search query, and use that to improve the reconstruction attack. Consider the following example where the database is $\mathbf{DB} = ((v_1^1, \ldots, v_N^1), (v_1^2, \ldots, v_N^2))$. A search query with $(\mathbf{Srch}, (1, 1))$ gives leakage $(4, 16)$, a search query with $(\mathbf{Srch}, (2, 2))$ gives leakage $(16, 1)$. Using the same argument as before, the observed volume for a search query that combines v_1^1 and v_2^1 on the first sub-database must be 16 or 32, and that on the second sub-database must be 16 or 32. Combining these two together gives that the observed volume for the whole query must be $(16, 16)$, $(16, 32)$, $(32, 16)$ or $(32, 32)$. The chance of a false positive is lower in this case as the actual observed volumes are entangled.

From the practical perspective, the client is likely to create small sub-databases for the updates as it otherwise defeats the purpose of outsourcing. This means that a search query will retrieve a large number of volumes and make it easier for the attacker to identify the distribution of the database uniquely.

Attack Overview. The attack algorithm $\mathcal{A}_9$ takes as input a set of tuples W and the maximum value of the database N, and outputs a set of tuples of tuples $\{(w_1, \ldots, w_N)\}$ where each tuple of tuples $(w_1, \ldots, w_N)$ represents a guess of the padded version of $\mathbf{DB}$. Just as attack $\mathcal{A}_9$, the algorithm begins by generating the set of initial solutions. All tuples of observed volumes are used as the initial solutions. Then the solutions are extended iteratively just as before. The only difference is that the attacker can no longer check for exact equalities due to padding. To resolve the problem, the attacker computes the lower and upper bounds on the sums of volumes, and check if they exist in the set of observed volumes. The finalization step is omitted from the attack as the sums of elementary volumes are not uniquely identified by the elementary volumes.

Correctness of the Attack. The attack algorithm does not always return a unique solution (up to reflection) as there may be other solutions generating the same leaked volumes as the underlying database. However, the attack algorithm returns all solutions s such that $\mathcal{L}_2(s) = W$. We state this as a theorem and below. We omit the proof as it is very similar to the proof of Theorem 5.11.

Theorem 5.12 (Correctness of Attack $\mathcal{A}_7$)**.** Let $\mathbf{DB}$ be any database and $W = \mathcal{L}_2(\mathbf{DB})$. Let $S = \mathcal{A}_9(W, |\mathbf{DB}|)$. Then for all databases $\mathbf{DB}'$, $\mathcal{L}_2(\mathbf{DB}') = W$ if and only if $\mathbf{DB}' \in S$.

Algorithm 5.13 Approximate Distribution Reconstruction Attack

procedure $\mathcal{A}_9(W, N)$
  /* *Generate the set of initial solutions* */
  $S_1 \leftarrow \{(w) \mid w \in W\}$

  /* *Iteratively extend the partial solutions* */
  $i \leftarrow 1$
  while $i < N$ **do**
    $S_{2i} \leftarrow \{\}$
    for $s_1, s_2 \in S_i \times S_i$ **do**
      if $s_1 \parallel s_2$ can be generated from W **then**
        $S_i \leftarrow S_i \cup \{s_1 \parallel s_2\}$
        $i \leftarrow 2 \cdot i$
  /* *Finalise the solutions* */
  $S \leftarrow \{\}$
  for $s \in S_N$ **do**
    if $\mathcal{L}_2(s) = W$ **then**
      $S \leftarrow S \cup \{s\}$
  return S

5.3.6.4 Experimental Results

We present the experimental results of the attacks on Logarithmic-SRC, RANGE-SRC-SE and RANGE-ADJ-SE in this section.

Dataset and Pre-processing. We used the HCUP dataset (see Section 5.3.1.5) for our experiments. To simulate document insertions, each of the database is split into sub-databases of size 128, 256 and 512 respectively. In the figures below, the experimental results for sub-databases of different sizes are shown in that order. The splitting of the database is done uniformly randomly, so that the elementary volumes of the sub-databases are distributed identically. For the attacks on Logarithmic-SRC and RANGE-SRC-SE, the sparse version of the dataset is used. For the attacks on RANGE-ADJ-SE, the dense version of the database is used.

Experimental Setup. The attacks we have presented in this section can take a long time to terminate as the number of partial solutions can grow rapidly. To address this problem, we pre-emptively terminate an attack if the number of partial solutions at any iteration exceeds $10\,|W|^2$ where W is the set of observed volumes, or if the time taken for an iteration exceeds 120 seconds. In these cases, we report the results of the attacks as aborts.

Experimental Results on Attack $\mathcal{A}_7$. The experimental results on the attack against Logarithmic-SRC are shown in Figure 5.7. The attack is able to recover an overwhelming proportion of the dense databases. On sparse databases, the attack performs well on attributes with small numbers of values and struggles against attributes with large numbers of values. Length of stay (`LOS`) is the hardest attribute to attack with less than 20% of the sparse databases reconstructed uniquely. There is a significant proportion of the databases on `LOS` that can only be reconstructed ambiguously, indicating that the attack is weak against sparse databases with long and flat tails.

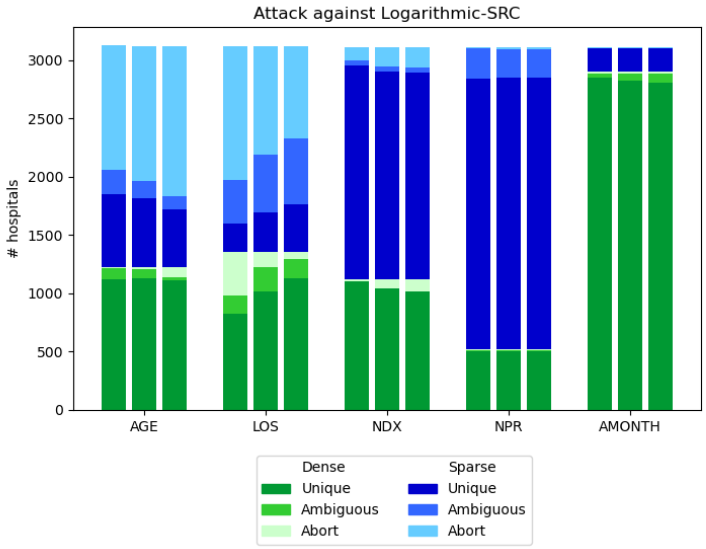


Figure 5.7: Experimental results on the attack against Logarithmic-SRC.

Experimental Results on Attack $\mathcal{A}_8$. The experimental results on the attack against Range-SRC-SE are shown in Figure 5.8. There are two sets of experiments performed. The first set (top figure) used small ORAMs with size 16 each ($\alpha = \log(N) - 4$), whereas the second set (bottom figure) used large ORAMs with size 64 each ($\alpha = \log(N) - 6$). The attack with larger ORAMs is noticeably worse on attributes `AGE`, `LOS` and `NDX`. The attack almost always aborts on sparse databases, indicating that the use of ORAMs is effective in protecting sparse databases against volume-leakage attacks.

Experimental Results on Attack $\mathcal{A}_9$. The experimental results on the attack against RANGE-ADJ-SE are shown in Figure 5.9. The attack almost always aborts on the attributes with large numbers of values so they are not shown in the figure. On the attributes with relatively small numbers of values, the attack is still able to recover a significant proportion of the databases uniquely. Attack $\mathcal{A}_9$ performs better on small sub-databases. This can be explained by the fact that smaller sub-databases means more of them, so there are more constraints that can be tested by the attack, leading to more unique recoveries.

5.4 Discussion

This chapter studies access-pattern leakage attacks and volume leakage attacks on structured encryption schemes that support range queries. We show that the constructions with countermeasures in mind in the literature still do not offer enough security as the structures in the leakages can be exploited by an attacker relatively easily.

A natural question to ask is: why do we not use more aggressive parameters on the existing schemes to achieve security? We certainly can do that, except that it will make the already inefficient schemes worse. Just to quote a few numbers (see Figure 13 of [57]), RANGE-SRC-SE has a query time overhead on order of 10 times if the sizes of

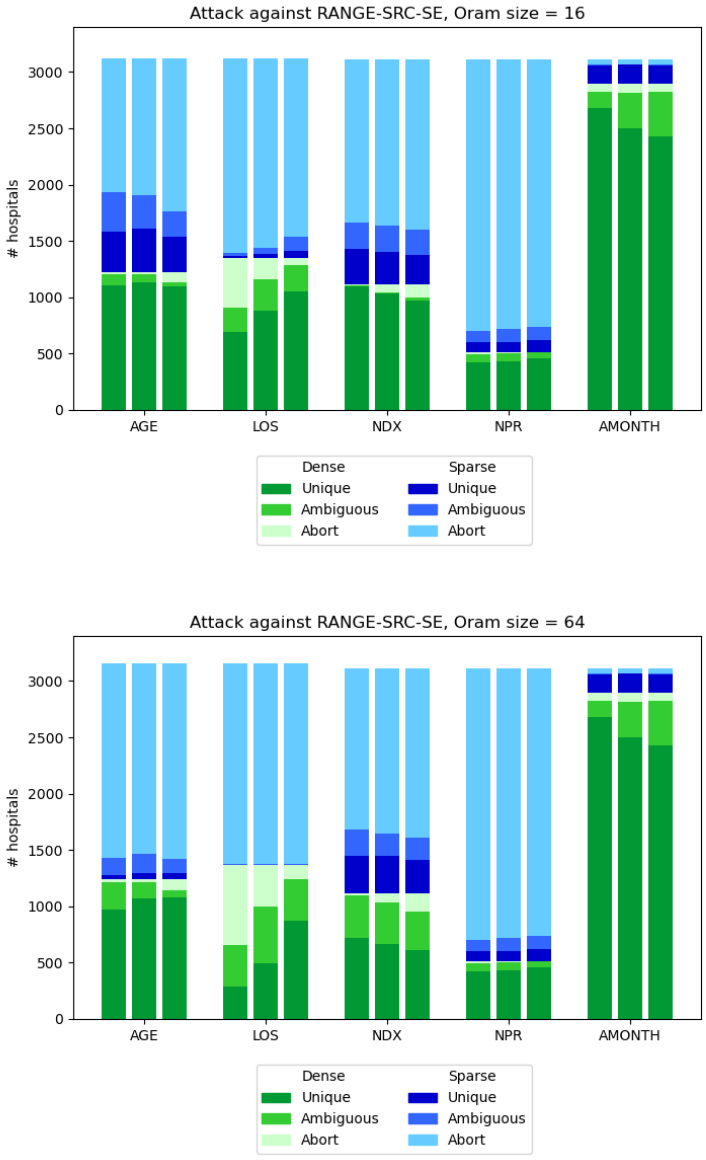


Figure 5.8: Experimental results on the attack against Range-SRC-SE.

the nodes in the Logarithmic-SRC tree are padded to powers of 2 and no ORAM is used; the overheads grows to order of 10^2 times if the sizes of the nodes are padded to powers of 2^5. If RANGE-ADJ-SE is used instead, these two numbers are on order of 10^4 instead. To make it worse, the overheads here are with respect to a searchable encryption scheme, which means it is a few more orders of magnitudes slower than a plaintext database. This gives practitioners very little incentive to adopt a scheme like this.

It is not to say that we do not value security. It is important for us to hide access-pattern leakage and suppress volume leakage. However, we need to take a radically different approach to this in order to come up with a scheme that is *practically secure*

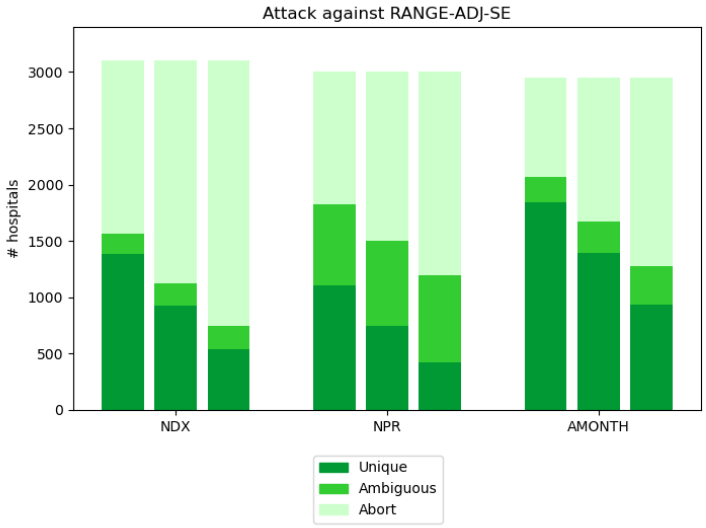


Figure 5.9: Experimental results on the attack against RANGE-ADJ-SE.

and efficient. We present our key ideas and leave a full construction as future work.

To hide access-pattern leakage and still achieving a good storage overhead, we need to avoid duplication as much as possible. Instead, we need to rely on random reads and writes just like an ORAM. However, using a full ORAM incurs too much query time overhead so we need to look for a more efficient (and less secure) alternative.

To suppress volume leakages, we need to be a lot more aggressive and use padding in a way which is less dependent on the true query response volumes. Recall from our attacks in Section 5.3, one of the key reasons why the attacks work against the padding-based countermeasures is because the paddings used in those countermeasures depend on the true query response volumes. For example, if we start with $v_1 = 2$ and $v_2 = 10$ and pad both to the next power of 2, we get $v'_1 = 2$ and $v'_2 = 16$. Even though we cannot tell the original volumes from the padded ones, we can still tell that $v_1 < v_2$ and $v_1 + v_2$ falls within a certain range. To address this, one possible solution is to use random amount of padding on the fly, meaning that every time a query is issued, the number of documents retrieved changes in some way. This is of course not going to work by itself if the access pattern is leaked, so we do need to consider the padding strategy together with how access-pattern is suppressed.

As a starting point, we can also look for inspirations from the rich literature of traffic analysis and countermeasures [14, 72, 196, 54]. In particular, dummy traffic and traffic morphing are similar to the countermeasures we have proposed above, and they can be explored further for better security and efficiency trade-offs.

In all, designing an efficient and secure structured encryption scheme is challenging, and it is a recurring theme in the chapters to follow.

Chapter 6

Cryptanalysis II: Searchable Encryption

A keyword search on text-based database usually consists of two steps. In the first step, a search is performed on an *inverted index* and the client gets the set of document identifiers containing the queried keyword by the end. In the second step, these document identifiers are used to retrieve the actual documents from the document array.

A typical searchable encryption scheme works in a similar way: an encrypted inverted index and an encrypted document array are built, and a search query is broken into document identifier retrieval and document retrieval just as before. In this chapter, we show that almost all searchable encryption schemes can only be used to solve the document identifier retrieval problem, and are not scalable for the document retrieval problem. We further argue that the only efficient deployment of searchable encryption is to use a naively encrypted document array, which generates additional leakage which we call *system-level leakage*. We design novel query reconstruction attacks and show how this source of leakage can be exploited.

This chapter is organised as follows. In Section 6.1, we experimentally demonstrate inefficiency of the state-of-the-art searchable encryption schemes at the document retrieval problem, and argue that the only efficient use of searchable encryption is to encrypt the documents naively. In Section 6.2, we give a detailed overview of previous attacks and motivate our own attacks. Section 6.3 gives a formal introduction to access-pattern leakage-abuse attacks. Section 6.4 identifies the leakages of the constructions we are targetting and develops the attack algorithm we use. Section 6.5 provides an empirical evaluation of our attacks. Finally, we close the chapter with a discussion in Section 6.6.

Contents

6.1 Efficient Deployment of Searchable Encryption

Recall from Section 4.2 that the core idea of designing a searchable encryption scheme is to build an inverted index and encrypt it in a clever way so that it can be queried later. To suppress leakage, the most common technique used is called *duplication*, where instead of having an inverted index between keywords and sets of document identifiers, an inverted index between *instances* of keywords and document identifiers is used. A pictorial demonstration of the technique can be found in Figure 4.1.

As we have argued, the duplication technique is not scalable – it may be fine to be used on inverted indices as the entries are short, but the storage overhead it creates on actual documents easily overwhelms any modern storage system. We demonstrate this experimentally in Section 6.1.1.

This leaves us with two other options as outlined in Section 4.6, namely using searchable encryption to build an encrypted index only and relying on a space-efficient primitive to handle document retrieval, or using searchable encryption to build an encrypted index only and encrypting the document array naively. We show experimentally in Section 6.1.1 that the earlier option is not practically feasible and the only efficient deployment of searchable encryption is the later option.

6.1.1 Comparison of Overheads between Deployments of Searchable Encryption

6.1.1.1 Target Schemes

In this section, we investigate the overheads incurred when one tries to protect the actual documents in the same way as the inverted index. This means the actual document retrieval should not leak more than what the inverted index already does. For example, it is okay to leak query equality (when two queries are for the same keyword) but it is not okay to leak query response volume (how many documents match a query) or co-occurrence (if two different queries touch the same document). We consider three document encryption methods based on searchable encryption schemes and two additional methods based on other primitives.

The three searchable encryption schemes include a naive searchable encryption scheme without duplication (see the last construction in Section 4.2) and two state-of-the-art SSE schemes. We pick pseudo-random transform from [98] and volume-hiding encrypted multi-maps from [149][1] as they represent the typical storage overhead seen in the literature. We give a brief overview of the two schemes below.

The two other primitives are Path ORAM [171] and SealPIR [6]. We choose these two constructions because they are the state-of-the-art schemes in their respective fields. We give a brief overview of the two schemes below.

Pseudo-random Transform. As a potential countermeasure to leakage-abuse attacks, Kamara and Moataz [98] introduced the concept of *volume-hiding* encrypted multi-maps (EMMs) that hide response length patterns (and exact access patterns)

[1] These schemes are called multi-maps but they are really intended to be used as searchable encryption.

while providing better search performance compared to worst-case padding (adding fake documents so that all keywords return the same number of documents). They proposed the first construction of volume-hiding EMMs based on an obfuscation mechanism called *pseudo-random transform*. The idea is to pad or truncate the query response lengths of queries on a multi-map with a pseudo-random function. We call this construction **PRT-EMM** in this thesis.

Volume-hiding Encrypted Multi-maps. Patel *et al.* proposed two volume-hiding encrypted multi-maps (EMM) in [149]. Both of the constructions use Cuckoo hashing [145] as the underlying data structure. The two schemes proposed by the authors are only different in terms of the padding mechanism on the query response lengths.

The first scheme uses full padding, meaning that all query response lengths are padded to the maximum query response length. This is done by querying additional addresses in the hash table deterministically (generated by a pseudo-random function) so no additional fake documents are required. We call this construction **FP-EMM** in this thesis. The second scheme uses differentially-private volume hiding as opposed to full padding. It has the same storage overhead as the first scheme. We call the construction **DP-EMM** in this thesis.

Path ORAM. Path ORAM is an ORAM construction proposed by Stefanov et al. [171] with a logarithmic communication overhead and small client storage. The idea of the construction is as follows. The server stores the data elements (documents in the searchable encryption setting) in random locations of a binary tree. The binary tree has depth $\log(N)$ where N is the number of data elements and each node of the tree consists of Z data blocks of size B each. The client stores the the locations of the data elements locally. On top of that, it also has a stash that can be used to store a small number of data elements.

When the client wants to fetch a data element, it first looks up its location in the lookup table he stored locally. It then fetches all data elements on the path of the target data element. At this stage, it has already obtained the data element it is looking for. After that, it randomises the locations of the data elements that are in the stash and flush as many of them as possible to the binary tree stored on the server.

It is possible to reduce the storage on the client further by storing the lookup map for the locations of the data elements on the server by recursively applying the ORAM construction outlined above, but we do not consider that in our experiments below for simplicity.

It is straightforward to use Path ORAM to retrieve documents. The client can simply break down the documents into chunks that are small enough to fit into the blocks of Path ORAM and simply run the setup algorithm of Path ORAM. When the client wants to make a query, it first queries the inverted index to obtain the document identifiers, the documents can then be fetched from the ORAM one by one. To hide volume leakage, the client can pad the number of fetch requests to the maximum query response volume, i.e. making dummy queries to the ORAM if the true query response volume is smaller.

SealPIR. SealPIR [6] is a computational private information retrieval scheme that is highly efficient. On a high level, the scheme uses FHE to encode the plaintexts (performed by the server). When the client wants to retrieve a particular plaintext, he encodes the position of that plaintext in a few FHE ciphertexts and send them to the

server. The ciphertexts are then processed by the server to obtain a few FHE ciphertexts that contain the response and they are sent back to the client.

To turn SealPIR into a document retrieval scheme, we can simply encrypt the documents first before running the SealPIR protocol. Similar to the case of Path ORAM, the query response volume has to be padded to the maximum query response volume to hide volume leakage.

6.1.1.2 Experimental Data

We use the Enron email corpus [194] as the target database as it is the most common choice in the literature. This dataset is also used for other experiments later in this thesis. This section provides some general information on the Enron email corpus and how it is pre-processed in our experiments.

General Information. The Enron email corpus is a collection of over 600 thousand emails generated by 158 employees of the Enron Corporation and acquired by the Federal Energy Regulatory Commission (FERC) during its investigation of the Enron scandal. At the conclusion of the investigation, and upon the issuance of the FERC staff report, the email corpus is released to the public for historical research and academic purposes. The Enron dataset is widely used as a target for cryptanalysis on structured encryption [94, 33, 17] as it is one of the only public real-world dataset.

Pre-processing. We implemented our email processing and keyword extraction script in Python using the Natural Language Toolkit [156] module as the tokeniser. The English stop words and other keywords with frequency higher than 5% are removed. Some of our experiments require stemmed keywords and we used the Porter Stemming Algorithm [153] for this process.

To suppress document volume leakage, the emails in our dataset are split into documents of length at most two thousand characters. If this cannot be done for any reason (e.g. the header is longer than two thousand characters), the emails are split into documents of length the smallest multiple of two thousand characters.

General Statistics. Figure 6.1 gives some general statistics of the Enron email corpus after pre-processing.

	Without Stemming	With Stemming
# documents	480000	480000
# keywords	33366	24947
# keyword-document pairs	17415721	16881119
Max. keyword frequency	23989	40714
Min. keyword frequency	1	1
Mean keyword frequency	522.0	676.7
Max. # keywords per document	3483	2939
Min. # keywords per document	1	1
Mean # keywords per document	36.8	35.7

Figure 6.1: General statistics of the Enron email corpus after pre-processing.

Figure 6.2 shows the frequency distribution of the 5000 most frequent keywords after pre-processing.

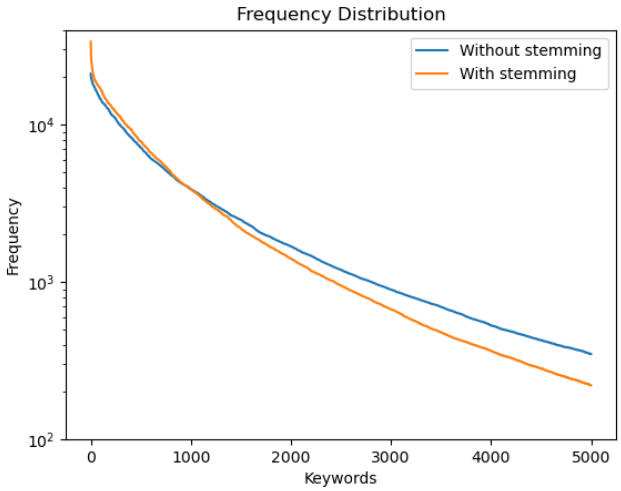


Figure 6.2: Frequency distribution of the 5000 most frequent keywords.

6.1.1.3 Empirical Evaluation

Parameters. The following parameters are used in our evaluation. We use PRFs with 256-bit output. We use the most space-efficient parameters for **PRT-EMM** proposed in the original papers [98], namely $\alpha = 0.5$. For Path ORAM [171], we assume each block has size 1 KB and there are 4 blocks per bucket. For SealPIR, we assume that the degree of ciphertext is $N = 2048$, the size of the coefficients are 60 bits and the database is represented in $d = 2$ dimensions as per the original paper [6].

Evaluation. We report communication volume, storage cost, and the number of core cryptographic operations needed for each option described above in Table 6.1. We split computation and communication costs into client and server costs, and report only server storage costs (client storage costs are low).

Storage and communication costs are measured in total volumes. Additional overheads arising from how the data is structured and packaged for communication are ignored.

Computation costs are measured by the number of core cryptographic operations. The operations that we consider include: prf for PRF computation; enc and dec for encryption and decryption with a symmetric primitive; acc for disk/RAM access (read or write); henc and hdec for encryption and decryption with FHE; hmul, hsub, hadd for multiplication, substitution and addition for FHE ciphertexts. Reporting operation counts in place of running times makes our comparison independent of implementation details.

We opt to not include latency as this depends on several factors such as data access speed and network delay, and these are hard to compare concretely and fairly.

Scheme	Storage (Server)	Query (Client)		Query (Server)	
		Computation	Communication	Computation	Communication
Naïve*	470 MB	f prf f dec	32 B	f acc	f KB
Duplication	17 GB (36x)	24K prf 24K dec	750 KB (46x)	24K acc	23 MB (46x)
PRT-EMM [98]	390 GB (860x)	12K prf 12K dec	370 KB (23x)	12K acc	12 MB (23x)
FP-EMM [149]	43 GB (94x)	48K prf 48K dec	1.5 MB (92x)	48K acc	47 MB (92x)
SealPIR [6]	120 GB (260x)	1 henc 1 hdec	1.46 GB (94,000x)	23B hmul 11B hsub 11B hadd	5.9 GB (12,000x)
Non-recursive Path ORAM [171]	1.8 GB (4x)	3.4M acc 1.7M dec 1.7M enc	1.65 GB (110,000x)	3.4M acc	1.7 GB (3,300x)

Table 6.1: Evaluation of different document retrieval primitives with minimal leakage. The numbers in the brackets indicate overheads beyond the baseline provided by the **Naïve** scheme. We assume 522 documents (mean keyword frequency) are retrieved by the **Naïve** scheme in the computations of the overheads. *: f is the real query response volume (since there is no padding).

Discussion. It is clear that all of the options suffer significant storage overheads. For **Duplication**, **PRT-EMM** and **FP-EMM**, this is caused by duplication. The expansion factor grows linearly with the number of keywords per document. For PIR, the expansion factor comes from the use of homomorphic encryption. It is not clear how the ciphertexts can be compressed to reduce the overhead. It is conceivable that alternative PIR schemes might avoid such expansion. For ORAM, the overhead comes from the use of multiple blocks per bucket. This is necessary to prevent overflowing buckets, meaning the storage overhead cannot be reduced significantly.

With regard to queries, **Duplication**, **PRT-EMM** and **FP-EMM** have reasonable computational costs, but the communication costs from the server to the client are high in each case. The server needs to send 2.5% to 10% of the entire document collection to the client per query, which is a lot more than the average keyword frequency might suggest (0.109% for the Enron corpus). The PIR and ORAM options naturally suffer from high computation and/or communication overheads since they are not designed for large-scale document retrieval. In fact, with bad choice of parameters, as shown in [79], the communication volume produced and ORAM can be several orders of magnitude larger than the database itself.

Of course, the state-of-the-art implementations of various cryptographic primitives that we have chosen for our experiments could very well be improved upon in terms of concrete efficiency in future work. Nonetheless, the concrete numbers that we provide are indicative of what is possible by employing state-of-the-art approaches for realizing these primitives.

6.1.2 Discussion

To summarise, with currently available techniques, searchable encryption can only be used as a tool for secure index retrieval, and there is no known secure primitive that is both time and space-efficient for document retrieval. If we ever want to use searchable encryption in an efficient way, we are forced to encrypt and retrieve the documents naively.

This chapter shows that the approach above inevitably generates more leakage than what the schemes claimed for. And this source of leakage, which we call *system-level leakage*, can be exploited by an adversary to achieve query and data reconstruction

attacks.

In Section 6.2, we give a detailed overview of access-pattern leakage attacks in the literature and outline their shortcomings. In Section 6.3, we formally define access-pattern leakage attack. Section 6.4 describes our generic attack and how we apply it to the state-of-the-art schemes. Section 6.5 provides empirical evaluation of our attacks and demonstrate their effectiveness on the state-of-the-art schemes on real-world datasets. Finally, we conclude with final remarks in Section 6.6.

6.2 Attack on System-level Leakage

In this section, we give a detailed overview of our query reconstruction attacks on the state-of-the-art searchable encryption schemes [98, 149], assuming that they are only used to encrypt the inverted index, and a naively encrypted document array is used.

We begin by giving an informal description of access-pattern leakage attacks and an overview of the existing attacks in the literature. We show that those attacks are insufficient to break the state-of-the-art searchable encryption schemes even if the document array is naively encrypted and motivate our own attacks.

6.2.1 Access-pattern Leakage Attacks

Access-pattern Leakage. Access-pattern leakage is an important class of leakage for searchable encryption. Informally, access-pattern leakage can be described as the document identifiers that an adversary can learn for each query. Using the last construction in Section 4.2 as an example, a query on keyword kw consists of two steps:

1. the client computes $F(kw)$ and retrieves all encrypted document identifiers associated to keyword kw,
2. the client decrypts the encrypted document identifiers and retrieves all encrypted documents associated to those identifiers.

For simplicity, we assume that the document identifiers of the encrypted documents are their original document identifiers, i.e. they are not encrypted. Encrypting the document identifiers (e.g. by using PRF output of the original document identifiers) does not add any security against our attacks as the same encrypted documents are retrieved regardless of whether the document identifiers are encrypted.

Aggregated Access-pattern Leakage. Access-pattern leakage becomes interesting when multiple queries are made. Consider a collection of documents $\{doc_1, doc_2, doc_3\}$ where document doc_1 contains keywords kw_1 and kw_2, document doc_2 contains keywords kw_1 and kw_3, and document doc_3 contains keywords kw_3. A query on keyword kw_1, kw_2, kw_3 leaks document identifiers $\{1, 2\}, \{1\}, \{2, 3\}$ respectively. As queries on keywords kw_1 and kw_3 both return two documents, it is impossible to distinguish them just by using query response volume. However, only keyword kw_3 appears by itself in a document, so by looking at the access-pattern leakage from multiple queries, an adversary can distinguish a query on keyword kw_1 from that on keyword kw_3.

Access-pattern Leakage Attacks. There are three major factors in characterising an access-pattern leakage attack:

- **Attack Target.** The adversary may either target query recovery or data recovery (or both).
- **Attack Assumptions.** The adversary may have access to some auxiliary data. In a known-data attack, the adversary knows a subset of entries in the original database. In an inference attack the adversary has a set of entries that are distributed statistically close to the entries in the database (but not necessarily an actual subset of the data). In a known-query attack, the adversary knows a subset of the queries made by the client.
- **Attack Nature.** The adversary may either passively observe the leakage (referred to as leakage-abuse attacks) or actively create leakage by tampering with the client's database (referred to as injection attacks).

We say that an access-pattern leakage attack is successful if it breaches the security guarantee offered by the target searchable encryption scheme.

6.2.2 Previous Attacks

Known-data Attacks. Most of the query reconstruction attacks exploiting access-pattern leakage known in the literature are known-data attacks. This includes the first attack proposed by Islam *et al.* [94], the Count attack proposed by Cash *et al.* [33] and the more recent attacks proposed by Blackstone *et al.* [17].

These attacks are known-data attacks for different reasons. For the IKK attack [94], the authors did propose the attack as an inference attack (i.e. auxiliary data is a noisy statistical representation of the target database). However, Cash *et al.* [33] pointed out later in their paper that the IKK attack only performs well as a known-data (and known-query) attack. They have also proposed their own attack called Count attack [2]. Later in [17], Blackstone *et al.* pointed out that the earlier two attacks only performs well when the attacker has access to close to the entire database in the known-data setting, and developed new known-data attacks which work well even when only a small fraction of the documents are known to the attacker.

Inference Attacks. There are two works on inference attack against searchable encryption for query reconstruction in the literature. The first work was developed by Pouliot and Wright [155]. The paper is a direct improvement over the IKK attack: the authors proposed to use graph-matching algorithms instead of simulated annealing as the optimization algorithm, and showed that their attacks work in the inference setting. The attack targets of the paper are Shadowcrypt [92] and Mimesis aegis [119]. Shadowcrypt uses deterministic hashing to generate search tags so it leaks naive access pattern. On the other hand, Mimesis aegis uses bloom filters, and its leakage is more complicated. Still, the authors managed to show that their attack works in the latter case.

[2]The authors found bugs in their conference version of the paper, and updates have been made in their eprint version of the paper [34]

More recently, Oya and Kerschbaum [143] have proposed the first inference attacks exploiting search pattern and volume leakage at the same time [3]. Their attack works even if the volume leakage is perturbed by the underlying construction [41, 149, 57].

Other Attacks. Attacks with other assumptions on auxiliary data and adversarial power are studied in the literature too. For example, the IKK attack described earlier is in fact a known-query attack too. There are also attacks [200, 17] assuming stronger adversarial power, such as the ability to inject documents.

6.2.3 Practical Attacks against the State-of-the-Art Schemes

A natural question to ask is "*does any of the attacks above work on the state-of-the-art schemes*?". The answer to that question is unfortunately "No". All of the attacks above, except the OyaKer21 attack [143], only work on unperturbed (or *almost* unperturbed) access-pattern leakage where the documents are naively encrypted and retrieved. For the state-of-the-art schemes, even if we assume that the documents are naively encrypted, the access-pattern leakage will be noisy, as the inverted index has built-in noise (e.g. padding keyword response volume to the maximum). Moreover, these attacks are known-data attacks which is hard to justify in practice.

On the other hand, the OyaKer21 attack [143] requires search-pattern leakage (if two queries are the same) and background knowledge on query frequency (how often each query is issued) to achieve a meaningful attack. As background knowledge on query frequency is hard to find in the real world, we believe that the attack has limited impact on the security of the targeted schemes.

It is therefore an open question whether the state-of-the-art schemes with a naively encrypted document array are secure to practical access-pattern leakage attacks. In this chapter, we answer this in the negative. We design an attack with the following characteristics:

- is query reconstruction in nature and can be extended to database reconstruction,
- works in inference mode,
- requires minimal distributional auxiliary information,
- is effective against the state-of-the-art schemes [98, 149] with a naively encrypted document array.

We also show that our attack is flexible and can be adapted to attack other schemes [41] with heavily perturbed access-pattern leakage from the inverted index directly. Table 6.2 gives a comparison of our attack and previous attacks.

6.2.4 Overview of Our Attacks

Our query reconstruction attacks in this chapter work as follows. Given a scheme, we begin by identifying its distribution of co-occurrence leakage. This allows us to compute

[3] The authors claimed that their attacks exploit search pattern and access pattern leakage, but the only form of access-pattern leakage they used is volume leakage.

Attack	Attack Assumption	Leakage Exploited	Additional Requirements	Perturbed Leakage?
IKK [94]	Known-data	Access pattern	Known queries	No
Count [33]	Known-data	Access pattern	Known queries*	No
BKM20 [17]	Known-data	Access pattern	–	No
Graph Matching Attacks [155]	Inference	Access pattern	–	Yes**
SAP [143]***	Inference	Search pattern and response length	Query frequency pattern	Yes
This chapter	Inference	Access pattern	–	Yes

Table 6.2: Comparison of existing passive and persistent query reconstruction attacks exploiting co-occurrence and/or access pattern leakage. * Count attack does not need known queries if the entire database is known by the attacker; known queries are only helpful when only part of the database is known by the attacker. ** The attack targets [92, 119] in the graph matching attacks [155] have weakly perturbed leakage. *** The SAP attack makes strong assumptions on the availability of auxiliary information about query frequency patterns.

the *likelihood score* for every assignment of queries to keywords given the observed co-occurrence and auxiliary data. To find the best assignment, we simply use simulated annealing [2]. Technical details of our attacks can be found in Section 6.4.

We verify the effectiveness of our attacks against the target constructions [41, 98, 149] using the Enron email corpus [194]. Our attacks are run in inference mode – the dataset is randomly split into two halves, one half of it is used as the target dataset whereas a portion of the other half is used as auxiliary information. We investigate the effect of different factors on our attacks, including:

- the number of observed queries,
- the choices of security parameters of the target constructions,
- the size of the keyword universe,
- the level of noise in auxiliary information, and
- the use of stemming in the pre-processing of the dataset.

Our experiments show that in most scenarios, our attacks manage anywhere from 50% to 80% query recovery rate, indicating that the target schemes are vulnerable to query reconstruction attacks. Details of our experiments can be found in Section 6.5 and a discussion of the implications of our attacks is presented in Section 6.6.

6.3 Formal Description of Access-pattern Leakage Attacks

In this section, we give a formal description of access-pattern leakage attacks on searchable encryptions. The description focuses on a subset of access-pattern leakage known as co-occurrence pattern leakage (see Section 4.5).

Access-pattern Leakage. We are now ready to formally describe the class of leakage which we are targeting in this chapter. The class of leakage, known as access-pattern leakage refers to the information leakage associated to retrieval of documents/values. For instance, in a naive searchable encryption scheme (see Algorithm 4.4 and 4.5), the **Srch** protocol may return the exact set of encrypted documents matching the queried keyword, hence, leaking the information that the given set of documents contains the queried keyword.

In a query reconstruction attack using access-pattern leakage, we assume that the underlying scheme Σ leaks some form of access pattern from the queries and the goal of the adversary is to recover the keywords associated to the queries. There can be other leakage from scheme Σ, such as search-pattern leakage, but it is not used in our attacks so it is omitted from the description of leakage. For example, if scheme Σ leaks the original access pattern, we can write the leakage of a query $\mathbf{q}$ on database $\mathbf{DB}$ as

$$\mathcal{L}_{\mathbf{Srch}}(\mathbf{q}, \mathbf{DB}) = \{i \mid kw(\mathbf{q}) \in kw(doc_i), (doc_i, kw(doc_i)) \in \mathbf{DB}\},$$

where $kw(\mathbf{q})$ denotes the keyword associated to query $\mathbf{q}$. We note that although the leakage here is represented by the document identifiers, it is equivalent to an encrypted document based representation. We choose the earlier as the representation is more compact.

Co-occurrence Pattern Leakage. Access pattern leakage from different queries can be represented equivalently as a matrix, known as co-occurrence pattern leakage. Consider a database $\mathbf{DB}$ where

$$\begin{aligned}\mathbf{DB} = \{&(doc_1, \{kw_1, kw_2, kw_3\})\\ &(doc_2, \{kw_1, kw_2\})\\ &(doc_3, \{kw_3\})\}.\end{aligned}$$

Let $\mathbf{q}_i$ be a query on keyword kw_i. If the original access pattern is leaked, we know that

$$\begin{aligned}\mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_1, \mathbf{DB}) &= \{1, 2\},\\ \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_2, \mathbf{DB}) &= \{1, 2\},\\ \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_3, \mathbf{DB}) &= \{1, 3\}.\end{aligned}$$

This allows us to take intersections between the leakages as follows.

$$\begin{aligned}\mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_1, \mathbf{DB}) \cap \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_2, \mathbf{DB}) &= \{1, 2\},\\ \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_1, \mathbf{DB}) \cap \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_3, \mathbf{DB}) &= \{1\},\\ \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_2, \mathbf{DB}) \cap \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_3, \mathbf{DB}) &= \{1\}.\end{aligned}$$

The cardinality of the intersections can be very useful in an attack. For example, the co-occurrence pattern leakage of the database above can be represented as a *co-occurrence matrix* $\bar{M}$:

$$\bar{M}(\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3; \mathbf{DB}) = \begin{bmatrix} 2 & 2 & 1 \\ 2 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix},$$

where the i,j-th entry of the matrix is $\left|\mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_i, \mathbf{DB}) \cap \mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_j, \mathbf{DB})\right|$. If we know the underlying database perfectly, we can re-identify $\mathbf{q}_3$ as a query on kw_3 as it is the only keyword that only shares one document with other keywords. This qualifies as a query reconstruction attack.

We note that co-occurrence pattern leakage contains strictly less information than the original access pattern leakage. However, co-occurrence pattern leakage is often sufficient in practice so it is used instead of the full leakage.

There are three complications to the representation of co-occurrence pattern leakage in practice. Firstly, the schemes we consider usually leak search pattern too. That is, if $kw(\mathbf{q}_i) = kw(\mathbf{q}_j)$, the attacker knows that the two queries are for the same keyword. In terms of co-occurrence pattern leakage, we use only one of the queries in the representation to simplify the problem. Secondly, the queries are unordered in practice. That means there is no standard representation of the leakage in terms of the known keywords. We use the convention that the i-th row and column of the co-occurrence matrix corresponds to the i-th non-repeating query in our representation. Finally, not all schemes leak the original access pattern and some schemes may even be randomised. In those cases, we need to use a suitable representation of the co-occurrence information, which may differ from what we have described above.

Auxiliary Information. Similar to a co-occurrence matrix, the auxiliary information the attacker receives can be represented as a matrix M. The matrix is indexed by the known keywords and typically contains full information on all keywords. In stronger attacks, M is assumed to be noisy in the sense that it is not generated directly from the target document collection. Instead, an auxiliary dataset is used for the purpose.

Let $\mathbf{DB} = \{(doc_i, kw(doc_i))\}$ be an auxiliary document collection with keywords $\{kw_1, \ldots, kw_n\}$. In our attack, the i,j-th entry of M represents the empirical probability (derived from the auxiliary data $\mathbf{DB}$) of seeing kw_i and kw_j together in a document. It is computed as:

$$M_{i,j} = \frac{|\{doc_i \mid kw_i \in kw(doc_i) \wedge kw_j \in kw(doc_i)\}|}{|\mathbf{DB}|},$$

where $|\mathbf{DB}|$ denotes the number of documents in the collection.

Attack Setting. Let Σ be a searchable encryption scheme. Our attack exploits its co-occurrence pattern leakage $\bar{M}(\cdot; \mathbf{DB})$ to recover the queries. The attack can be formally described as follows. Let queries $\mathbf{q}_1, \ldots, \mathbf{q}_l$ be a sequence of queries on the document collection, so the attacker observes co-occurrence pattern leakage $\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})$. Suppose that the attacker has access to some auxiliary information M. *The goal of the attacker is to recover $kw(\boldsymbol{q}_i)$ after observing the co-occurrence pattern leakage $\bar{M}$ and knowing auxiliary information M.*

6.4 New Access-pattern Leakage Attacks

In this section, we show how system-level leakage from the state-of-the-art searchable encryption schemes [98, 149] can be modelled and attacked. We also show how our attack can be adapted to another scheme [41] with heavily perturbed co-occurrence leakage.

There are four main steps in each attack:

1. Identify the leakage function.
2. Derive a mathematical model of the co-occurrence leakage.
3. Design an attack for the co-occurrence leakage.
4. Experimentally evaluate the attack in various settings.

The first three steps of the attacks are presented in Section 6.4.1 through 6.4.3 and the last step of the attacks is presented in Section 6.5. The mathematical concepts (i.e. distributions and parameter estimations) can be found in Section 2.1.6.

The derivation of the attacks relies heavily on probability theory and statistics. See Section 2.1 for a refresher.

6.4.1 Attack Targets and Co-occurrence Pattern Leakages

Volume-hiding EMMs via Pseudo-random Transform. As a potential countermeasure to leakage-abuse attacks, Kamara and Moataz [98] introduced the concept of *volume-hiding* encrypted multi-maps (EMMs) that hide response length patterns (and exact access patterns) while providing better search performance compared to naïve (or worst-case) padding. They proposed the first construction of volume-hiding EMMs based on an obfuscation mechanism called *pseudo-random transform*. Their idea is to pad or truncate the query response lengths of queries on a multi-map with a pseudo-random function as follows. Let key be a key for the multi-map and $F_{\mathsf{sk}}(\cdot)$ be a pseudo-random function with key sk. The client computes: $n'_{\mathsf{key}} = \lambda + F_{\mathsf{sk}}(\mathsf{key}||n_{\mathsf{key}})$

$$n'_{\mathsf{key}} = \lambda + F_{\mathsf{sk}}(\mathsf{key}||n_{\mathsf{key}})$$

as the new query response length, where λ is a free parameter which the client can choose and n_{key} is the original query response length.

These new query response lengths are used to build a multi-map on *document identifiers* as follows:

- If $n_{\mathsf{key}} \leq n'_{\mathsf{key}}$, add $\perp$ symbols in the multi-map on key key before encryption.
- If $n_{\mathsf{key}} > n'_{\mathsf{key}}$, truncate the multi-map on keyword key to the first n'_{key} entries.

The multi-map is then encrypted with an underlying encrypted multi-map scheme and uploaded to the server. As a searchable encryption scheme, we assume that **PRT-EMM** is used to encrypt inverted index and a naive encrypted document array is used; when a query is issued, the client retrieves a set of document identifiers from the encrypted inverted index, and all of the document identifiers are used in the document retrieval phase.

We note that the original construction pads query responses with $\perp$ symbols if the real query response length is shorter. If the $\perp$ symbols are ignored in actual document retrieval, the attacker will be able to learn the true query response lengths when there

is padding. This leakage may allow for simpler attacks which use frequency information only. However, in our attack, we assume the $\perp$ symbols are replaced by randomly picked indices, so the true query response lengths are not leaked. We show that even in this setting, the scheme is vulnerable to access-pattern leakage attacks.

Newer Constructions of Volume-hiding EMMs. Recently, Patel *et al.* proposed two volume-hiding EMM constructions in [149]. Both of the constructions use Cuckoo hashing [145] as the underlying data structure. Just like **PRT-EMM** [98], we assume that the constructions are used to encrypt inverted index only.

The two schemes proposed by the authors are only different in terms of the padding mechanism on the query response lengths. The first scheme uses full padding, meaning that all query response lengths are padded to the maximum query response length. In terms of the hash table, this is done by querying additional addresses deterministically (generated by a pseudo-random function) for each key.

The second scheme uses differentially-private volume hiding as opposed to full padding. Let $2n_{\mathsf{key}}$ be the true query response length of a query on key key, where 2 comes from the fact that Cuckoo hashing uses two hash tables. Then the scheme pads the query response length to $2n_{\mathsf{key}} + n^* + \mathbf{Lap}_{\mathsf{sk}}(2/\epsilon)$, where n^* is a parameter set by the client to offset the query response length in case the latter random variable is negative, and $\mathbf{Lap}_{\mathsf{sk}}(\cdot)$ is a Laplace distribution with secret key sk as the seed.

Searchable Encryption with Differentially-Private Access Pattern. Chen *et al.* [41] proposed using differential privacy as a means to prevent leakage-abuse attacks that exploit access pattern leakages. They proposed a searchable encryption scheme where a differential privacy mechanism is used to obfuscate the plaintext database before building an encrypted database, such that a slight change in the real access pattern does not affect the obfuscated access pattern significantly. There are two key ingredients in their construction. Firstly, they used an erasure code [63] to split every document into m *shards*, each with size $\frac{1}{k}$ of the original document. The erasure code has the property that any k shards of a document can be used to reconstruct the original document. The client then picks two probabilities p and q, and does the following to each shard:

1. For any keyword that is originally in the shard, remove the keyword with probability $1 - p$.

2. For any keyword that is originally not in the shard, add it to the shard with probability q.

We refer to this scheme as **DPAP-SE**.

Intuitively, a smaller p and a larger q means more distortion to the co-occurrence information, and hence more "secure" against access-pattern leakage attacks. However, to ensure that enough shards are returned with high probability, p has to be decently large. Similarly, to control the communication overhead, q has to be small. In terms of the effect of the countermeasure on the co-occurrence pattern leakage, it transforms query response lengths into noisy keyword response lengths on the shards; co-occurrence counts are now leaked as noisy co-occurrences on the shards.

6.4.2 Mathematical Derivations of the Distributions of the Co-occurrence Matrices

In this section, we give the mathematical derivations of the distributions of the co-occurrence matrices. We note that leakage referred in the literature usually mean *realisation* as opposed to *distribution*. Here, we derive the latter so that we can compute likelihood scores for *realisations*.

Derivation for **PRT-EMM** [98]. Recall that in **PRT-EMM**, the query response lengths are padded or truncated according to:

$$n'_{\mathsf{key}} = \lambda + F_{\mathsf{sk}}(\mathsf{key}||n_{\mathsf{key}}).$$

Let **DB** be a multi-map and $\mathbf{q}_1, \ldots, \mathbf{q}_l$ be non-repeating search queries with associated keys $\mathsf{key}_1, \ldots, \mathsf{key}_l$ on **DB** encrypted with **PRT-EMM**. We abuse the notation $\mathsf{key}(\mathbf{q}_i)$ to mean the key associated to $\mathbf{q}_i$. By denoting the maximum value of the PRF F as $|F|$, the diagonal entries of the co-occurrence matrix can be expressed as:

$$\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,i} \sim \lambda + \mathbf{Uniform}(0, |F|),$$

where $\mathbf{Uniform}(\cdot)$ is a uniform distribution.

There are three cases to be considered for the off-diagonal entries of the co-occurrence matrix. Without loss of generality, let the keys in concern be keys key_i and key_j. In the first case, both of the query response lengths associated to the keys are larger than the true query response lengths. This corresponds to $n'_{\mathsf{key}_i} - |\mathbf{DB}(\mathsf{key}_i)|$ random document retrievals for queries on key key_i and $n'_{\mathsf{key}_j} - \left|\mathbf{DB}(\mathsf{key}_j)\right|$ random document retrievals for queries on key key_j. These random document retrievals can create additional co-occurrence counts among themselves or with the real document retrievals. The co-occurrence counts in this case can be approximated by:

$$\begin{aligned}
&\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,j} \\
\sim &\left|\mathbf{DB}(\mathsf{key}_i, \mathsf{key}_j)\right| \\
+&\mathbf{Hypergeometric}\left(n'_{\mathsf{key}_i} - |\mathbf{DB}(\mathsf{key}_i)|, |\mathbf{DB}|, n'_{\mathsf{key}_j}\right) \\
+&\mathbf{Hypergeometric}\left(n'_{\mathsf{key}_j} - \left|\mathbf{DB}(\mathsf{key}_j)\right|, |\mathbf{DB}|, n'_{\mathsf{key}_i}\right),
\end{aligned}$$

where $\mathbf{Hypergeometric}(n, N, K)$ denotes a hypergeometric distribution which makes n draws without replacement, from a population of size N that contains exactly K objects with the desired feature.

In the second case, one of the query response lengths is truncated and the other one is padded. Without loss of generality, let key key_i be the truncated key and key key_j be the padded key. Then, the co-occurrence count associated to keys key_i and key_j can be modelled as a process where the co-occurrence count is first reduced by the truncation and then increased by the padding. Its distribution is given below:

$$x \sim \mathbf{Hypergeometric}\Big(n'_{\mathsf{key}_i}, |\mathbf{DB}(\mathsf{key}_i)|, \left|\mathbf{DB}(\mathsf{key}_i, \mathsf{key}_j)\right|\Big),$$

$$\begin{aligned}
\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,j} \sim &x + \mathbf{Hypergeometric}\Big(n'_{\mathsf{key}_j} - \left|\mathbf{DB}(\mathsf{key}_j)\right|, \\
&|\mathbf{DB}|, n'_{\mathsf{key}_i} - x\Big).
\end{aligned}$$

Finally, in the last case, both of the query response lengths are truncated. Similar to above, the distribution of the co-occurrence count associated to keys key_i and key_j can be expressed as:

$$x \sim \mathbf{Hypergeometric}\Big(n'_{\mathsf{key}_i}, |\mathbf{DB}(\mathsf{key}_i)|, \big|\mathbf{DB}(\mathsf{key}_i, \mathsf{key}_j)\big|\Big),$$

$$\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,j} \sim \mathbf{Hypergeometric}\Big(n'_{\mathsf{key}_j}, \big|\mathbf{DB}(\mathsf{key}_j)\big|, x\Big).$$

Derivation for new volume-hiding multi-maps in [149]. The volume-hiding multi-maps in [149] are special cases of **PRT-EMM** [98], where the query response lengths are either padded to the maximum query response length or ones that are larger than the true query response lengths. Specifically, for the full padding version (**PRT-EMM**),

$$\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,i} \sim 2 \max_{\mathsf{key}} |\mathbf{DB}(\mathsf{key})|.$$

And for the differentially-private version (**DP-EMM**),

$$\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,i} \sim 2\,|\mathbf{DB}(\mathsf{key})| + n^* + \mathbf{Lap}(2/\epsilon),$$

where n^* is a fixed constant to offset the query response length in case the latter random variable is negative.

For the co-occurrence counts, we get:

$$\begin{aligned}
&\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,j} \\
\sim &\big|\mathbf{DB}(\mathsf{key}_i, \mathsf{key}_j)\big| \\
+&\mathbf{Hypergeometric}\left(n'_{\mathsf{key}_i} - |\mathbf{DB}(\mathsf{key}_i)|, |\mathbf{DB}|, n'_{\mathsf{key}_j}\right) \\
+&\mathbf{Hypergeometric}\left(n'_{\mathsf{key}_j} - \big|\mathbf{DB}(\mathsf{key}_j)\big|, |\mathbf{DB}|, n'_{\mathsf{key}_i}\right),
\end{aligned}$$

where n'_{key_i} and n'_{key_j} are the padded query response lengths for keyword kw_i and kw_j respectively.

Derivation for **DPAP-SE** [41]. Let **DB** be a database and $\mathbf{q}_1, \ldots, \mathbf{q}_l$ be non-repeating search queries with associated keywords $kw_1, \ldots, kw_l$ on **DB** encrypted with the searchable encryption scheme above [41]. The diagonal entries of the co-occurrence matrix $\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})$, i.e. the query response volumes, represent the numbers of shards retrieved by the client. For a particular query $\mathbf{q}_i$, the number of shards retrieved is determined by:

- The number of shards which contain keyword kw_i before the pre-processing step, and the keyword is not removed from them.
- The number of shards which do not contain keyword kw_i before the pre-processing step, but the keyword is added to them.

Formally, the diagonal entries of the co-occurrence matrix can be expressed in terms of the true query response lengths as:

$$\begin{aligned}
\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,i} \sim\ &\mathbf{Bin}(m \cdot |\mathbf{DB}(kw_i)|, p) \\
&+ \mathbf{Bin}(m \cdot |\mathbf{DB}| - m \cdot |\mathbf{DB}(kw_i)|, q),
\end{aligned}$$

where m comes from splitting the documents into shards, $\mathbf{DB}(kw_i)$ denotes the set of documents containing keyword kw_i associated to query $\mathbf{q}_i$, $|\mathbf{DB}|$ denotes the number of documents in database $\mathbf{DB}$, and $\mathbf{Bin}(\cdot)$ denotes a binomial distribution.

For the off-diagonal entries of the co-occurrence matrix, assume without loss of generality that the keywords in concern are kw_i and kw_j. The co-occurrence count for keywords kw_i and kw_j can increase if:

- A shard contains one of the keywords, say kw_i, and the keyword is not removed by the scheme. At the same time, the other keyword, kw_j in this case, is added to the shard.
- A shard contains none of the keywords, and both of the keywords are added to the shard.

On the other hand, the co-occurrence count for keywords kw_i and kw_j can decrease if a shard contains both of the keywords and at least one of the keywords is removed.

The actual distribution of the off-diagonal entries of the co-occurrence matrix is complicated due to dependencies. Here, we approximate the off-diagonal entries of the co-occurrence matrix by assuming independence:

$$\begin{aligned}
&\bar{M}(\mathbf{q}_1, \ldots, \mathbf{q}_l; \mathbf{DB})_{i,j} \\
\sim&\mathbf{Bin}(m \cdot |\mathbf{DB}(kw_i, kw_j)|, p^2) \\
+&\mathbf{Bin}\left(m \cdot (|\mathbf{DB}| - |\mathbf{DB}(kw_i)| - |\mathbf{DB}(kw_j)| + |\mathbf{DB}(kw_i, kw_j)|), q^2\right) \\
+&\mathbf{Bin}(m \cdot |\mathbf{DB}(kw_i)| - |\mathbf{DB}kw_i, kw_j|, pq) \\
+&\mathbf{Bin}(m \cdot |\mathbf{DB}(kw_j)| - |\mathbf{DB}kw_i, kw_j|, pq).
\end{aligned}$$

6.4.3 Attack Model

6.4.3.1 Attack Overview

Given the observed co-occurrence matrix $\bar{M}$ and auxiliary co-occurrence matrix M (which may not have the same dimensions), the goal of the adversary is to find an assignment P between the queries and the keywords such that the observed co-occurrence matrix fits the auxiliary information. In our formulation, the diagonal entries in the observed co-occurrence matrix are the query response lengths, and the off-diagonal entries are the number of documents accessed by two queries at the same time; the diagonal entries in the auxiliary co-occurrence matrix are probabilities such that a document contains the given keywords. For simplicity, we assume identical and independent distribution of the keywords, meaning that the true query response lengths can be modelled as binomial distributions. Furthermore, we assume that the off-diagonal entries in the auxiliary co-occurrence matrix specify the distribution of co-occurrences of keywords, and we use multinomial distributions to model the distributions. Given that there is randomness in the generation of leakage, we propose to use a likelihood function $\mathbf{L}\left[P \mid \bar{M}, M\right]$ to measure the fitness of the data. As the search space for the assignment is huge, a brute-force approach is impractical. We propose to use simulated annealing [2] to search for the most likely assignment. In the next section, we explain how simulated annealing works and outline the subroutines of the algorithm.

6.4.3.2 Simulated Annealing

We give a brief overview of simulated annealing [2] in this section. Simulated annealing is a probabilistic technique for searching for the global optimum of a given function. It is very similar to a greedy search algorithm – randomize the input of the function, in our case, that is the assignment P, recompute the score, and if the score is larger than before, the assignment is kept as the new solution, and it is discarded otherwise – except that a worse solution is accepted in simulated annealing if it is not *too bad*. This is to prevent the algorithm from sticking in a local optimum. More concretely, simulated annealing uses a *temperature* T which decreases per iteration and the differences between the current score of the target function and the previous best score maintained by the algorithm to compute an acceptance probability p, and with probability p the new solution is accepted. This probability is 1 if the new score is higher than the previous best, and less than 1 otherwise. For the same difference in the scores, a lower temperature T leads to a lower acceptance probability, which means simulated annealing acts more and more like a greedy search algorithm as the iterations go on.

Formally, simulated annealing consists of five subroutines, namely a function **InitPerm** to generate an initial assignment, a cooling scheme **Cooling**, a neighbourhood generation algorithm **Neighbour**, a function **Score** to compute the score and a function **AccptProb** to compute the acceptance probability. The syntax of the subroutines are defined below:

- **InitPerm**: takes as input an observed co-occurrence matrix $\bar{M}$ and a background co-occurrence matrix M, and outputs an assignment P.
- **Cooling**: takes as input a temperature T and the current iteration number i and outputs a new temperature T'.
- **Neighbour**: takes as input a assignment P, an observed co-occurrence matrix $\bar{M}$ and a background co-occurrence matrix M, and output a new assignment P'.
- **Score**: takes as input an observed co-occurrence matrix $\bar{M}$, a background co-occurrence matrix M and an assignment P, and output a score.
- **AccptProb**: takes as input a temperature T, a previous best score s and the new score s', and output a probability.

We are now ready to give an overview of simulated annealing. The algorithm begins with an initial temperature T_0 and a random assignment P. An initial score s is computed on this assignment P. Then, the algorithm computes a new temperature $T \leftarrow \textbf{Cooling}(T_0, 1)$, find a new assignment P' using the neighbourhood function $\textbf{Neighbour}(\cdot)$, and compute a new score s' with the score function $\textbf{Score}(\cdot)$. An acceptance probability is computed as $p \leftarrow \textbf{AccptProb}(T, s, s')$. A random number between 0 and 1 is generated and if the random number is less or equal to p, the new solution s' is accepted by the algorithm and kept as the new optimum solution. This process is repeated until the maximum number of iteration is reached. A detailed pseudocode of simulated annealing is presented in Algorithm 6.1.

Algorithm 6.1 Simulated Annealing

procedure ATTACK($\bar{M}, M, T_0, i_{\max}$)
  $P \leftarrow$ **InitPerm**($\bar{M}, M$)
  $T \leftarrow T_0$
  $s \leftarrow$ **Score**($\bar{M}, M, P$)
  for $i \leftarrow 1, \ldots, i_{\max}$ **do**
    $T \leftarrow$ **Cooling**(T, i)
    $P' \leftarrow$ **Neighbour**($P, \bar{M}, M$)
    $s' \leftarrow$ **Score**($\bar{M}, M, P'$)
    if **AccptProb**(T, s, s') $>$ `rand`$(0, 1)$ **then**
      $P \leftarrow P'$
      $s \leftarrow s'$
  return P

6.4.3.3 Application of Simulated Annealing to Query Reconstruction Attacks using Co-occurrence Pattern Leakage

In this section, we specify the subroutines we used in our attacks. We used $T' \leftarrow 0.995T$ as our cooling scheme **Cooling**$(\cdot)$ and $p \leftarrow \exp(-\frac{s-s'}{T})$ as our function **AccptProb**$(\cdot)$ to compute the acceptance probability for all three leakages we have considered. These choices are commonplace for simulated annealing problems [55, 168]. We used the likelihood functions as the score functions **Score**$(\cdot)$, and detailed derivations of them are shown in Section 6.4.4. We find the choices of **InitPerm**$(\cdot)$ and **Neighbour**$(\cdot)$ have a significant impact on the performance and effectiveness of our attacks. The subroutines presented in this section are the most effective variants we have found.

INITIAL ASSIGNMENT FINDING SUBROUTINE **InitPerm**$(\cdot)$. An initial assignment finding subroutine **InitPerm**$(\cdot)$ is an efficient algorithm for guessing keywords/keys of the queries, so as to provide a starting point for the more expensive iterative steps later. For our attacks, only the query response lengths are used to avoid expensive computations. We observe that although the observed query response lengths are different from the true query response lengths for all of the schemes we target, these two are related. In particular, for **DP-EMM** [149] and **DPAP-SE** [41], we can compute the expected observed query response lengths from the query response lengths in the background co-occurrence matrix, and matching the queries to the keywords in the background co-occurrence matrix as good as we can. For **PRT-EMM** [98] and **FP-EMM** [149], the observed keyword frequencies are independent from the true keyword frequencies.

NEIGHBOURHOOD GENERATION SUBROUTINE **Neighbour**$(\cdot)$. A neighbourhood generation subroutine generates new assignments for the attack. Although a uniformly randomly picked assignment works all the times, it may not be the most efficient choice. In particular, for **DP-EMM** [149] and **DPAP-EMM** [41], we know that if an observed query response length is too far from the expectation, the assignment is very unlikely, and can be safely discarded. This means the neighbourhood generation subroutines for the attacks on these two schemes can make use of this, and output a new assignment only if it is sound. The pseudocodes for these subroutines can be found in Algorithm 6.2 and 6.3.

We note that these neighbourhood generation subroutines may prevent some correct assignments in the output of the attack if their observed query response lengths are too

Algorithm 6.2 Neighbourhood Generation Algorithm for **DP-EMM** [149]

procedure Neighbour$(P, \bar{M}, M)$
  $i \xleftarrow{\$} \{1, \ldots, |kw(\mathbf{DB})|\}$
  $j \xleftarrow{\$} \{1, \ldots, |kw(\mathbf{DB})|\}$
  $b_0 \leftarrow NM_{j,j} - 1.96NM_{j,j}(1 - M_{j,j}) - 1.96/\epsilon$
  $b_1 \leftarrow NM_{j,j} + 1.96NM_{j,j}(1 - M_{j,j}) + 1.96/\epsilon$
  if there exists k such that $P(k) = j$ **then**
    $b_2 \leftarrow NM_{P(i),P(i)} - 1.96NM_{P(i),P(i)}(1 - M_{P(i),P(i)}) - 1.96/\epsilon$
    $b_3 \leftarrow NM_{P(i),P(i)} + 1.96NM_{P(i),P(i)}(1 - M_{P(i),P(i)}) + 1.96/\epsilon$
  /* *Check the condition with k only if it exits* */
  while $\neg(b_0 < \bar{M}i, i < b_1) \vee \neg(b_2 < \bar{M}_{k,k} < b_3)$ **do**
    Resample i, j, k
  $P' \leftarrow P$
  $P'(i) \leftarrow j$
  if there exists k such that $P(k) = j$ **then**
    $P'(k) \leftarrow P(i)$
  return P'

Algorithm 6.3 Neighbourhood Generation Algorithm for **DPAP-SE** [41]

procedure Neighbour$(P, \bar{M}, M)$
  $i \xleftarrow{\$} \{1, \ldots, |kw(\mathbf{DB})|\}$
  $j \xleftarrow{\$} \{1, \ldots, |kw(\mathbf{DB})|\}$
  $b_0 \leftarrow kpNM_{j,j} + kqN(1 - M_{j,j}) - 1.96kNM_{j,j}(1 - M_{j,j}) - 1.96kN(p + q)$
  $b_1 \leftarrow kpNM_{j,j} + kqN(1 - M_{j,j}) + 1.96kNM_{j,j}(1 - M_{j,j}) + 1.96kN(p + q)$
  if there exists k such that $P(k) = j$ **then**
    $b_2 \leftarrow kpNM_{P(i),P(i)} + kqN(1 - M_{j,j}) - 1.96kNM_{P(i),P(i)}(1 - M_{P(i),P(i)}) - 1.96kN(p + q)$
    $b_3 \leftarrow kpNM_{P(i),P(i)} + kqN(1 - M_{j,j}) - 1.96kNM_{P(i),P(i)}(1 - M_{P(i),P(i)}) - 1.96kN(p + q)$
  /* *Check the condition with k only if it exits* */
  while $\neg(b_0 < \bar{M}i, i < b_1) \vee \neg(b_2 < \bar{M}_{k,k} < b_3)$ **do**
    Resample i, j, k
  $P' \leftarrow P$
  $P'(i) \leftarrow j$
  if there exists k such that $P(k) = j$ **then**
    $P'(k) \leftarrow P(i)$
  return P'

far from the expected query response lengths. By relaxing the bounds, we can make the chance of that happening arbitrarily small. However, the algorithm is going to be less efficient as more iterations are required for a convergence. Hence, we see our choice of bounds as a trade-off between query recovery rate and attack efficiency.

For **PRT-EMM** [98] and **FP-EMM** [149], we have to use uniformly randomly picked assignments as the observed query response lengths are independent from the true query response lengths. The pseudocode of this neighbourhood generation subroutine can be found in Algorithm 6.4.

Algorithm 6.4 Neighbourhood Generation Algorithm for **PRT-EMM** [98] and **FP-EMM** [149]

procedure Neighbour$(P, \bar{M}, M)$
  $i \xleftarrow{\$} \{1, \ldots, |kw(\mathbf{DB})|\}$
  $j \xleftarrow{\$} \{1, \ldots, |kw(\mathbf{DB})|\}$
  $P' \leftarrow P$
  $P'(i) \leftarrow j$
  if there exists k such that $P(k) = j$ **then**
    $P'(k) \leftarrow P(i)$
  return P'

6.4.4 Mathematical Derivations of the Likelihood Functions

The likelihood function $\mathbf{L}\left[P \mid \bar{M}, M\right]$ can be written as follows:

$$\begin{aligned}
&\mathbf{L}\left[P \mid \bar{M}, M\right] \\
=&\mathsf{Pr}\left[\bar{M}, M \mid P\right] \\
=&\sum_{M' \in \mathcal{N}^{N \times N}} \mathsf{Pr}\left[\bar{M}, M, M' \mid P\right] \\
=&\sum_{M' \in \mathcal{N}^{N \times N}} \mathsf{Pr}\left[\bar{M} \mid M, M', P\right] \mathsf{Pr}\left[M' \mid M, P\right] \\
=&\sum_{M' \in \mathcal{N}^{N \times N}} \mathsf{Pr}\left[\bar{M} \mid M', P\right] \mathsf{Pr}\left[M' \mid M\right],
\end{aligned}$$

where N is the number of keywords and $\mathcal{N}^{N \times N}$ is all N by N natural number valued matrices. In the third line of the equation, we used the law of total probability to turn the likelihood into a summation over all possible real co-occurrence matrices. The lines after break the probability into a sum of products of two probabilities. The first probability $\mathsf{Pr}\left[\bar{M}, M' \mid P\right]$ is the probability that $\bar{M}$ is the observed co-occurrence matrix and M' is the real co-occurrence matrix given P is the permutation. The second probability is the probability of getting M' as the real observed co-occurrence matrix knowing that M is the auxiliary co-occurrence matrix.

We assume the same structure of the auxiliary co-occurrence matrix M for all of our leakage functions so its derivation is shared by all three leakage functions. We note that only some of the real co-occurrence matrices generate a non-zero likelihood, as the sum of off-diagonal entries of a row must be less or equal to the diagonal entry for correctness. By writing a row of a matrix M without the i-th entry as $M_{i,\cdot}$, for those real co-occurrence matrices, we can derive the probability as:

$$\begin{aligned}
&\mathsf{Pr}\left[M' \mid M\right] \\
=&\sum_{i} \mathsf{Pr}\left[M'_{i,i} \mid M_{i,i}\right] \mathsf{Pr}\left[M'_{i,\cdot} \mid M'_{i,i}, M_{i,\cdot}\right].
\end{aligned}$$

In the second line, the first term is the probability of getting $M'_{i,i}$ documents containing keyword kw_i, and the second term is the probability of observing the off-diagonal co-occurrence counts.

Derivation for **PRT-EMM**. For **PRT-EMM** [98], query response lengths may be truncated by a random amount. This means that based on the query response length in

the auxiliary co-occurrence matrix M' and that in the observed co-occurrence matrix, an attacker can estimate how many documents in the off-diagonal entries are expected to be removed. For observed co-occurrence count between keywords kw_j and kw_j where $i \neq j$, the real process can be modelled as a sequential application of two hypergeometric distributions on the real co-occurrence count.

$$\begin{aligned}
&\mathsf{Pr}\left[\bar{M}, M' \mid P\right] \\
=&\prod i < j \mathsf{Pr}\left[\bar{M}_{i,j}, M' \mid P\right] \\
=&\prod i < j \sum_k \mathsf{Pr}\left[\textbf{Hypergeometric}\left(M'_{P(i),P(i)}, M'_{P(i),P(j)}, \bar{M}_{i,i}\right) = k\right] \\
&\mathsf{Pr}\left[\textbf{Hypergeometric}\left(M'_{P(j),P(j)}, k, \bar{M}_{j,j}\right) = \bar{M}_{i,j}\right].
\end{aligned}$$

Derivation for **FP-EMM** [149]. To simplify the first term of the likelihood decomposition, we assume independence of the entries in the observed co-occurrence matrix. Without loss of generality, we assume that all query response lengths are padded to m. This means we can express the probability as:

$$\begin{aligned}
&\mathsf{Pr}\left[\bar{M}, M' \mid P\right] \\
=&\prod i < j \mathsf{Pr}\left[\bar{M}_{i,j}, M'_{P(i),P(j)} \mid P\right] \\
=&\prod i < j \mathbf{Pr}\Big[\textbf{Hypergeometric}\big(2N, 2m - 2M'_{P(i),P(i)}, \\
&2m - 2M'_{P(j),P(j)}\big) = \bar{M}_{i,j} - M'_{P(i),P(j)}\Big].
\end{aligned}$$

Derivation for **DP-EMM** [149]. The first term of the likelihood decomposition for differentially private volume-hiding EMMs [149] is similar to that of the full padding version, except that the query response lengths are padded according to a Laplacian distribution as opposed to padding to the maximum query response length. Let n^* be the constant to offset the Laplacian random variable $\textbf{Lap}(2/\epsilon)$, the first term of the likelihood decomposition can be expressed as:

$$\begin{aligned}
&\mathsf{Pr}\left[\bar{M}, M' \mid P\right] \\
=&\prod i = j \mathsf{Pr}\left[\bar{M}, M'\right] + \sum_{i<j} \mathsf{Pr}\left[P \mid \bar{M}, M' \mid P\right] \\
=&\prod i \mathsf{Pr}\left[\bar{M}_{i,i}, M'_{P(i),P(i)} \mid P\right] + \prod i < j \mathsf{Pr}\left[\bar{M}_{i,j}, M'_{P(i),P(j)} \mid P\right] \\
=&\prod i \mathsf{Pr}\left[2M'_{P(i),P(i)} + n^* + \textbf{Lap}(2/\epsilon) = \bar{M}_{i,i}\right] \\
&+ \prod i < j \mathbf{Pr}\Big[\textbf{Hypergeometric}(2N, 2\bar{M}_{i,i} - 2M'_{P(i),P(i)}, \\
&2\bar{M}_{j,j} - 2M'_{P(j),P(j)}) = \bar{M}_{i,j} - 2M'_{P(i),P(j)}\Big].
\end{aligned}$$

Derivation for **DPAP-SE**. Recall that in **DPAP-SE** [41], the documents are split into shards and the keywords for the shards are randomized. This means that each diagonal entry of the observed co-occurrence matrix contain the counts from the real shards which have kept the keyword, and the counts from the other shards which have gained the keyword from the randomization process. Similarly, each off-diagonal entry

of the observed co-occurrence matrix contain the counts from the real shards which have kept both of the keywords, and the other counts from the other shards which have gained one of the keywords or both of them from the randomization process. Let p be the probability that a shard keeps its keywords, q be the probability that a fake keyword is introduced to a shard, and m to be the number of shards, we can express the first term in the likelihood decomposition as:

$$
\begin{aligned}
&\mathsf{Pr}\left[\bar{M}, M' \mid P\right] \\
=&\prod_{i=j} \mathsf{Pr}\left[\bar{M}, M' \mid P\right] \times \prod_{i<j} \mathsf{Pr}\left[\bar{M}, M' \mid P\right] \\
=&\prod_{i} \mathsf{Pr}\left[\bar{M}_{i,i}, M' \mid P\right] \times \prod_{i<j} \mathsf{Pr}\left[\bar{M}_{i,j}, M'_{P(i),P(j)} \mid P\right] \\
=&\prod_{i} \mathsf{Pr}\left[\mathbf{Bin}\left(mM'_{P(i),P(i)}, p\right) + \mathbf{Bin}\left(mM'_{P(j),P(j)}, q\right) = \bar{M}_{i,i}\right] \\
\times&\prod_{i<j} \mathbf{Pr}\Bigg[\mathbf{Bin}\left(mM'_{P(i),P(j)}, p^2\right) \\
+&\mathbf{Bin}\left(m\left(M'_{P(i),P(i)} - \sum_{k>1} M'_{P(i),P(k)}\right), pq\right) \\
+&\mathbf{Bin}\left(m\left(M'_{P(j),P(j)} - \sum_{k>1} M'_{P(j),P(k)}\right), pq\right) \\
+&\mathbf{Bin}\left(m\left(N - M'_{P(i),P(i)} - M'_{P(j),P(j)} + M'_{P(i),P(j)}\right), q^2\right) = \bar{M}_{i,j}\Bigg].
\end{aligned}
$$

Approximation Techniques. As it can be seen, it is computationally infeasible to sum over all possible real co-occurrence matrices. We propose to sum over all possible real co-occurrence matrices such that $\mathsf{Pr}\left[M' \mid \bar{M}\right]$ is significant. In our experiment, we used symmetric endpoints on every entry of M' such that the resultant interval covers at least 95% of the probability density function. We use Normal approximation in the first term of the likelihood decomposition for **PRT-EMM** [98] to remove the need of a convolution. To further improve the computational efficiency, we used simple rectangle rule to approximate large products, such as the convolutions in the first term of the likelihood decomposition for **DPAP-SE** [41].

Speeding up the **Score** function. The **Score** functions are by far the most computationally demanding functions of our attacks. If we just implement them naïvely, the amount of computation required in an iteration is proportional to l^2, where l is the number of non-repeating queries observed (it is also the dimension of the observed co-occurrence matrix $\bar{M}$). However, we note that the score functions in our attacks are essentially likelihood functions of the shape

$$\prod_{i \leq j} \mathsf{Pr}\left[\bar{M}_{P(i),P(j)}, M\right],$$

and the neighbourhood function **Neighbour** only changes the assignment P for one or two values. Without loss of generality, let $P(a)$ be the changed assignment. It means only the probabilities with $P(a)$ involved are changed, that is, the new likelihood

function can be written as

$$\prod_{\substack{i \leq j \\ i,j \neq a}} \mathsf{Pr}\left[\bar{M}_{P(i),P(j)}, M\right] \times \prod_{i \leq a} \mathsf{Pr}\left[\bar{M}_{P(i),P(a)}, M\right] \times \prod_{a < j} \mathsf{Pr}\left[\bar{M}_{P(a),P(j)}, M\right].$$

The terms in the first product were already computed in the previous iteration so they can be used directly. The only terms that need re-computation are in the second and third products. This reduces the amount of computation required for the **Score** function (from the second iteration onwards) to something that is proportional to l.

In our implementation, we maintain an l-by-l matrix where the i, j-th entry of the matrix records $\mathsf{Pr}\left[\bar{M}_{P(i),P(j)}, M\right]$. Only l (or $2l$ if the assignment is changed on two queries) of these entries are updated according to the likelihood function, and the score function simply outputs the sum of the entries of this matrix.

6.5 Empirical Evaluation

6.5.1 Overview

Experimental Data and Auxiliary Information. We use the Enron email corpus [194] as the target dataset for all of our attacks. A description of the dataset and our pre-processing step can be found in Section 6.1.1.2. A major challenge for inference-style leakage-abuse attacks is deciding an appropriate model for evaluating their effectiveness in practice. Such a model should take into account both the distribution of queries as well as the distribution of the auxiliary information available to the adversary. Unfortunately, there do not exist concrete guidelines in the literature for how to construct such models; given this lack of precedence, we make certain assumptions that we believe are reasonable in practice.

Query Distribution. We use uniformly distributed keyword queries to evaluate our attacks. This is exactly as in previous attacks [94, 33, 17]. We note here that our attacks do not explicitly depend on the distribution of queries; hence a uniform distribution appears to be a reasonable choice.

Auxiliary Data Distribution. For the IKK attack, Islam *et al.* [94] proposed a method to model auxiliary information in an inference-style attack setting; their suggestion was to use an auxiliary co-occurrence pattern leakage obtained by adding Gaussian noise to the original co-occurrence pattern. However, this implicitly assumes a homogeneous distribution of keywords amongst the documents, which may not always be the case in practice. Instead, we opt to split the overall dataset into two halves: out of the 480000 documents in the dataset, half of the documents are used as the attack target and (a subset of) the other half of the documents are used to generate auxiliary information about the dataset. In total, we generate 10 different splits of the documents. For each split, we run 10 independent attacks with freshly generated observed co-occurrence

matrices. We measure the fraction of correctly guessed keywords/keys and report the average over the 100 runs as the query recovery rate.

KEYWORD EXTRACTION AND STEMMING. We extract keywords using the Natural language toolkit [156] in Python. Since previous attacks [94, 33, 17] used stemming, we run additional experiments with stemming and study its effects on query reconstruction rate in Section 6.5.5.

KEYWORD AND QUERY SELECTION. We use the $1000, 2000, 3000$ and 4000 most frequent keywords to build auxiliary co-occurrence matrices, and sample uniformly randomly without replacement from these most frequent keywords subsets of $250, 500, 750$ keywords as queried keywords. These queried keywords are used to build observed co-occurrence matrices. These observed and auxiliary co-occurrence matrices are then used as the inputs to our attacks. Further discussion on the choice of keywords for our attacks is presented in Section 6.6.

SECURITY PARAMETER SELECTION FOR THE TARGET CONSTRUCTIONS. We use the security parameters suggested in the original papers to run our attacks. We also investigate how changes in the security parameters affect query reconstruction rates.

Recall that **PRT-EMM** from [98] allows the client to pick a public parameter λ which controls the padded query response lengths as:

$$n'_{\mathsf{key}} = \lambda + F_{\mathsf{sk}}(\mathsf{key}||n_{\mathsf{key}}).$$

The authors suggested to set λ between 0 and $0.25n_{max}$. We use $\lambda = 0$ and $0.25n_{max}$ in our experiments. In addition, we use $\lambda = 0.5n_{max}$ to see the effect of additional padding on query reconstruction rate.

FP-EMM from [149] does not have a tunable parameter and we run our attacks on the **FP-EMM** as it is. **DP-EMM** from [149] uses parameter ϵ to set query response volumes to:

$$2n_{\mathsf{key}} + n^* + \mathbf{Lap}_{\mathsf{sk}}(2/\epsilon).$$

The authors suggested $\epsilon = 0.2$. In our attack, we use $\epsilon = 0.2$ just as suggested in the original paper. We also run experiments where ϵ is significantly smaller (more "secure"), ranging from 0.1 to 0.01.

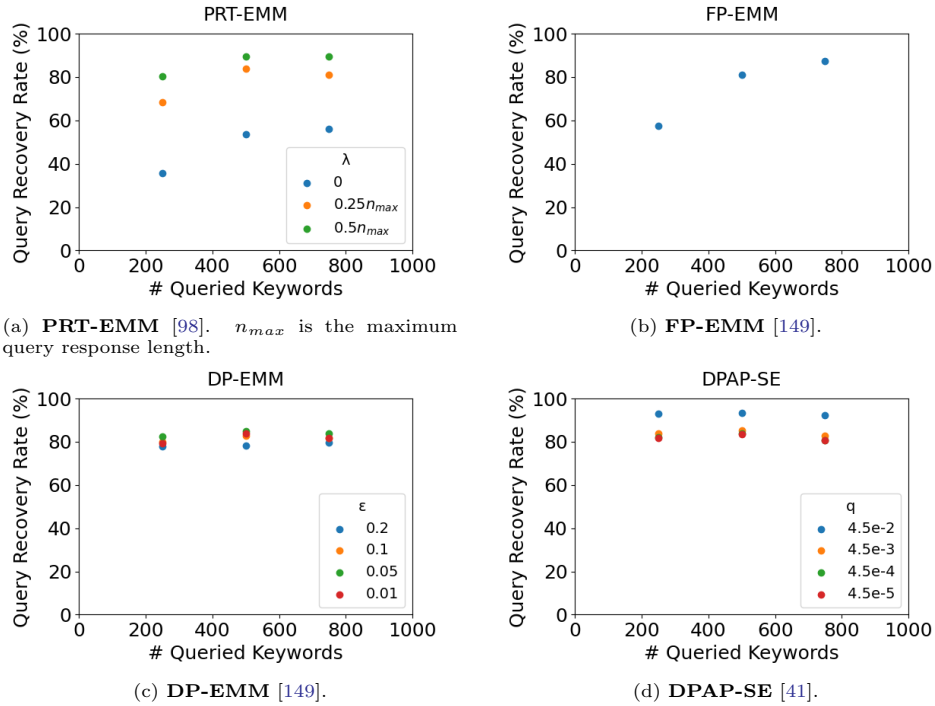
For **DPAP-SE** [41], the authors suggested $m = 6$ (the number of shards per document), $k = 2$ (a parameter of the erasure code which does not affect our attack), $p = 0.88703$ (the probability for which a keyword is kept in a shard) and $q = 0.04416$ as the parameters for the Enron [194] dataset. We used similar parameters where $m = 6, k = 2, p = 0.89$ and $q = 0.045$ in our experiments. A smaller q or a bigger q significantly reduces the efficiency of the construction so we opt to not run additional experiments with those parameters. Instead, we investigate how a smaller q affects query reconstruction rate. We use $q = 0.0045, 0.00045$ and 0.000045 as additional choices of parameters in our experiments.

IMPLEMENTATION. We implement our attacks in C using GNU Scientific Library [85] for randomness generation and probability calculations. We use our custom code for simulated annealing for the best performance. We parallelize our implementation using OpenMP [140]. Our implementation is highly scalable. It takes less than one minute

per run on the differentially-private schemes (**DP-EMM** and **DPAP-SE**) and no more than 6 minutes per run on the other schemes (**PRT-EMM** and **FP-EMM**) for all of our experimental settings, on a machine with an 8-core (16-thread) Sandy Bridge CPU clocked at 2.6 GHz.

Experiments. We present three sets of experiments on the target constructions in this section. In Section 6.5.2, we present the experimental results in basic settings, where the auxiliary co-occurrence matrix is built from all of the documents allocated for auxiliary information (50% of the total) using the 1000 most frequent keywords. We set the number of queried keywords to 250, 500 or 750, and the security parameters are allowed to vary. In Section 6.5.3, we set the number of queried keywords to 250, 500 or 750, and the security parameters to those suggested in the original papers, and vary the number of keywords used to build auxiliary information between 1000 and 4000. Just as before, all available documents are used in building auxiliary information. Finally, in Section 6.5.4, we use anywhere from 2.5% to 20% of the documents (5% to 40% of the 50% of documents allocated for auxiliary information) to build the auxiliary co-occurrence matrix, as a means to simulate auxiliary information with different levels of noise. The number of keywords used is set to 1000, and the number of queries is allowed to vary from 250 to 750. The security parameters are set to the ones recommended in the original papers.

6.5.2 Varying the Security Parameters of the Constructions



(a) **PRT-EMM** [98]. n_{max} is the maximum query response length.

(b) **FP-EMM** [149].

(c) **DP-EMM** [149].

(d) **DPAP-SE** [41].

Figure 6.3: Experimental results with varying security parameters. The 1000 most frequent keywords are used in auxiliary information.

The experimental results on **PRT-EMM** are shown in Figure 6.3a. We observe an increasing query recovery rate with more queried keywords and larger λ. The attack performs significantly worse with $\lambda = 0$. This is likely due to removal of co-occurrence

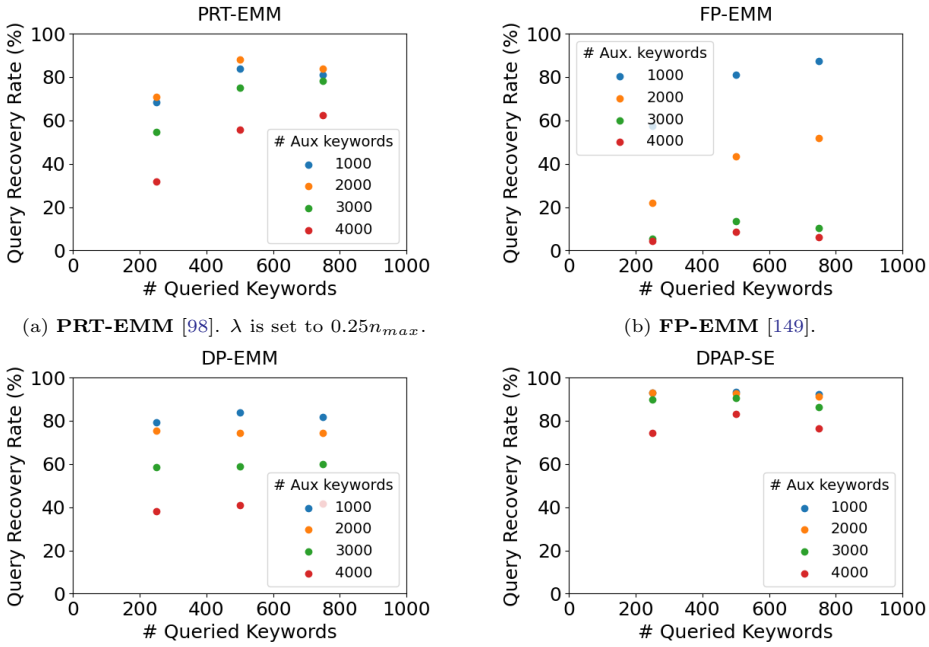
counts as the query response lengths are significantly shorter.

The experimental results on **FP-EMM** [149] are shown in Figure 6.3b. As expected, the attack performs better with more queried keywords. The attack is able to recover over 80% of the queried keywords if over 500 keywords have been queried, suggesting that full padding is ineffective at adding noise to the co-occurrence pattern leakage.

The experimental results on **DP-EMM** are shown in Figure 6.3c. The attack does not seem to be affected by the number of observed queries and the choice of ϵ much.

The results on **DPAP-SE** are shown in Figure 6.3d. The attack is able recover over 80% of the queries in all cases we have considered. The attack does not seem to perform worse with fewer observed queries. Interestingly, the attack performs better with a larger q. Intuitively, a larger q should generate more noise in the co-occurrence matrix, but it works in the favour of our attack. One possible explanation is that the auxiliary co-occurrence matrix is very different from the observed one, so our neighbourhood generation subroutine over-fits the assignments.

6.5.3 Varying the Number of Keywords in Auxiliary Information



(a) **PRT-EMM** [98]. λ is set to $0.25n_{max}$.

(b) **FP-EMM** [149].

(c) **DP-EMM** [149]. $\epsilon = 0.2$.

(d) **DPAP-SE** [41]. The security parameters used are $m = 6, k = 2, p = 0.89$ and $q = 0.045$.

Figure 6.4: Experimental results with varying number of keywords in auxiliary information.

Our experimental results on varying the number of keywords in the auxiliary information are shown in Figure 6.4. The security parameters we used can be found in the captions. Interestingly, the constructions behave very differently with respect to the number of keywords in the auxiliary information. In particular, our attack on **DPAP-SE** [41] is able to recover more than 70% of the keywords even with 4000 queried keywords. Our attack on **PRT-EMM** [98] works reasonably well with large numbers of keywords in the auxiliary information, managing over 50% query recovery rate except for the case

with 250 queried keywords and 4000 keywords in the auxiliary information.

Our attacks are less successful on **FP-EMM** and **DP-EMM** [149]. For **FP-EMM**, the query recovery rate falls rapidly as soon as the number of keywords in the auxiliary information is greater than 2000, whereas our attack on **DP-EMM** has lower query reconstruction rates with more than 3000 keywords in the auxiliary information. This suggests that **FP-EMM** and **DP-EMM** (with our choice of parameters) introduces more uncertainty than the other schemes.

6.5.4 Varying the Level of Noise in Auxiliary Information

Given that there is no widely accepted way of modelling noise in auxiliary information, we opt to use different numbers of documents in auxiliary information as a way to simulate different levels of noise – fewer documents means more noise. We use absolute distance and modified probability score to measure the level of noise introduced in each set of experiments we run. These measurements are defined as follows.

Absolute distance. Inspired by the Kolmogorov–Smirnov test [165], we define absolute distance to be the maximum absolute difference between the target co-occurrence matrix and auxiliary co-occurrence matrix:

$$D = \max_{i,j} \left| \frac{\bar{M}_{P(i),P(j)}}{N} - M_{i,j} \right|,$$

where $\bar{M}$ is the co-occurrence matrix generated from the target database (without using any construction on top), M is the co-occurrence matrix generated for auxiliary information, P is the true keyword assignments between the queries and keywords, and N is the number of documents in the target database. Intuitively, more noisy auxiliary information means a larger absolute distance.

Modified Probability Score. The second measurement of the level of noise we propose is the probability score. As the name suggests, the measurement is simply:

$$\mathsf{Pr}\left[\bar{M} \mid M\right].$$

It is clear that less noisy auxiliary information produces a larger probability score.

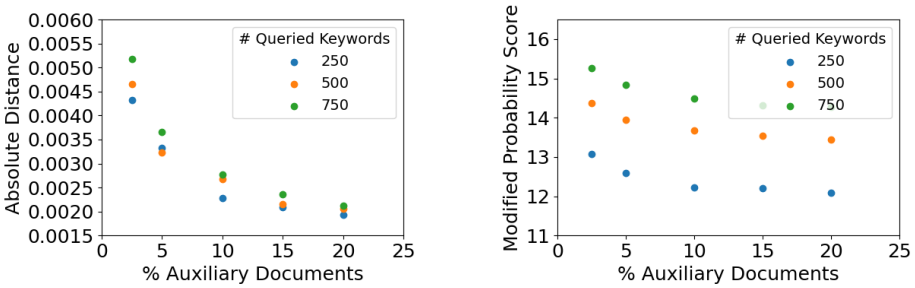
The probability score is very small for our datasets, so we use

$$D = \log(-\log(\mathsf{Pr}\left[\bar{M} \mid M\right]))$$

as a modified probability score instead. Less noisy auxiliary information produces a larger modified probability score just as before.

The measurements on the level of noise for the auxiliary datasets used in our attacks can be found in Figure 6.5. It can be seen clearly that the absolute distance and modified probability score increase as less documents are used as auxiliary information.

Our experimental results on varying auxiliary information are shown in Figure 6.6. The security parameters we used can be found in the captions. We observed that the attacks do not perform well when only 2.5% of the documents are used to construct the auxiliary information. This is likely due to the fact that the keywords are not identically distributed within the documents, as indicated by Figure 6.5. On the other

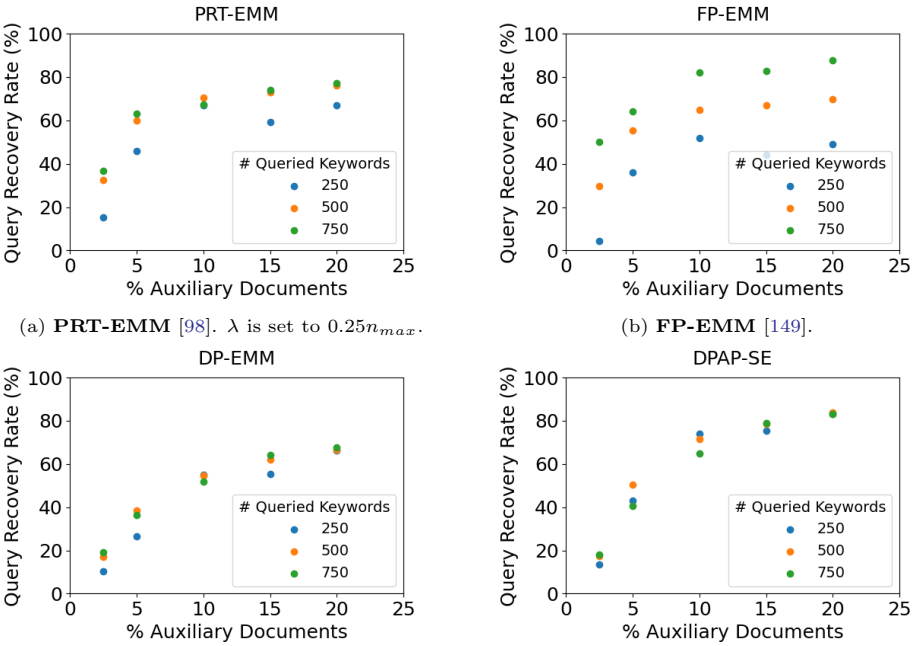


(a) Absolute distance.

(b) Modified probability score.

Figure 6.5: Measurements of the level of noise of the auxiliary data in our experiments.

hand, our attacks have comparable query reconstruction rates with 10% and 50% of the documents in the auxiliary information, suggesting that 10% of the documents is sufficient as auxiliary information and that our attacks are robust in a noisy setting.



(a) **PRT-EMM** [98]. λ is set to $0.25n_{max}$.

(b) **FP-EMM** [149].

(c) **DP-EMM** [149]. ϵ is set to 0.2.

(d) **DPAP-SE** [41]. The security parameters used are $m = 6, k = 2, p = 0.89$ and $q = 0.045$.

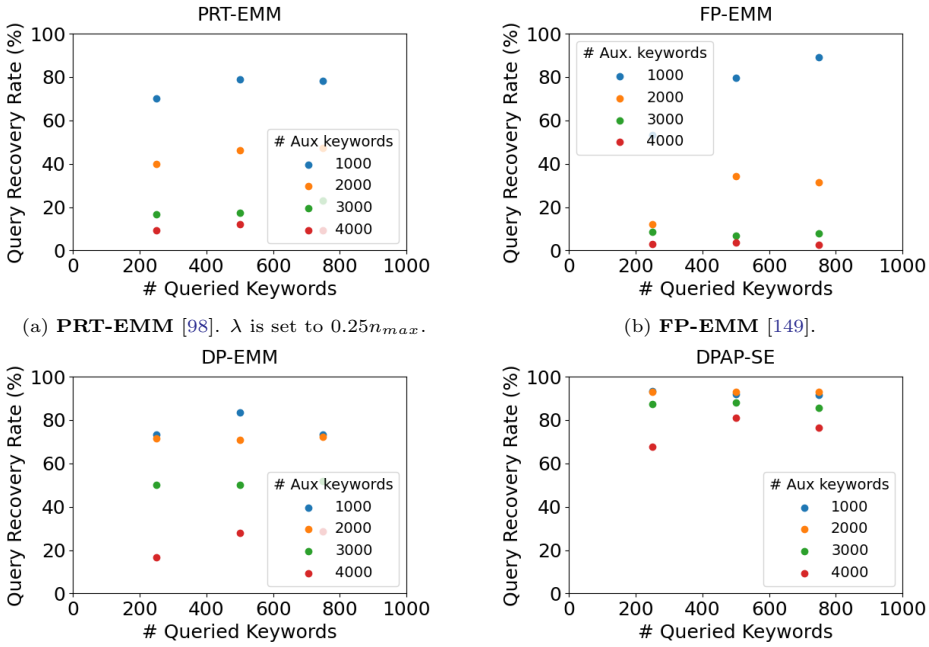
Figure 6.6: Experimental results with varying auxiliary information.

6.5.5 Use of Stemming

This section provides additional experimental results on the Enron dataset after stemming. We used the Porter Stemming Algorithm [153] implemented in the Natural language toolkit [156] as our stemming algorithm. The security parameters used for the constructions can be found in the captions.

Varying the Number of Keywords in Auxiliary Information. Our experimental results with varying number of keywords in auxiliary information is shown in Figure

6.7. The security parameters for the constructions used in our experiments can be found in the captions. The experimental results with stemming agree with those without stemming (Section 6.5.3), except for **PRT-EMM**, which performs worse with stemming as the number of keywords in auxiliary information increases. This is possibly because significantly more noise is introduced with stemming for **PRT-EMM**, as indicated by the frequency distribution plot in Figure 6.2 .



(a) **PRT-EMM** [98]. λ is set to $0.25n_{max}$.

(b) **FP-EMM** [149].

(c) **DP-EMM** [149]. $\epsilon = 0.2$.

(d) **DPAP-SE** [41]. The security parameters used are $m = 6, k = 2, p = 0.89$ and $q = 0.045$.

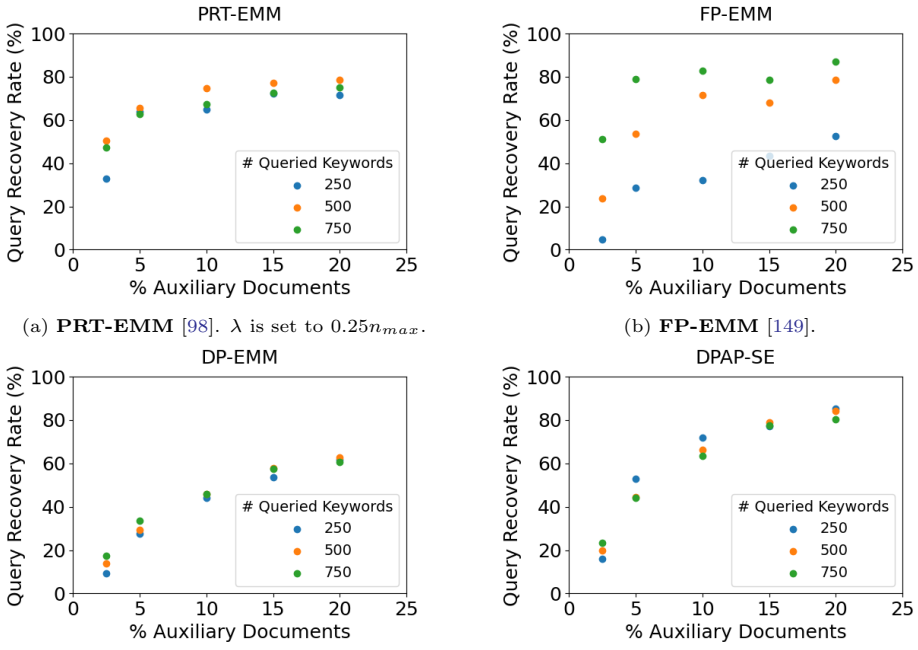
Figure 6.7: Experimental results with varying number of keywords in auxiliary information.

Varying the Level of Noise in Auxiliary Information. Our experimental results with varying level of noise in auxiliary information is shown in Figure 6.8. Just like before, the most frequent 1000 keywords are used to build auxiliary information. The security parameters for the constructions used in our experiments can be found in the captions. The performance of our attacks on auxiliary information with stemmed keywords is almost indistinguishable from those on auxiliary information with unmodified keywords.

6.6 Discussion

In this section, we discuss the implications of our attacks, the choices made in our experiments, and the practicality of our attacks.

On System-level leakage. As the state-of-the-art schemes do not scale for document retrieval and there is no other secure primitive that does, the only efficient instantiation of searchable encryption is to use one of the state-of-the-art schemes for the inverted index and build the encrypted document array naively. Our attacks show that this leads to insecure schemes.



(a) **PRT-EMM** [98]. λ is set to $0.25n_{max}$.

(b) **FP-EMM** [149].

(c) **DP-EMM** [149]. ϵ is set to 0.2.

(d) **DPAP-SE** [41]. The security parameters used are $m = 6, k = 2, p = 0.89$ and $q = 0.045$.

Figure 6.8: Experimental results with varying auxiliary information.

In fact, this is not a limitation of the state-of-the-art schemes, but a limitation of the duplication technique (which is used in almost all schemes in the literature) – as long as we have to use it for the encrypted document array to suppress access-pattern leakage, we will not be able to design a scalable searchable encryption scheme that is free from access-pattern leakage.

The natural question to ask is therefore:

How to design an efficient and system-wide secure searchable encryption scheme?

We answer that question in Chapter 7.

New Research Directions. In Section 6.1, we have argued that the information retrieval techniques in the literature are not suitable for searchable encryption. This is because the available techniques considered the single-document retrieval problem and they are not optimised for the many-document retrieval problem. This naturally raises the question: *How do we design information retrieval schemes that are optimised for the many-document retrieval problem?* The schemes we present in the next chapter are, in some sense, work in this direction.

Another interesting research direction is the trade-off between security and efficiency. Currently, the focus of the research is, by and large, on zero-knowledge information retrieval techniques. On the other hand, if we know we can leak certain information in an application without causing security/privacy problems, allowing for that leakage to exist may allow for design of much more efficient schemes. These papers [178, 185] can be used as the starting point of research in this direction.

On Differentially Private Access Patterns. Our attack on **DPAP-SE** [41] serves as a warning about the potential pitfalls of applying techniques from the differential privacy literature to STE without appropriately modelling and analysing the resulting leakage. As stated by Chen *et al.* in [41]:

"d-privacy implies that the adversary cannot distinguish between queries using distinct search terms that induce access patterns that are within specified distance (in terms of a distance metric d) of one another."

We believe, as demonstrated by our experiments, that d-privacy is not sufficient for query privacy, as the statistical distance between queries are far apart and easy to distinguish. Thus, provable guarantees in terms of differential privacy do not necessarily translate into security guarantees against leakage-abuse attacks *in general*.

We note here that the authors of [41] did establish the security of **DPAP-SE** (for certain sets of parameters) against existing attacks [94, 33], which necessarily rely on *exact* co-occurrence leakage. However, it is perhaps unwise to assume that security against a small set of known attacks translates to security against all possible attacks. Rather, one should assume that attacks can always get stronger. This is precisely what we demonstrate by showcasing stronger leakage-abuse attacks that work even in the presence of "noisy" co-occurrence pattern leakage.

On the Practicality of Our Attacks. As inference attacks exploiting perturbed access-pattern leakage and using only noisy auxiliary information, our attacks are more practical than the previous attacks [94, 33, 17, 143]. However, there are three important issues that impact the practicality of our attacks. These issues include modelling of auxiliary information, modelling of query distribution, and understanding the security impacts of our attack on different databases.

In our experiments, we use part of the database as auxiliary information. This is not always possible in practice. This raises questions on how different auxiliary information can be from the target database: do we need something that is statistically close, or some generic distribution suffices (just like classic frequency analysis). In addition, we used some of the most frequent keywords as the keyword universe and generated queries from a subset of these keywords in our experiments. This is most likely not how users would construct and query a database. In that sense, our experimental results can only be used as indications of security vulnerabilities. Finally, we have only experimented with one dataset. There can certainly be databases, for example, small databases with "poor" co-occurrence information, that are resilient against our attacks. Nonetheless, our attacks serve as a warning of excessive access-pattern leakage and system-level leakage.

This motivates us to propose a more robust security notion in Chapter 8. In addition to capturing system-wide security, the notion allows for different choices of auxiliary information, query distribution, and database. Hence, a security proof with respect to this notion fully quantifies if an attack (with a specific setting) is viable on a scheme.

Extending Our Attacks. We could potentially extend/strengthen our attacks based on other related attacks on property-preserving encryption/STE. For example, we could use Bayesian inference as in [16] if a good model for the prior distribution could be established.

Chapter 7

Construction: Searchable Encryption

As we have seen in Chapter 6, system-level leakage can lead to devastating attacks on the state-of-the-art searchable encryption schemes. Unfortunately, no existing technique can suppress this leakage in an efficient way – standard searchable encryption techniques lead to impractical storage overhead and other space-efficient primitives are not time-efficient.

This motivates SWiSSSE, a system-wide secure and practically efficient searchable encryption scheme, which we present in this chapter as a middle ground between these two classes of primitives. We build SWiSSSE gradually in this chapter. In Section 7.1, we devise the system-wide security notion for searchable encryption and present the basic data structure used by our scheme. Section 7.2 introduces the core ideas behind SWiSSSE with a simplified construction. Section 7.4 develop the construction into a system-wide secure static searchable encryption scheme. The leakage of the static scheme is then carefully cryptanalysed in Section 7.5 with the techniques we have developed in Chapter 6. Section 7.6 extends the static construction into a dynamic one. Section 7.7 presents an asymptotic performance analysis of SWiSSSE. Section 7.8 showcases practical efficiency of SWiSSSE. Finally, Section 7.9 summarises our contribution with SWiSSSE and discusses future research problems.

Contents

7.1 Preliminaries and Background

7.1.1 System-wide Security Definition for Searchable Symmetric Encryption

Recall from Section 4.3 that the standard security notion for searchable symmetric encryption captures security of the underlying scheme with a leakage function. A scheme that is proven secure with leakage function $\mathcal{L}$ should not leak more than what $\mathcal{L}$ describes. All of the schemes we have attacked in Chapter 6 are proven secure under this notion, but only for the index – *that is the very reason why our attacks work in the first place.*

To formally prevent system-wide attacks, the security notion should not focus on the index only. It should include document retrieval. We present the system-wide secure notion in Definition 7.1 below.

It is helpful to distinguish between the different places where leakage can occur, so we formally define a leakage function as a quadruple $\mathcal{L} = (\mathcal{L}^{\mathbf{Setup}}, \mathcal{L}^{\mathbf{Srch}}, \mathcal{L}^{\mathbf{Insert}}, \mathcal{L}^{\mathbf{Delete}})$.

The security definition is parametric in the type of queries the adversary is allowed: if these are only search queries then we are in the static setting whereas if they also include update queries then we are in the dynamic scenario. Furthermore, we can distinguish between non-adaptive and adaptive adversaries, depending on how they chose their queries. In both cases the adversary attempts to distinguish between a real world (where it interacts with the real scheme) and an ideal world (where it interacts with a simulator which can only access the leakage from the real execution).

Definition 7.1 (Security of SSE schemes)**.** Let $\Sigma = (\mathbf{Setup}, \mathbf{Srch}, \mathbf{Insert}, \mathbf{Delete})$ be an SSE scheme and consider the following probabilistic experiment where $\mathcal{A}$ is a stateful adversary, $\mathcal{S}$ is a stateful simulator and $\mathcal{L}$ be the leakage function.

$\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda)$:

1. The adversary $\mathcal{A}$ selects a database $\mathbf{DB}$ and gives it to the challenger $\mathcal{C}$.
2. The challenger $\mathcal{C}$ generates a key sk, and encrypts the database as $\mathbf{EDB} \leftarrow \mathbf{Setup}(1^\lambda, \mathsf{sk}, \mathbf{DB})$. The challenger $\mathcal{C}$ sends the encrypted database $\mathbf{EDB}$ to the adversary $\mathcal{A}$.
3. The adversary picks a polynomial number of queries $\mathbf{q}_1, \ldots, \mathbf{q}_{\mathtt{poly}(\lambda)}$. For each query, the challenger $\mathcal{C}$ interacts with the adversary $\mathcal{A}$ to execute the query protocol, where the challenger plays the client and the adversary plays the server.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

$\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda)$:

1. The adversary $\mathcal{A}$ selects a database $\mathbf{DB}$ and gives $\mathcal{L}^{\mathbf{Setup}}(\mathbf{DB})$ to the simulator $\mathcal{S}$.
2. Using $\mathcal{L}_{\mathbf{Setup}}(\mathbf{DB})$, the simulator $\mathcal{S}$ generates an encrypted database $\mathbf{EDB}$ and return it to the adversary $\mathcal{A}$.

3. The adversary picks a polynomial number of queries $\mathbf{q}_1, \ldots, \mathbf{q}_{\mathtt{poly}(\lambda)}$. For each query, the simulator $\mathcal{S}$ computes the transcript of the query using the appropriate component of the leakage function $\mathcal{L}$, and sends it to the client.
4. Finally, the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$.

We say that Σ is $\mathcal{L}$-secure, if there exists a probabilistic polynomial-time (PPT) simulator $\mathcal{S}$ such that for all PPT adversary $\mathcal{A}$,

$$\left|\Pr[\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda) = 1]\right| \leq \mathtt{negl}(\lambda).$$

We say that Σ is non-adaptively secure if the adversary $\mathcal{A}$ chooses the queries before executing them. We say that Σ is adaptively secure if the adversary $\mathcal{A}$ can choose his queries based on the past transcripts.

7.1.2 Key-value Store

The basic data structure used by all our constructions is a "key-value store". This data structure implements an associative array abstract data type that maps (non-cryptographic) *keys* to *values*. The data structure supports efficient execution of the following operations:

- **init** : takes no input and initialises an empty key-value store.
- **get** : takes as input a key k and returns the value associated to the key. We abuse the notation and use **get** for getting the values for a set of keys too.
- **put** : takes as input a key-value pair (k, v) and sets the value associated to k to v. We abuse the notation and use **put** for putting a set of key-value pairs too.
- **contains** : takes as input a key k and returns a boolean which indicates if k is one of the keys in the key-value store.
- **del**: takes as input a set of keys $\mathcal{I}$ and remove the key-value pairs with the keys in $\mathcal{I}$ from the key-value store.
- **pop**: takes as input a natural number n. It selects n uniformly random key-value pairs from the store, removes them from the store and returns them as the result of the call.

We do not explicitly distinguish between the name of the data structure and its state. For example, if S is a key-value store we write $S.\mathbf{get}(k)$ for the result returned by $\mathbf{get}(k)$ on the current state of S.

7.2 Simple Construction

To highlight some of the key techniques underlying SWiSSSE, we start by considering a highly simplified setting where we have a static database in which each document contains precisely one searchable keyword. We explain how to extend this approach to the general case of arbitrarily many keywords per document in Section 7.4.

Server Storage. Our SSE scheme offloads the storage of two encrypted data structures to the server – an encrypted *lookup table* that is indexed by the set of keywords

in the database, and an encrypted *document array*, which is indexed by the documents. For each keyword, the corresponding entry in the lookup table stores (in encrypted form) pointers to the corresponding entries in the document array. For each document, the document array stores (again in encrypted form) the contents of the document, the list of keywords it contains, and some auxiliary information necessary for searches. Both indices are implemented as key-value stores where keys are calculated using a pseudorandom function, and values are encryptions under a symmetric key owned by the client.

Bucketization. To limit keyword frequency leakage (i.e. in how many documents a keyword appears), we use a frequency bucketization strategy, i.e., we pad the outsourced database with fake occurrences of keywords as well as with additional fake documents. For each entry in the keyword lookup index we store a mix of pointers pointing to "real" and "fake" documents in the document array. Bucketization inherently introduces a trade-off between security and efficiency, since the work done by the server is now proportional to the "bucket frequency" as opposed to the true frequency of the keyword. We expand more on this in Section 7.3.

Local Stash and Write-backs. If the server-side data structures use static addresses (meaning that a given address in the keyword lookup index and/or the document array always corresponds to a fixed keyword and/or document), then the scheme inherently suffers from "access pattern leakage". To prevent such leakage (and hence the known attacks exploiting them), we ensure that all accesses, even those involving the same keyword, "touch" different parts of the server state. We achieve this through a delayed, pseudorandom write-back technique: after each operation involving a keyword we update the addresses of the keyword and of all documents containing it across the encrypted data structures stored at the server.

More concretely, we first "locally stash" all the information returned by a server in response to a keyword search query at the client. This information includes the list of documents, the number of times the keyword has been accessed, and the number of times each document containing the keyword has been accessed. Then, at a later time, the client issues a "write-back operation", wherein it flushes out this information from its stash onto the encrypted data structures at the server, using fresh encryptions and to a new, pseudorandomly generated set of addresses. The next time the client issues a search query involving the same keyword, the server will access the new set of addresses in both the lookup index and the document array, and will observe only the fresh encryptions of the same (or updated) content.

Note that addresses as described above correspond to keys in the server's key-value stores. The client need only assume that the server provides a correct implementation of the key-value store and avoid using colliding addresses/keys. We defer a formal description of the stashing and write-back procedure to Section 7.4. Note also that, intuitively, a larger client stash leads to a larger storage requirement on the client, but lowers the frequency with which the client needs to flush its stash and trigger write-back operations. This allows flexibility in trading-off client storage with bandwidth requirements.

Garbage Collection. Observe that each write-back operation uses up newly generated addresses (keys) in the encrypted keyword lookup index and the encrypted document array. The corresponding "old addresses" are never used in the future and can be

garbage-collected by the server after handling a query. This avoids the required server-side storage growing linearly with the number of queries. Of course, at this point, the data read is already stored in the stash of the client, so there is no data loss on account of the garbage collection.

Similar Techniques in the Literature. The techniques we used in SWiSSSE have been widely used in the literature.

Bucketization or padding has been widely used in the literature. This includes padding in traffic volumes [72], storage systems [197], and structured encryption schemes [33, 26, 98, 149, 57]. The idea of stash and randomised write-back were heavily used in the literature of ORAM [84, 171, 50, 189, 74, 147].

However, SWiSSSE is the first scheme that combines the two in the particular way and it is the first time we are able to show that a "leaky" scheme is practically secure with cryptanalysis.

Search Queries. A search query involving a keyword kw proceeds as follows:

1. The client first checks its private stash to see if the transcript of a previous query involving kw already exists in its stash. If yes, it directly obtains the search result from the stash and does not initiate any further interaction with the server. Otherwise, the client sends a search token to the server and receives back the entry corresponding to kw in the encrypted lookup index.
2. Next, the client decrypts the entry received from the server and retrieves the list of pointers to addresses in the encrypted document array, which it sends across to the server. The server retrieves the corresponding entries from the encrypted document array and sends these back to the client.
3. Upon receiving the encrypted entries from the server, the client decrypts them, discards any fake documents and retains the real ones.
4. Finally, the client caches the whole transcript of the search operation in its local stash, so that its contents may be written-back at a later point in time.

Figure 7.1 illustrates with an example how keyword searches and delayed write-backs affect the client stash and the state of the encrypted data structures at the server. Note that write-backs are only triggered intermittently by the client whenever its private stash is filled to capacity and needs to be flushed. Hence, the search latency is not affected by the latency of write-back operations.

System-level Leakage. There are two kinds of system-level leakage potentially incurred by SWiSSSE – access-pattern leakage and write-back leakage. We provide here an informal overview of what the server can infer from these leakage. We defer the formal leakage profile enumeration and its cryptanalysis to Sections 7.4 and 7.5.

Observe that across both encrypted data structures, the server accesses any given address at most twice, once during a write-back and once during a subsequent read operation (the entry is subsequently deleted and the corresponding address is never re-used). Hence, unlike most existing SSE schemes, including those in [53, 36, 35, 97, 116], the access-pattern leakage in our simplified schemes does not allow the server to immediately infer

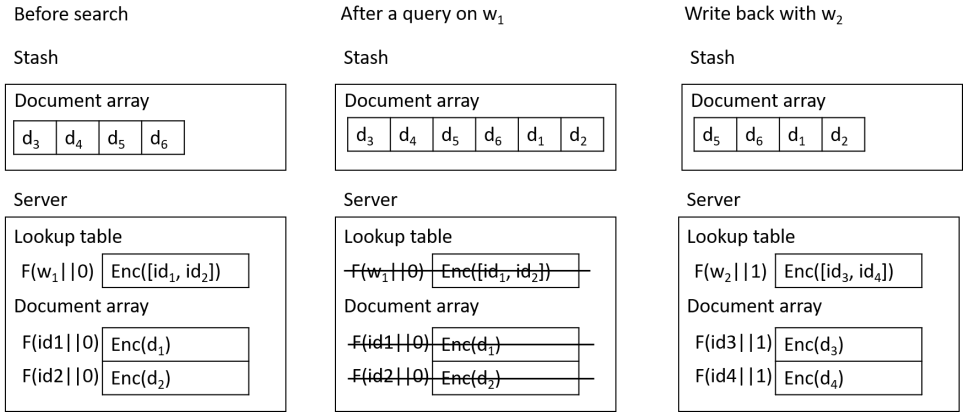


Figure 7.1: An illustration of the operations related to keyword search and delayed write-backs corresponding to two keywords kw_1 (which occurs in documents doc_1 and doc_2) and kw_2 (which occurs in documents doc_3 and doc_4). The query on kw_2 is executed before the query on kw_1. The figure on the left shows the status of the client stash and the encrypted data structures before the query on kw_1. As soon as the query on kw_1 is made, the corresponding entries in the encrypted data structures are deleted by the server, while the client stash gets updated with the same information. The figure in the middle reflects this. The figure on the right illustrates a delayed write-back operation for kw_2.

whether the same keyword was queried twice. It also does not immediately reveal if the same document identifier appears across searches pertaining to different keyword queries.

The other potential source of leakage is the write-back operations. In order to make it hard to correlate write-back operations relating to the same search operation, we use a "randomized" write-back strategy. More specifically, write-backs corresponding to multiple queries are randomly interspersed in time and do not follow any specific ordering: what is written back during a particular write-back operation is pseudorandomly selected from everything sitting in the stash. Thus write-backs "mix and match" across multiple queries from the client's stash. Hence, each individual write-back that occurs at time t may correspond to more than one query executed prior to time t, and an adversarial server will find it hard to correlate a write-back operation to the query to which it corresponds.

Intuitively, the ease with which the leakage can be exploited grows with the frequency of write-back operations, and hence inversely with the size of the local stash at the client. This provides yet another source of security-efficiency trade-off that we discuss in detail subsequently.

7.3 Bucketization

As mentioned in Section 7.2, we mitigate volume leakage in SWiSSSE through a frequency bucketization strategy over the set of keywords in the document collection. At a high level, this entails dividing the list of all keywords across multiple buckets such that each keyword in the same bucket is "padded" to have the same frequency as the most frequent keyword in that bucket (also referred to as "bucket frequency"). The core aim of bucketization is to prevent generic volume/frequency leakage-based attacks on SSE.

Bucketization can either be implemented as a pre-processing step on the raw docu-

ments (i.e. adding/removing keywords from the real documents and/or creating fake documents) or during queries directly (i.e. retrieve more/less document identifiers and documents depending on the bucket size). We pick the earlier option.

In this section, we detail the bucketization strategy used in our constructions. Later, in Section 7.5, we present our cryptanalysis results on the bucketization strategy, propose concrete parameters and discuss security versus efficiency trade-offs.

Worst-case padding. Note that a trivial bucketization strategy is "worst-case padding" where every keyword is padded to have the same frequency. Although theoretically free of volume leakage, this approach is extremely inefficient and either imposes a linear search overhead [149] or a quadratic storage complexity, which we wish to avoid. Hence we opt for strategies that, while theoretically less secure, offer significantly better leakage versus efficiency trade-offs in practice.

Bucketization strategy. Our bucketization strategy works as follows. Firstly, we arrange the keywords in decreasing order of frequency and divide them into buckets of certain sizes (it can vary from bucket to bucket). We then pad with fake data so as to equalise the frequency of all keywords in each bucket. Hence, the adversary can guess the queried keyword with probability at most one over the bucket size if there is no other leakage. The trade-off here is that larger buckets incur lower leakage but more padding and so greater search overheads.

We provide in Section 7.5 an empirical evaluation of the security offered by this strategy with different bucket sizes against co-occurrence leakage-based cryptanalysis over real-world databases, as well as their impacts on the storage and communication overheads of SWiSSSE.

Padding Strategies. Bucketization of keywords requires padding the original database with fake data. A natural padding strategy is to add "fake occurrences" of each keyword across the existing documents, such that the keyword frequencies grow to the desired bucket frequency. This approach incurs only moderate additional storage overheads at the server since the number of documents remains the same, and only the encrypted lookup index grows in size. Indeed this suffices for the static version of SWiSSSE.

However, we choose to additionally insert fake documents containing uniformly random keywords into the database, subject to the constraints imposed by the bucketization strategy. While this requires more storage on the server, it supports dynamic databases more elegantly, since once can perform document insertion by "transforming" the slot for a fake document into a slot for a real one in a computationally indistinguishable manner (see the dynamic version of SWiSSSE in Section 7.6 for details).

7.4 Static SWiSSSE

In this section, we formally describe SWiSSSE for general but static databases, where each document in the database contains multiple searchable keywords. The main technical challenge compared to the simplified case described in Section 7.2 is that each time we update a document address (during a write-back) we must also update all pointers to that new address for the multiple keywords that appear in that document. We handle this using auxiliary write-backs.

We begin with an overview of the key changes from the simplified case, and then provide the formal description. The formal description adheres to the definitions for SSE in Section 7.1.1.

7.4.1 Changes from the Simplified Case

Auxiliary Write-Backs. At a high level, we opt for the following strategy in the case of general databases: whenever a document doc_ℓ is scheduled to be flushed from the client's stash and written back to the encrypted document array, the client additionally schedules an *auxiliary* write-back for each keyword kw_i occurring in doc_ℓ.

To understand why auxiliary write-backs ensure search correctness, consider the following three scenarios:

1. Suppose that a query on kw_i is issued before doc_ℓ is written back. At this point, the client can directly retrieve doc_ℓ from its private stash.
2. Next, suppose that a query on kw_i is issued after doc_ℓ has been written back but before the auxiliary write-back for kw_i is executed. In this case, the pointer in the lookup index is invalid, but client can refer to its stash to check if an auxiliary write-back for kw_i is scheduled. This allows it to recover the new pointer and use it for the search operation.
3. Finally, suppose that a query on kw_i is issued after the auxiliary write-back for kw_i has been executed. This again does not affect search correctness because the entry for kw_i in the keyword lookup index now points to the "new" address for doc_ℓ in the document array.

Moreover, these write-backs impose no extra bookkeeping overhead at the client, since they do not need to be executed in sync with the original write-back for the document.

Restructuring the Keyword Lookup Index. To support auxiliary write-backs, we restructure the keyword lookup index. For simplicity of presentation, we present a simplified version of the restructuring. This incurs some undesirable leakage, which we address subsequently.

Instead of storing a single entry for each keyword kw_i, we now store an entry for each keyword-document pair (kw_i, doc_ℓ) such that doc_ℓ contains kw_i. The address for this entry is generated as $F(K, kw_i||j||\mathsf{cnt}_{kw_i})$, where F is a PRF with key K, j is a counter that runs from 0 to $|\mathbf{DB}(kw_i)| - 1$ (where $\mathbf{DB}(kw_i)$ denotes the set of documents containing keyword kw_i), and cnt_{kw_i} is a per-key word counter held in the client's stash which records how many times kw_i has appeared in search queries. Each entry stores a ciphertext encrypting a *single* pointer to the address of some doc_ℓ in the document array.

In keeping with our "frequency bucketization strategy", we also create and store in the lookup index additional "fake" entries of the form $(kw_i, \widetilde{doc}_\ell)$, where $\widetilde{doc}_\ell$ is a fake document. The address for each such fake entry is generated as $F(K, kw_i||j'||\mathsf{cnt}_{kw_i})$, where j' is a counter that runs from $|\mathbf{DB}(kw_i)|$ to one less than the bucket size for kw_i. Each such entry again stores a single ciphertext, now encrypting a pointer from kw_i to the fake document $\widetilde{doc}_\ell$.

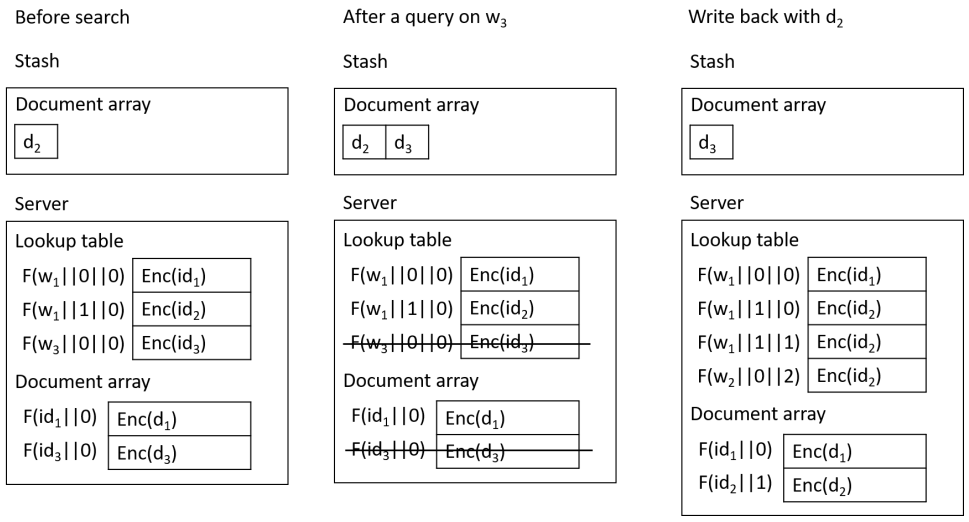


Figure 7.2: An illustration of the operations related to keyword search and delayed normal/auxiliary write-backs corresponding to keyword kw_3 (which occurs in the document doc_3) and document doc_2 (which contains multiple keywords, including kw_1 and kw_2), respectively. The figure on the left shows the client stash and the encrypted data structures before the query on kw_3. As soon as the query on kw_3 is made, the corresponding entries in the encrypted data structures are deleted by the server, while the client stash gets updated with the same information. The figure in the middle reflects this. Finally, the figure on the right illustrates a delayed auxiliary write-back operation for doc_2. Observe that doc_2 gets written back (in encrypted form) to a pseudorandomly generated location in the document array. Additionally, as part of the auxiliary write-backs associated with doc_2, the keyword-document pairs (w_1, doc_2) and (w_2, doc_2) get written back (in encrypted form) to pseudorandomly generated locations in the keyword lookup index.

This makes the lookup index amenable to auxiliary write-backs. In particular, the *auxiliary* write-back for a keyword kw_i occurring in doc_ℓ targets the address $F(K, kw_i||j||\mathsf{cnt}_{kw_i})$, and updates specifically the pointer from kw_i to doc_ℓ.

Leakage to the Server. Note that each auxiliary write-back corresponding to a keyword-document pair (kw_i, doc_ℓ) causes the server to *overwrite an existing entry* in the keyword lookup index. This is never the case for "normal" write-back operations, since they target freshly generated pseudorandom addresses. This leaks some information to the server. In particular, the server is able to correlate each auxiliary write-back involving a certain keyword kw_i with the last normal write-back involving kw_i (since both sets of write-back operations target the same addresses in the lookup index). This leaks the following information: (a) whether kw_i was the subject of a previous query, and if yes, (b) that kw_i must have been queried at some point in time prior to the corresponding normal write-back operation.

Auxiliary Addresses for Auxiliary Write-Backs. To prevent the server from correlating auxiliary write-backs to the last normal write-back involving the same keyword, we choose to dissociate the two sets of addresses. More concretely, we generate separate sets of addresses for normal and auxiliary write-backs involving the same keyword:

$$\mathsf{addr}_{\mathrm{norm}}(kw_i, doc_\ell, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(2 * \mathsf{cnt}_{kw_i})),$$
$$\mathsf{addr}_{\mathbf{aux}}(kw_i, doc_\ell, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(2 * \mathsf{cnt}_{kw_i} + 1)),$$

where j is again a counter that runs from 0 to $|\mathbf{DB}(kw_i)| - 1$ and cnt_{kw_i} is again the per-key word counter held in the client's stash which records how many times kw_i has

appeared in search queries. Similarly, for the fake documents associated with kw_i, we generate separate sets of addresses for normal and auxiliary write-backs:

$$\mathsf{addr}_{\mathrm{norm}}(kw_i, \widetilde{doc_\ell}, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(2 * \mathsf{cnt}_{kw_i})),$$

$$\mathsf{addr}_{\mathbf{aux}}(kw_i, \widetilde{doc_\ell}, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(2 * \mathsf{cnt}_{kw_i} + 1)),$$

where j' is a counter that runs from $|\mathbf{DB}(kw_i)|$ to one less than the bucket size for kw_i.

Ensuring Search Correctness. Finally, to ensure that a search query on the keyword kw_i correctly takes into account all auxiliary write-backs involving kw_i, the client now requests the server to access both sets of write-back addresses for kw_i – normal and auxiliary – in the keyword lookup index. If both sets of addresses exist for a particular document, the client uses the pointer stored in the auxiliary write-back address; otherwise it uses the pointer stored in the normal write-back address.

Figure 7.2 illustrates with an example how keyword searches and delayed normal/auxiliary write-backs affect the client stash and the state of the encrypted data structures at the server in the general version of static SWiSSSE.

7.4.2 Formal Description of Static SWiSSSE

We now turn the aforementioned ideas into a formal description of static SWiSSSE.

Notations. We write $\mathbf{DB}[i]$ to mean the i-th document in the database $\mathbf{DB}$ (by assuming an explicit ordering). We write $KW(\mathbf{DB}[i])$ to mean the set of keywords in document $\mathbf{DB}[i]$.

SWiSSSE.**Setup**. Algorithm 7.1 describes the setup procedure of static SWiSSSE. The description uses a symmetric encryption scheme $(\mathbf{KGen}_1, \mathbf{Enc}, \mathbf{Dec})$ and a pseudorandom function F with key generation algorithm $\mathbf{KGen}_2$. Note that for readability, we omit explicitly describing the keys used by these cryptographic functions.

Note that the server stores an encrypted lookup table `Svr.EI` (to store the map between the keywords and the document addresses in the encrypted form) and an encrypted document array `Svr.EA` (to store the map between the document addresses and the actual documents in encrypted form). The addressing mechanism and encrypted contents for these data structures are as described in Section 7.1.2.

Similarly, the client stores in its local stash a plaintext lookup index `Clt.I` (to store the map between the keywords and where they are in the encrypted document array before they are written back to the server), and a plaintext document array `Clt.A` (to store the map between the document identifiers and the documents). In particular, the plaintext document array is used to store the documents retrieved from the previous queries and randomly select an appropriate number of documents for auxiliary write-backs.

To implement the keyword bucketization strategy, the client creates a "padded version" $\mathbf{DB}'$ of the original database $\mathbf{DB}$ before encrypting and offloading it to the server. More concretely, the client selects a map $\mathtt{G} : KW \to \mathbb{N}$. This map assigns each keyword to a bucket, such that all keywords in the same bucket have the same frequency in the padded database $\mathbf{DB}'$.

Algorithm 7.1 Static SWiSSSE.**Setup**

procedure $\texttt{Clt}.\textbf{Setup}(1^\lambda, \mathbf{DB})$
  /* *Key generation* */
  $\texttt{Clt.sk}_1 \leftarrow \textbf{KGen}_1(1^\lambda)$
  $\texttt{Clt.sk}_2 \leftarrow \textbf{KGen}_2(1^\lambda)$
  Generate map $\texttt{G} : KW \to \mathbb{N}$ for bucketization
  $\texttt{Clt.G} \leftarrow \texttt{G}$
  /* *Generate fake documents* */
  $\mathbf{DB}' \leftarrow \textbf{Fake\_Doc\_Gen}(\mathbf{DB}, \texttt{Clt.G})$
  $\texttt{Clt}.N \leftarrow \left|\mathbf{DB}'\right|$
  /* $\texttt{Clt}.N$ *allows the client to locally maintain the size of the padded database. This is used subsequently in the keyword search algorithm.* */
  $\texttt{EI}, \texttt{EA} \leftarrow \{\}$
  for $i \in 1, \ldots, \left|\mathbf{DB}'\right|$ **do**
    /* *Get the set of keywords with counters* */
    $x \leftarrow \left\{(kw, \texttt{Clt.KWCtr}[kw]) \mid kw \in KW(\mathbf{DB}'[i])\right\}$
    /* *Update the lookup index* */
    /* *In the following expression,* $KW(\boldsymbol{DB}'[i])$ *denotes the set of keywords in the* i*-th document of the padded database.* */
    for $kw \in KW(\mathbf{DB}'[i])$ **do**
      $j \leftarrow \texttt{Clt.KWCtr}[kw]$
      $\texttt{EI} \leftarrow \texttt{EI} \cup (F(kw||j||0), \textbf{Enc}(\mathsf{id}(\mathbf{DB}'[i])))$
      /* *Note that the zero at the end of the PRF input essentially indicates that the number of search operations involving* kw *is initially* 0 *at setup* */
      $\texttt{Clt.KWCtr}[kw] \leftarrow \texttt{Clt.KWCtr}[kw] + 1$
    /* *Insert the encrypted document* */
    $\texttt{EA} \leftarrow \texttt{EA} \cup (F(i||0), \textbf{Enc}(x||\mathbf{DB}'[i]))$
  /* *Reset the keyword counter* */
  for $kw \in KW(\mathbf{DB}')$ **do**
    $\texttt{Clt.KWCtr}[kw] \leftarrow 0$
  /* *Initialise the stash* */
  $\texttt{Clt.I}.\textbf{init}()$
  $\texttt{Clt.A}.\textbf{init}()$
  Send $(\texttt{EI}, \texttt{EA})$ to the server

procedure $\texttt{Svr}.\textbf{Setup}(\texttt{EI}, \texttt{EA})$
  $\texttt{Svr.EI}.\textbf{init}()$
  $\texttt{Svr.EA}.\textbf{init}()$
  $\texttt{Svr.EI}.\textbf{put}(\texttt{EI})$
  $\texttt{Svr.EA}.\textbf{put}(\texttt{EA})$

Let $|\mathbf{DB}(kw)|$ and $\left|\mathbf{DB}'(kw)\right|$ denote the frequency of the keyword kw in the original and padded databases, respectively. The map $\texttt{G}$ allows the client to determine a padding procedure **Fake_Doc_Gen**; the procedure pads the input database $\mathbf{DB}$ with "fake" documents to obtain a padded version $\mathbf{DB}'$ such that $\left|\mathbf{DB}'(kw)\right| = \texttt{G}(kw)$ for each keyword kw. The client may use any padding strategy to achieve the desired keyword frequencies as specified by $\texttt{G}$.

Additionally, to suppress volume leakage, the document addresses and the document contents are padded to fixed lengths ℓ_0 and ℓ_1 respectively (both assumed to be pub-

Algorithm 7.2 Static SWiSSSE.**Srch**: Address Retrieval Sub-Routine

procedure Clt.**TokenGen**(kw)
    $L \leftarrow \{\}$
    /* *Generate all the possible addresses for the keyword* */
    for $j \in 0, \ldots,$ Clt.G(kw) $- 1$ **do**
        $L \leftarrow L \cup \{F(kw||j||2 * \texttt{Clt.KWCtr}[kw])\}$
        $L \leftarrow L \cup \{F(kw||j||2 * \texttt{Clt.KWCtr}[kw] + 1)\}$
    /* *Roll forward the counter corresponding to kw in preparation for the next query on kw* */
    Clt.KWCtr[kw] $\leftarrow$ Clt.KWCtr[kw] $+ 1$
    Send L to the server

procedure Svr.**Index_Lookup**(L)
    Send Svr.EI.**get**(L) to the client

Algorithm 7.3 Static SWiSSSE.**Srch**: Encrypted Document Retrieval Sub-Routine

procedure Clt.**Document_Retrieval**(kw, EL)
    $L \leftarrow$ the latest addresses of the keywords from **Dec**(EL) if they are not in Clt.I
    Add random document identifiers between 0 and Clt.$N - 1$ that are not in the stash to L until $|L| = 2 \cdot$ Clt.G(kw)
    $M \leftarrow \{\}$
    for $id \in L$ **do**
        /* *Compute the document addresses* */
        $M \leftarrow M \cup F(id||\texttt{Clt.ArrCtr}[id])$
        /* *Increase the counters* */
        Clt.ArrCtr[id] $\leftarrow$ Clt.ArrCtr[id] $+ 1$
    Send M to the server

procedure Svr.**Document_Retrieval**(M)
    Send Svr.EA.**get**(M) to the client

lic parameters) prior to encryption. Note, however, that we do not perform worst-case document padding, which would potentially incur huge storage and communication overheads. Instead we use a fragmentation strategy where each document is fragmented into sub-documents of size ℓ_1; so we only really perform padding for the last fragment in case it has size less than ℓ_1. We avoid these details in Algorithm 7.1 for the sake of readability.

SWiSSSE.**Srch**. We now describe the keyword query procedure for SWiSSSE. For ease of representation, SWiSSSE.**Srch** is broken up into three sub-routines described in Algorithms 7.2, 7.3 and 7.4. Again, for readability, we omit explicitly describing the keys used by the cryptographic functions in these algorithms.

These algorithms formally depict the following steps taken by the client during a keyword query:

- **Algorithm 7.2:** The client generates a search token to look up both the normal and auxiliary write-back addresses for kw_i in the encrypted keyword lookup index, receives the corresponding encrypted entries from the server, and decrypts the results locally to identify the relevant entries in the encrypted document array. It

Algorithm 7.4 Static SWiSSSE.**Srch**: Auxiliary Write-Back Sub-Routine

1: **procedure** Clt.**Write_Back**$(M, \bar{kw})$
2: $UA \leftarrow \{\}$
3: /* *Get random documents from the stash,* b_i *is a bit to indicate if* kw_i *was the leading keyword* */
4: $D \leftarrow$ Clt.A.**pop**$(\lfloor |\text{Clt.A}| / 2 \rfloor)$
5: **for** $(\{(kw_i, j_i, b_i)\}, doc) \in D$ **do**
6: /* *Encrypt the new documents* */
7: $UA \leftarrow UA \cup \{(F(\mathsf{id}(doc)) || \text{Clt.ArrCtr}[\mathsf{id}(doc)]),$ **Enc**$(\{(kw_i, j_i)\} || doc))\}$
8: /* *Update the stash for the lookup index* */
9: **for** $(kw, j, b) \in \{(kw_i, j_i, b_i)\}$ **do**
10: Clt.I.**put**$((F(kw||j||2 * \text{Clt.KWCtr}[kw] + b),$ **Enc**$(\mathsf{id}(doc))))$
11: /* *Decrypt the documents retrieved and insert them into the document array* */
12: Clt.A.**put**(**Dec**(M))
13: Send $(\text{Clt.I.}$**pop**$(\lfloor |\text{Clt.I}| / 2 \rfloor), UA)$

14: **procedure** Svr.**Write_Back**$((UI, UA))$
15: Svr.EI.**put**(UI)
16: Svr.EA.**put**(UA)

also updates the counter keeping track of the number of search operations involving kw_i (this helps generate the new write-back address for kw_i in the encrypted keyword lookup index).

- **Algorithm 7.3:** The client then generates a search token and retrieves the corresponding encrypted documents from the encrypted document array at the server, and decrypts the results locally to filter out the fake documents. For each accessed document, the client updates the local counter keeping track of the number of times the document has been addressed (this helps generate the new write-back address for kw_i in the encrypted document array).

- **Algorithm 7.4:** Finally, the client updates its local stash with the documents retrieved in the previous step. These will be written back to the encrypted document array at the server via normal auxiliary write-backs at a later point of time. Each write-back operation involves the client randomly sampling a certain proportion of the documents and half of the lookup indices from its local stash (created over multiple search operations in the past), and writing them back to the corresponding encrypted data structures at the server. For the specific instance described in Algorithm 7.4, this fraction is set to one-half; however this can also be a parameter input to the write-back sub-routine.

7.4.3 Correctness

The scheme as described does not have perfect correctness: the addresses where various information is stored are generated with a PRF and so the client may generate repeated addresses unintentionally and overwrite the encrypted document addresses or the encrypted documents. Worse still, it is possible that the queries are adversarially controlled to maximize the failure probability.

The following theorem establishes an upper bound on the advantage of any adversary against the correctness of our scheme. The probability is upper-bounded by negligible terms which account for collisions in the outputs of the PRF used in the construction, and the security of the PRF itself.

Theorem 7.1. [Correctness of Static SWiSSSE] Let $|\mathbf{DB}|$ and $|KW\{\mathbf{DB}\}|$ denote the total number of documents and document-keyword pairs, respectively, in the database $\mathbf{DB}$, and let l denote the output length of the PRF F used in static SWiSSSE. Then the advantage of any adversary $\mathcal{A}$, which issues at most k queries, in breaking the correctness of static SWiSSSE over the database $\mathbf{DB}$ is at most:

$$\frac{\left(|\mathbf{DB}|^2 + 4t_0\,|\mathbf{DB}| + 4\,|KW\{\mathbf{DB}\}|^2 + 8t_1\,|KW\{\mathbf{DB}\}|\right)}{2^{l+1}} + \mathsf{Adv}^{PRF,|\mathbf{DB}|+2t_0}_{F,\mathcal{B}} + \mathsf{Adv}^{PRF,2|KW\{\mathbf{DB}\}|+2t_1}_{F,\mathcal{C}},$$

where $t_0 = k \cdot \max_k w\,|\mathbf{DB}(kw)|$, $t_1 = k \cdot \max_k w\,|kw\,\{\mathbf{DB}(kw)\}|$, and $\mathcal{B}$ and $\mathcal{C}$ denote probabilistic polynomial-time adversaries in independent security experiments against the PRF F.

Proof. We use standard game-hopping to reduce the correctness game G to finding a pair of collisions in a game where the adversary interacts with truly random functions. Let game G_2 be the game where the PRF used to generate the addresses for the encrypted documents is replaced by a truly random function. The number of addresses used for the encrypted documents with k queries is at most $|\mathbf{DB}| + 2k \cdot \max_k w\,|\mathbf{DB}(kw)|$, so the difference in the advantages between game G and G_2 is $\mathsf{Adv}^{PRF,|\mathbf{DB}|+2k\cdot\max_k w|\mathbf{DB}(kw)|}_{F}$.

In our static construction, for database $\mathbf{DB}$, we use $|\mathbf{DB}|$ addresses to store the encrypted documents during initialisation, which means the probability of a pair of collisions is upper-bounded by $|\mathbf{DB}|^2 \cdot 2^{-(l+1)}$.

In the subsequent write-backs, the number of active addresses in the encrypted document array is at most $|\mathbf{DB}|$ and the number of new addresses we generate is upper bounded by $\max_k w\,|\mathbf{DB}(kw)|$. This means that there are at most $|\mathbf{DB}| \cdot \max_k w\,|\mathbf{DB}(kw)|$ new potential pairs of collisions for each query. Using the birthday bound, the probability of finding a collision in the addresses of the encrypted document array in each write-back is upper bounded by $|\mathbf{DB}| \cdot \max_k w\,|\mathbf{DB}(kw)| \cdot 2^{-l}$.

Similarly, we define game G_3 be the game where the PRF used to generate the addresses for the encrypted lookup table is replaced with a truly random function. The number of addresses used for the encrypted lookup table is at most $2\,|KW\{\mathbf{DB}\}| + 2\max_k w\,|\mathbf{DB}\,\{kw\}|$, so the difference in the advantages between game G_2 and G_3 is $\mathsf{Adv}^{PRF,2KW\{\mathbf{DB}\}+2\max_k w|\mathbf{DB}\{kw\}|}_{F}$.

Using a similar argument as above, the probability of a pair of collision in the addresses for the encrypted lookup table during initialisation is at most $4\,|KW\{\mathbf{DB}\}|^2 \cdot 2^{-(l+1)}$, and the probability of a pair of collision for the encrypted lookup table for each query is at most $4\,|KW\{\mathbf{DB}\}| \cdot \max_k w\,|\mathbf{DB}\,\{kw\}| \cdot 2^l$.

Combining everything together with a union bound on all k queries, we conclude that

the failure probability of the construction is at most

$$\left(|\mathbf{DB}|^2 + 4t_0\,|\mathbf{DB}| + 4\,|KW\,\{\mathbf{DB}\}|^2 + 8t_1\,|KW\,\{\mathbf{DB}\}|\right) \cdot 2^{-(l+1)} + \mathsf{Adv}_F^{PRF,|\mathbf{DB}|+2t_0} + \mathsf{Adv}_F^{PRF,2KW\{\mathbf{DB}\}+2t_1},$$

where $t_0 = k \cdot \max_k w\,|\mathbf{DB}(kw)|$ and $t_1 = k \cdot \max_k w\,|kw\,\{\mathbf{DB}(kw)\}|$. □

7.4.4 Formal Leakage Description

We now formally describe the leakage profile for SWiSSSE with respect to static databases. Following the approach introduced by previous works on SSE (such as [53] and [36]), we use a simulation-based framework where a PPT adversary is required to distinguish between the real world (where the adversary interacts with a real execution of SWiSSSE that uses the secret key) and the ideal world (where the adversary interacts with a simulator that only has access to the described leakage profile for SWiSSSE). We say that the enumeration is provably sound if no PPT adversary can distinguish between these two worlds with non-negligible advantage over a random guess.

We formally describe the leakage of static SWiSSSE. Unlike most prior SSE constructions, the leakage function for SWiSSSE is stateful. This follows naturally from the fact that the search protocol in SWiSSSE makes abundant usage of random address accesses across the encrypted data structures at the server. We use a stateful definition to capture the leakage from such random accesses.

Leakage at Setup. At setup, the client offloads the encrypted lookup index and the encrypted document array to the server. These data structures are essentially key-value stores with pseudorandomly generated keys/addresses and values/entries that are encrypted under an IND-CPA secure encryption scheme. Hence, at setup, the server learns no information about the original database $\mathbf{DB}$ other than the number of documents in the padded database $\mathbf{DB}'$ (including both real and fake documents), and the total number of keyword-document pairs post-bucketization. Formally, we have:

$$\mathcal{L}_{\Sigma}^{\mathbf{Setup}}(\mathbf{DB}, \mathtt{G}) = (\left|\mathbf{DB}'\right|, \left|KW\left\{\mathbf{DB}'\right\}\right|, \mathbf{St}_{\mathcal{L}}),$$

where Σ denotes a concrete instance of static SWiSSSE.

Note that the leakage function is stateful; it maintains in $\mathbf{St}_{\mathcal{L}}$ a realisation of the padded database which is used later by the leakage function for the search queries.

Leakage during Searches and Write-Backs. At a high level, each search query reveals the following to the server:

1. The set of normal and auxiliary write-back addresses in the encrypted keyword lookup index corresponding to the queried keyword (but not precisely which of these are normal and which of these are auxiliary).
2. The set of addresses in the encrypted document array for the real and fake documents containing the queried keyword (but not precisely the document identifiers, or even which of the addresses correspond to real documents and the fake documents, respectively).

Similarly, each write-back operation reveals to the server the set of addresses in the encrypted keyword lookup index and the encrypted document array that are written to using content from the stash.

We capture these leakage using a probabilistic and stateful leakage function, described formally in Algorithm 7.5, again for an instance Σ of the static SWiSSSE scheme. The state of the leakage function contains a realisation of the padded database, everything in the stash of the client, and two data structures, namely the lookup history **IndHist** and the array-access history **ArrHist**, which we describe formally below.

Lookup and Array-Access Histories. We define the lookup history **IndHist** and the array-access history **ArrHist** to be lists of records corresponding to operations on the encrypted keyword lookup index and the encrypted document array, respectively. These capture the access pattern information leakaged during searches in SWiSSSE.

Formally, each lookup (respectively, array-access) record is of the form (i, b, t), where i is a unique identifier associated with the index (respectively, array) address being operated on, b is a bit which indicates whether the entry corresponds to a read operation ($b = 0$) or a write operation ($b = 1$), and t is the timestamp of the query that resulted in the operation. We also overload notation and let $\mathbf{IndHist}\,[\{i_j\}]$ and $\mathbf{ArrHist}\,[\{i_j\}]$ be the sets of all records corresponding to a given set $\{i_j\}$ of index/array addresses, i.e.,

$$\mathbf{IndHist}\,[\{i_j\}] = \{(i,b,t) \mid (i,b,t) \in \mathbf{IndHist} \wedge i \in \{i_j\}\}\,.$$

$$\mathbf{ArrHist}\,[\{i_j\}] = \{(i,b,t) \mid (i,b,t) \in \mathbf{ArrHist} \wedge i \in \{i_j\}\}\,.$$

The following theorem formalizes the security of SWiSSSE.

Theorem 7.2 (Security of Static SWiSSSE)**.** *Let $\mathcal{L}_{\Sigma}^{\mathbf{Setup}}$ and $\mathcal{L}_{\Sigma}^{\mathbf{Srch}}$ be the leakage functions defined above. Then the instance Σ of the static* SWiSSSE *scheme is $\left(\mathcal{L}_{\Sigma}^{\mathbf{Setup}}, \mathcal{L}_{\Sigma}^{\mathbf{Srch}}\right)$-secure.*

Proof. The proof proceeds with a hybrid argument. Let **DB** be a database and $\mathbf{q}_1, \ldots, \mathbf{q}_k$ be the set of single-keyword queries with the leading keywords $kw_1, \ldots, kw_k$. Let $\mathsf{Adv}_F^{PRF,t}(\lambda)$ be the PRF advantage of the PRF F with at most t evaluations used in the construction and $\mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$ be the IND-CPA advantage of the scheme Σ' used for lookup address encryption and document content encryption. Finally, we assume the plaintext of the lookup addresses is padded to ℓ_0 and the plaintext of the document contents is padded to ℓ_1. The simulator has access to ℓ_0 and ℓ_1 as they are public parameters.

(**Game 0.**) Let the real execution of the scheme on the database **DB** with queries $\mathbf{q}_1, \ldots, \mathbf{q}_k$ be game G_0. Then we have that for any adversary $\mathcal{A}$, $\Pr[\mathbf{Real}_{\Sigma,\mathcal{A}}(1^\lambda) = 1] = \Pr[G_0 = 1]$.

(**Game 1.**) We define game G_1 by letting the leakage function generate the padded database and replace the addresses in the encrypted lookup index and the encrypted documents with outputs generated from a truly random function **RF** with output length l. We omit the conversion between an integer to a string of appropriate length in the use of the random function for simplicity. In addition, the encryptions of the document addresses and the documents themselves are replaced with encryptions of zeros of length ℓ_0 and ℓ_1 respectively.

Algorithm 7.5 Static SWiSSSE: Leakage Function for Searches and Write-Backs

procedure $\mathcal{L}_{\Sigma}^{\mathbf{Srch}}(\mathbf{q}, \mathbf{St}_{\mathcal{L}})$
  $(\mathbf{Srch}, kw) \leftarrow query$
  $\mathtt{I}, \mathtt{A}, \mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr}, \mathbf{IndHist}, \mathbf{ArrHist} \leftarrow \mathbf{St}_{\mathcal{L}}$
  /* *Leakage from the query tokens* */
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 2 * \mathtt{KWCtr}[\bar{kw}_1]), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 2 * \mathtt{KWCtr}[\bar{kw}_1] + 1), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathtt{KWCtr}[kw] \leftarrow \mathtt{KWCtr}[kw] + 1$
  /* *Leakage from document array access* */
  $L \leftarrow \mathtt{I}[kw]$
  while $|L| < 2 \cdot \mathtt{Clt}.\mathtt{G}(kw)$ **do**
    $id \leftarrow \mathbf{Rand}(|\mathtt{A}|)$
    if $id \notin \{\mathsf{id}(doc) \mid doc \in \mathtt{A}'\}$ **then**
      $L \leftarrow L \cup id$
  $\mathtt{ArrCtr}[L] \leftarrow \mathtt{ArrCtr}[L] + 1$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(l, \mathtt{ArrCtr}[l]), 0, k) \mid l \in L\}$

  /* *Leakage from write-back* */
  $UI \leftarrow \mathtt{I}'.\mathbf{pop}(|\mathtt{I}'|/2)$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(i, 1, k) \mid i \in UI\}$
  $UA \leftarrow \mathtt{A}'.\mathbf{Pop}(\lfloor |UA|/2 \rfloor)$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(\mathsf{id}(doc), \mathtt{ArrCtr}[\mathsf{id}(doc)]), 1, k) \mid (\{w_i, j_i, b_i\}, doc) \in UA\}$

  /* *Update the stash for consistency* */
  $\mathtt{I}' \leftarrow \mathtt{I}' \cup \mathbf{Index}(UA, \mathtt{KWCtr})$
  $\mathtt{A}' \leftarrow \mathtt{A}' \cup \mathbf{Retrieve}(\mathtt{A}[L], kw)$
  $\mathbf{St}_{\mathcal{L}} \leftarrow (\mathtt{I}, \mathtt{A}, \mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr}, \mathbf{IndHist}, \mathbf{ArrHist})$
  Return $(\mathbf{IndHist}, \mathbf{ArrHist}), \mathbf{St}_{\mathcal{L}}$

procedure Index$(UA, \mathtt{KWCtr})$
  $\mathtt{I} \leftarrow \{\}$
  for $(\{kw_i, j_i, b_i\}, doc) \in UA$ **do**
    $\mathtt{I} \leftarrow \mathtt{I} \cup \{\mathbf{T}(kw, j, 2 * \mathtt{KWCtr}[kw] + b) \mid (kw, j, b) \in \{kw_i, j_i, b_i\}\}$
  Return I

procedure Retrieve$(\mathtt{A}, kw)$
  $\mathtt{A}' \leftarrow \{\}$
  for $(\{kw_i, j_i\}, doc) \in \mathtt{A}$ **do**
    $\mathtt{A}' \leftarrow \mathtt{A}' \cup \{kw_i, j_i, (kw_i = kw)\}, doc)$
  Return $\mathtt{A}'$

The number of addresses that needs to be generated in the initialisation step is equal to $t_0 = 2\sum_{kw} \mathtt{G}(kw) + |\mathbf{DB}|$, and an equal number of encryptions need to be created, so the difference in advantages between G_0 and G_1 is upper-bounded by $\mathsf{Adv}_F^{PRF,t_0} + t_0 \cdot \mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$.

(**Game 2.**) In game G_2, we replace the single-keyword query algorithm with a simulator that has access to the output of the leakage function only. As before, the addresses are generated by applying the truly random function **RF** on the indices provided by the

Algorithm 7.6 Game G_1 (static construction). Only the setup step is changed.

procedure $\texttt{Clt}.\mathbf{Setup}(\mathbf{DB})$
  $(N, p, \mathbf{St}_{\mathcal{L}}) \leftarrow \mathcal{L}_{\Sigma}^{\mathbf{Setup}}(\mathbf{DB}, \texttt{G})$
  $\texttt{EI}, \texttt{EA} \leftarrow [\,]$
  /* *Generate the encrypted documents* */
  for $i = 0, \ldots, N-1$ **do**
    $\texttt{EA}.\mathbf{Insert}(\mathbf{RF}(2i), \mathbf{Enc}(0^{l_0}))$
  /* *Generate the encrypted document addresses* */
  for $i = 0, \ldots, 2p$ **do**
    $\texttt{EI}.\mathbf{Insert}(\mathbf{RF}(2i+1), \mathbf{Enc}(0^{l_1}))$
  Send $(\texttt{EI}, \texttt{EA})$ to the server

Algorithm 7.7 Game G_2 (static construction).

procedure $\texttt{Clt}.\mathbf{Srch}(\mathbf{q})$
  $(\mathbf{IndHist}, \mathbf{ArrHist}), \mathbf{St}_{\mathcal{L}} \leftarrow \mathcal{L}_{\Sigma}^{\mathbf{Srch}}(\mathbf{DB}, \mathbf{q}, \mathbf{St}_{\mathcal{L}})$

  /* *Encrypted document array address retrieval* */
  $L \leftarrow \{\}$
  $t' \leftarrow$ the number of single-keyword queries executed
  for $i \in \{i \mid (i, b, t) \in \mathbf{IndHist}, b = 0, t = t'\}$ **do**
    $L \leftarrow L \cup \mathbf{RF}(2i+1)$
  Send L to the server

  /* *Encrypted document retrieval* */
  $L \leftarrow \{\}$
  for $i \in \{i \mid (i, b, t) \in \mathbf{ArrHist}, b = 0, t = t'\}$ **do**
    $L \leftarrow L \cup \mathbf{RF}(2i)$
  Send L to the server

  /* *Write-back* */
  $UI, UA \leftarrow \{\}$
  for $i \in \{i \mid (i, b, t) \in \mathbf{IndHist}, b = 1, t = t'\}$ **do**
    $UI \leftarrow UI \cup (\mathbf{RF}(2i+1), \mathbf{Enc}(0^{l_0}))$
  for $i \in \{i \mid (i, b, t) \in \mathbf{ArrHist}, b = 1, t = t'\}$ **do**
    $UA \leftarrow UA \cup (\mathbf{RF}(2i), \mathbf{Enc}(0^{l_1}))$
  Send (UI, UA) to the server

leakage function. The encrypted documents and the encrypted document addresses are generated with encryptions of zeros of appropriate length. The way the addresses are generated is consistent with game G_1 as the only difference between the two games is that the leakage function is responsible for randomising the write-backs.

The number of addresses the algorithm has to generate is upper-bounded by $t_1 = 2\sum_i \texttt{G}(KW(\mathbf{q}_i)) + 2\sum_i |\mathbf{DB}(KW(\mathbf{q}_i))|$, and the number of encryptions needs to be created is upper-bounded by the same t_1. This means the difference in advantages between G_1 and G_2 is upper-bounded by $\mathsf{Adv}_F^{PRF,t_1} + t_1 \cdot \mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$.

(**Conclusion**.) By combining the two games above, we see that the difference in advantages between G_0 and G_2 is at most $\mathsf{Adv}_F^{PRF,t_0+t_1} + (t_0+t_1) \cdot \mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$. □

7.5 Cryptanalysis of Static SWiSSSE

Having established the precise leakage of static SWiSSSE, we now turn to its cryptanalysis. For a corresponding analysis of dynamic SWiSSSE, see Section 7.6. We begin by briefly arguing that SWiSSSE is resilient to most well-known cryptanalytic techniques in the literature, including the query recovery attacks proposed in [94, 33], the document recovery attacks proposed in [33] and the file-injection attacks proposed in [200]. We then present a highly refined cryptanalytic technique based on system-level leakage with a much stronger attack model than assumed by these existing attacks, and demonstrate that for appropriately chosen bucketization parameters, SWiSSSE manages to resist even this attack.

7.5.1 Cryptanalysis with Known Attacks

Query and Document Recovery Attacks. Existing query recovery attacks (such as those proposed in [94] and [33]) and document recovery attacks (such as those proposed in [33]) typically rely on a combination of three kinds of deterministic leakage – volume/frequency leakage (i.e., the adversary deterministically recovers the number of documents matching a given query), document access pattern leakage (i.e., the adversary deterministically recovers which (encrypted) document identifiers are accessed across two or more queries), and query equality leakage (i.e., the adversary recovers whether two or more queries correspond to the same underlying keyword).

Through the use of keyword bucketization, SWiSSSE makes it possible to suppress volume leakage sufficiently to prevent these attacks (we subsequently discuss in detail the appropriate bucket-sizes needed). Similarly, the use of delayed pseudorandom write-backs corresponding to each query prevents the adversary from deterministically learning the document access patterns and the query equality patterns across multiple queries. In summary, existing cryptanalytic techniques for query and document recovery cannot be applied directly to cryptanalyse the leakage profile for SWiSSSE.

File-Injection Attacks. File-injection attacks [200] are an extremely powerful class of query recovery attacks in which the adversary has the power to inject maliciously crafted files into the database. The adversary uses the occurrences of these files in query outputs to identify the keyword(s) underlying a given query. Once again, query recovery via file-injection relies crucially on the document access pattern leakage. In particular, it requires the adversary to identify which of the maliciously crafted files appear in the outcome of a given query (either from accesses to the search index or to the encrypted document array).

In SWiSSSE, this leakage is not available during search queries as the document identifiers matching a given query are never revealed in the clear, and the locations of documents in the encrypted document array change with every write-back operation (making it hard for the adversary to trace the occurrence of malicious documents across queries). Hence, file-injection attacks cannot be applied directly to cryptanalyse the leakage profile for SWiSSSE.

7.5.2 Cryptanalysis with A Refined Attack

To examine security of SWiSSSE fully, we develop a new co-occurrence pattern leakage-abuse attack which aims at query recovery, by refining our attacks in Chapter 6. We have also made the attack as "unfriendly" for the scheme as possible. In particular, we have:

- given the adversary the exact database as auxiliary information as opposed to a noisy distribution of it;
- used all keywords in the keyword universe as queried keywords;
- given the adversary more leakage than what SWiSSSE leaks.

Even then, we show that with appropriate choice of bucketization strategy, SWiSSSE is secure against our refined attack.

Attack Assumption. Let $M : \{1, \ldots |kw(\mathbf{DB})|\}^2 \rightarrow \mathbb{N}$ to be a two-dimensional co-occurrence matrix that maps pairs of keywords to the number of documents containing them both. Formally, we have

$$M_{i,j} = |\mathbf{DB}(kw_i) \cap \mathbf{DB}(kw_j)| \,.$$

It is important to note that this matrix is defined with respect to the original database (before padding/bucketization). We assume here that at the beginning of the attack, the adversary has an exact copy of matrix M. This is a very strong assumption, because it is not obvious how the attacker might obtain this information.

We also simulate an observed co-occurrence matrix $\bar{M}$ as follows: let M' be the co-occurrence matrix for the padded version of the database. We simulate $\bar{M}_{i,j}$ as a value sampled according to a binomial distribution as follows:

$$\bar{M}_{i,j} \leftarrow \mathbf{Binom}(M'_{i,j}, 1/q),$$

where $1/q$ is a parameter of SWiSSSE denoting the fraction of the local stash flushed out by the client in each write-back operation (we use $q = 2$ for our cryptanalysis experiments in keeping with the description in Section 7.4.2). We assume that the attacker has access to a randomly permuted version of $\bar{M}$, which we abuse the notion $\bar{M}$ to denote it.

Again, this is a very strong assumption, because inferring the matrix $\bar{M}$ from the leakage profile of SWiSSSE is highly non-trivial. As already discussed, the intermittent and pseudorandom nature of the write-back operations corresponding to each search query makes it very hard for the attacker to identify if the same document appears across multiple search queries. Even in an ideal scenario where the client flushes out its stash after every q search queries, the attacker can guess any given entry $\bar{M}$ with probability at most $1/q$. In an actual implementation, the client only partially flushes its stash each time and writes-back documents from multiple search queries in a single batch, making such inferences even harder.

We note that search-pattern leakage is necessary to construct the above co-occurrence matrix. This is given to the adversary in our attack but SWiSSSE does not leak it with certainty.

Attack Strategy. Given auxiliary information M and observed co-occurrence matrix $\bar{M}$, the goal of the adversary is to identify a permutation $P : \{1, \ldots, |kw(\mathbf{DB})|\} \to \{1, \ldots, |kw(\mathbf{DB})|\}$ which assigns each queried keyword in $\bar{M}$ to keywords in the database.

Similar to the attacks in Chapter 6, we build a probabilistic model of the observed co-occurrence matrix $\bar{M}$, derive the likelihood function for any observed co-occurrence matrix, and use simulated annealing to find the best permutation P which maximizes the likelihood function.

Derivation of the Distribution of the Co-occurrence Matrix There are two steps in the derivation of the distribution of the co-occurrence matrix. In the first step, we derive the distribution of the co-occurrence matrix of the *padded database*, that is, the database after the keywords are bucketized and fake keywords are added. Secondly, we derive the distribution of the observed co-occurrence matrix.

Call the co-occurrence matrix of the padded database M'. Recall that fake keywords are added to the database to achieve bucketization. This creates fake co-occurrence counts in M'. Without loss of generality, let us focus on co-occurrence count $M'_{i,j}$, for $i \neq j$. There are three ways for $M'_{i,j}$ to get fake co-occurrence counts. Firstly, it can be the case that there is a document with the keyword kw_i and keyword kw_j is added to it. Secondly, it can be the case that both the keywords kw_i and kw_j are added to the same document that did not have them before. Finally, it can be the case that there is a document with keyword kw_j and keyword kw_i is added to it. Let d be the total number of documents after padding, then the number of co-occurrence counts generated by each way can be expressed as a hypergeometric distribution with parameters,

$$\begin{aligned}
&(d - M_{i,j}, M_{i,i} - M_{i,j}, \bar{M}_{j,j} - M_{j,j}),\\
&(d - M_{i,j}, \bar{M}_{i,i} - M_{i,i}, \bar{M}_{j,j} - M_{j,j}),\\
&(d - M_{i,j}, M_{j,j} - M_{i,j}, \bar{M}_{i,i} - M_{i,i})
\end{aligned}$$

respectively. And $M'_{i,j}$ (for $i \neq j$) is simply the sum of $M_{i,j}$ and the realisations of the three hypergeometric random variables. The diagonal entries of M' are simply the size of the buckets for which the keywords are assigned to.

We are now ready to derive the distribution of the observed co-occurrence. Here, the co-occurrence counts are obtained from the number of shared documents written back by the previous flush operation and the current query – documents in this intersection have a high change of containing the keyword queried in the previous iteration and in the current one. In our attack, we assume that half of the documents ($q = 2$) from the stash are written back to the server, which means the observed co-occurrence count $\bar{M}_{i,j}$ (for $i \neq j$) can be expressed as:

$$\bar{M}_{i,j} \leftarrow \mathbf{Binom}(M'_{i,j}, 0.5).$$

The diagonal entries of M are simply the size of the buckets for which the keywords are assigned to.

Derivation of the Likelihood Function. The likelihood function for this attack can be derived in the same way as that in Section 6.4.4. The only difference between the derivations is that M here refers to the exact database whereas M in Section 6.4.4 refers to a (noisy) distribution of it.

The likelihood is decomposed in the exact same way so the details are omitted here.

Briefly, we can express the likelihood as:

$$\begin{aligned}&\mathbf{L}\left[P \mid \bar{M}, M\right]\\ =&\sum_{M' \in \mathcal{N}^{N \times N}} \mathsf{Pr}\left[\bar{M} \mid M', P\right] \mathsf{Pr}\left[M' \mid M\right],\end{aligned}$$

where N is the number of keywords and $\mathcal{N}^{N \times N}$ is all N by N natural number valued matrices.

The first term in the likelihood decomposition is the probability of observing the co-occurrence matrix M given co-occurrence matrix M' of the padded database and permutation P. It can be decomposed as:

$$\begin{aligned}&\mathsf{Pr}\left[\bar{M} \mid M', P\right]\\ =&\prod_{i<j} \mathsf{Pr}\left[\bar{M}_{i,j} \mid M'_{i,j}, P\right]\\ =&\prod_{i<j} \mathsf{Pr}\left[\mathbf{Binom}(M'_{i,j}, 0.5) = \bar{M}_{P(i),P(j)}\right].\end{aligned}$$

The second term in the likelihood decomposition is the probability of observing M' as the co-occurrence matrix of the padded database given M. This probability can be expressed as:

$$\begin{aligned}&\mathsf{Pr}\left[M' \mid M\right]\\ =&\prod_{i<j} \mathsf{Pr}\left[M'_{i,j} \mid M_{i,j}\right]\\ =&\prod_{i<j} \mathbf{Prob}\Big[M_{i,j} + \mathbf{HyperGeom}(|\mathbf{DB}| - M_{i,j}, M_{i,i} - M_{i,j}, \bar{M}_{j,j} - M_{j,j})\\ &+\mathbf{HyperGeom}((d - M_{i,j}, \bar{M}_{i,i} - M_{i,i}, \bar{M}_{j,j} - M_{j,j})\\ &+\mathbf{HyperGeom}(d - M_{i,j}, M_{j,j} - M_{i,j}, \bar{M}_{i,i} - M_{i,i}) = M'_{i,j}\Big].\end{aligned}$$

Computational Approximations and Optimisations. We use the same computational approximations and optimisation techniques we have described in Section 6.4.4.

7.5.3 Experimental Results of the Refined Attack

We run our refined attack against the Enron email corpus [194]. See Section 6.1.1.2 for a description of the dataset. We generate co-occurrence matrices from 400000 emails, with keywords from different frequency ranges. Specifically, we arranged the keywords in decreasing order of frequency and chose 800 of the most frequent keywords and 800 of the keywords from the 95-th to 75-th percentile frequencies respectively. We constructed the co-occurrence matrices M and $\bar{M}$ with varying bucket sizes (in the range of 50 to 400) for each of these keyword sets. We repeated the attack for 100 times with freshly generated $\bar{M}$ matrices, and the average recovery rates are reported in Figure 7.3.

We observe that the keyword recovery rates do not follow a linear trend. For the most frequent keywords, we see very high query recovery rates, but as soon as we get to 85-th

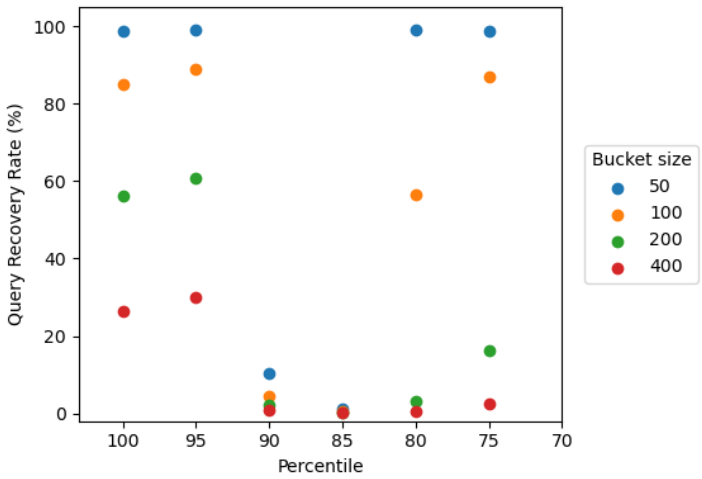


Figure 7.3: The average recovery rate over 100 executions of our simulated-annealing based cryptanalytic attack for keywords in various frequency percentiles. The recovery rate is measured as the fraction of keywords guessed correctly across all buckets in a single execution of the attack. The keyword frequencies of the buckets are set to double of the maximum frequencies of the keywords in the given buckets.

to 90-th percentile, the query recovery rate drops to almost zero. But as the keyword frequency decreases even further into the 80-th percentile, we begin to see significant query recovery rate for small bucket sizes again.

This phenomenon is likely due to the interaction between real and fake co-occurrence counts in different frequency ranges. Concretely, for the Enron dataset with 400000 documents, the two most frequent keywords occurred 28919 and 28587 times respectively. So we expect about 2114 fake co-occurrence counts between these two keywords due to random padding. on the other hand, most of the real co-occurrence counts between these two keywords and other keywords are in the range of 4000 to 9000, and the amount of randomness in the fake co-occurrence counts is insufficient to hide the real ones.

For keywords with frequencies in the 85-th to 90-th percentile, the real frequencies are on the order of 10^3, and one can expect fake co-occurrence counts on the order of tens. The later is exactly the range of values one expect to find the co-occurrence counts in, which means that the noise is just right to mask the real co-occurrence counts.

Finally, for keywords with frequencies in the 80-th percentile and lower, the real frequencies are on the order of 10^2, and one does not expect any fake co-occurrence counts between these keywords. This means that our attack is essentially trying to match the real co-occurrence count to itself (with padded keyword frequencies of course), so it is not at all surprising to see high query recovery rates for small bucket sizes. We argue that this is not a weakness of our construction as a real-world adversary is likely going to get a noisy auxiliary co-occurrence matrix as opposed to the perfect one. In that case, an attack on these keywords will be much harder as the co-occurrence counts with these keywords are very small and contain very little information. In fact, our attacks have shown that using larger bucket sizes on these keywords is already enough to create enough ambiguity.

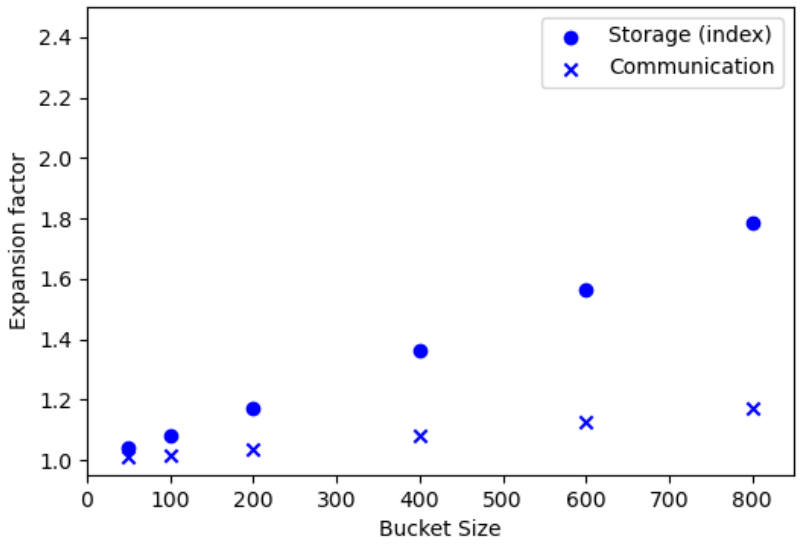


Figure 7.4: Overheads incurred by different bucket sizes on the Enron email corpus. The experiments are conducted with fixed bucket sizes.

7.5.4 Discussion

Implications for Bucketization Strategy. It is natural ask what these cryptanalysis results tell us about the choice of the bucket size we have discussed in Section 7.3. It is clear that a bucket size of 200 is sufficient for all keywords except the most frequent ones. We can simply use a larger bucket size for those keywords (e.g. a bucket size of 400 for the most frequent 10% keywords) to achieve better query privacy.

We stress that the attack model used for the cryptanalysis is a lot stronger than what SWiSSSE permits. And one can expect much lower query recovery rates from real-world attacks.

Security Versus Efficiency Trade-offs. Our experiments also reveal some interesting insights into the security versus efficiency trade-offs associated with choosing the bucket size. For example, we saw earlier that a bucket size of 50 for the most frequent keywords leads to almost 100% recovery, while a bucket size of 400 reduces this to around 30%. But what is the implication of using a larger bucket-size on the storage requirements and communication bandwidth requirements for SWiSSSE?

In Figure 7.4, we demonstrate through concrete figures how variations in bucket sizes affect the storage and communication overheads of our construction. Here, the storage overhead only applies to the index as the number of documents is unaffected by bucketization in SWiSSSE. In general, the total number of (real and fake) keyword-document pairs grows essentially linearly with the bucket size. This in turn implies that storage overhead of the search index also grows linearly with bucket size.

Interestingly, the growth in overhead is more gradual when compared to the fall in recovery rate. When the initial bucket size varies from 50 to 400, the storage overhead varies between 1.04× and 1.36×. On the other hand, as demonstrated earlier, the keyword

recovery rate falls from 100% to below 30%. This indicates that it is preferable to opt for a larger bucket size as long as the user can afford it, since it provides significantly stronger resistance to cryptanalysis while incurring only moderately larger overheads.

Setting Parameters in Practice. Our cryptanalysis experiments seem to suggest that for a given database, it is preferable to carefully optimise the bucketization strategy and the bucket size parameters for SWiSSSE to attain a desirable trade-off between security and efficiency. In practice, this is not easily deployable. Recall however that we assumed a very optimistic attack setting where the attacker has access to refined co-occurrence leakage. In practice, the leakage profile of SWiSSSE is significantly more "noisy"; it is not at all obvious how the attacker might obtain access to such refined leakage from a real implementation. Hence, we suggest that a pragmatic choice of bucket size 400 for the 2400 most frequent keywords and 200 for the remaining ones, which should be adequate for typical applications.

At the same time, we acknowledge the need for further cryptanalysis of the leakage profile of SWiSSSE and welcome such studies from the community.

7.6 Dynamic SWiSSSE

7.6.1 Overview

In this section, we present an overview of how we extend SWiSSSE to handle dynamic databases. We refer the reader to Section 7.6.2 for the detailed formal description.

We consider two kinds of updates to the database – document insertion and document deletion; a document update can be simulated via: (a) a deletion operation on the old document, followed by (b) an insertion operation on the modified document. We first present a simple idea for handling document insertions. At a high level, we use a technique similar to the auxiliary write-backs used in our static construction. This incurs some undesirable leakage, which we address subsequently.

Handling Insertions—Simple Version. When a document doc_ℓ is to be inserted, the client simply schedules: (a) a normal document write-back for doc_ℓ targeting a set of "insert write-backs" for every keyword $kw_i \in doc_\ell$. As with auxiliary write-backs, insert write-backs target a separate set of addresses to avoid any correlation with prior write-backs (normal and auxiliary) corresponding to the same keyword. More concretely, we now generate three separate sets of addresses for normal, auxiliary and update write-backs involving the same keyword:

$$\mathsf{addr}_{\text{norm}}(kw_i, doc_\ell, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(3 * \mathsf{cnt}_{kw_i})),$$

$$\mathsf{addr}_{\mathbf{aux}}(kw_i, doc_\ell, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(3 * \mathsf{cnt}_{kw_i} + 1)),$$

$$\mathsf{addr}_{\text{insert}}(kw_i, doc_\ell, \mathsf{cnt}_{kw_i}) = F(K, kw_i||j||(3 * \mathsf{cnt}_{kw_i} + 2)),$$

where F is a PRF with key K, j is a counter that runs from 0 to $|\mathbf{DB}(kw_i)| - 1$ (where $\mathbf{DB}(kw_i)$ denotes the set of documents containing keyword kw_i), and cnt_{kw_i} is a per-key word counter held in the client's stash which records how many times kw_i has appeared in search *and* insertion queries.

In other words, during the time interval between the t^{th} and $(t+1)^{\text{th}}$ queries on keyword kw_i, we use *three* sets of write-back addresses – $\{\mathsf{addr}_{\text{norm}}(kw_i, doc_\ell, j)\}$ for normal write-backs, $\{\mathsf{addr}_{\mathbf{aux}}(kw_i, doc_\ell, j)\}$ for auxiliary write-backs, and $\{\mathsf{addr}_{\text{insert}}(kw_i, doc_\ell, j)\}$ for insert write-backs. The insert write-backs happen intermittently and can be randomly interspersed with normal and auxiliary write-backs involving other keywords and documents.

During a search query involving kw_i, the client now requests the server to access all three sets of write-back addresses – normal, auxiliary and insert – in the keyword lookup index. The entries corresponding to the normal and insert write-back addresses allow the client to recover the pointers to already existing documents and freshly inserted documents, respectively, that contain kw_i. The entries corresponding to the auxiliary write-back addresses allow the client to identify if any of these pointers have been updated subsequently due to searches involving other keywords. Thus, search correctness is ensured. Finally, as before, we use additional pointers to fake documents to hide the exact frequency of the keyword kw_i, and reveal its bucket size instead.

Leakage. The solution outlined above leaks that a new document has been inserted: when the client executes a normal document write-back operation for the newly inserted document doc_ℓ, the total number of actual and dummy addresses in the encrypted document array increases by one. While this leakage is currently incurred by all existing dynamic SSE schemes, it has some repercussions with respect to file injection attacks [200]. For this attack vector to work, the adversary needs to infer exactly when an insert operation corresponding to a maliciously constructed file occurs, as well as the effect of this insertion on subsequent keyword search operations.

This motivates hiding the occurrences of inserts from the server, and hence, masking the aforementioned leakage. We describe how to achieve this next.

Handling Insertions. An effective way to mask when a document is inserted, is to avoid creating a fresh entry in the encrypted document array. Instead, we simply convert an (already existing) dummy entry into a real one.

Concretely, to insert a fresh document doc_ℓ, the client first identifies a "leading keyword" kw^* in doc_ℓ. We assume without loss of generality that kw^* is the keyword in doc_ℓ with the smallest occurrence frequency in the database. Next, the client issues a search query on kw^* and retrieves a list of pointers to real and dummy locations in the document array. To insert the new document, the client schedules a normal document write-back targeting one of the dummy addresses, as opposed to a newly generated address. The insert write-backs are scheduled exactly as described in the simple version above, except they now encapsulate a pointer to the dummy address as opposed to some newly generated address.

Handling Deletions. Finally, deletions are handled in a manner that is complementary to the insertion procedure described above. Namely, when a document is to be deleted, we convert the real entry corresponding to this document in the document array into a dummy entry with some garbage ciphertext. More concretely, the client again issues a search query on kw^*, and schedules a dummy document write-back targeting the address corresponding to the document to be deleted. The insert write-backs are scheduled exactly as for the insert operations, except they now encapsulate pointers to random addresses in the document array.

Algorithm 7.8 Dynamic SWiSSSE.**Setup**

procedure Clt.**Setup**(**DB**)
  /* *Generate fake documents* */
  $\mathbf{DB}' \leftarrow$ **Fake_Doc_Gen**(**DB**, Clt.G)
  $\texttt{Clt}.N \leftarrow |\mathbf{DB}'|$
  $\texttt{EI}, \texttt{EA} \leftarrow \{\}$
  for $i = 1, \ldots, |\mathbf{DB}'|$ **do**
    /* *Get the set of keywords with counters* */
    $x \leftarrow \{(kw, \texttt{Clt.KWCtr}[kw]) \mid kw \in KW(\mathbf{DB}'[i])\}$
    /* *Update the lookup index* */
    for $kw \in KW(\mathbf{DB}'[i])$ **do**
      $\texttt{EI} \leftarrow \texttt{EI} \cup (F(kw||\texttt{Clt.KWCtr}[kw]||0), \mathbf{Enc}(\mathsf{id}(\mathbf{DB}'[i])))$
      $\texttt{Clt.KWCtr}[kw] \leftarrow \texttt{Clt.KWCtr}[kw] + 1$
    /* *Insert the encrypted document* */
    $\texttt{EA} \leftarrow \texttt{EA} \cup (F(i||0), \mathbf{Enc}(x||\mathbf{DB}'[i]))$
  /* *Reset the keyword counter* */
  for $kw \in KW(\mathbf{DB}')$ **do**
    $\texttt{Clt.KWCtr}[kw] \leftarrow 0$
  /* *Initialise the stash* */
  Clt.I.**init**()
  Clt.A.**init**()
  Send (EI, EA) to the server

procedure Svr.**Setup**(EI, EA)
  Svr.EI.**init**()
  Svr.EA.**init**()
  Svr.EI.**put**(EI)
  Svr.EA.**put**(EA)

Note that in the above strategy, there is the possibility that we run out of dummy addresses in the document array after a certain number of insert operations. For simplicity of presentation and analysis, we implicitly assume a cap (determined at setup) on the maximum number of new document insertions supported by the system. We refer the reader to Section 7.6.2 for a more detailed discussion on how to generalize the above proposal to support an uncapped number of insertions.

7.6.2 Formal Description of Dynamic SWiSSSE

We now translate the informal presentation of our approach into a formal description of the various protocols involved in dynamic SWiSSSE.

Setup. The setup procedure (Algorithm 7.8) for the dynamic construction is very similar to the static one. The only differences are that the client has to initialise an array Clt.InsCtr to keep track of the number of insertions for each keyword since the last time they have been queried as the leading keywords. We omit the key generation part of the algorithm as that is identical to that of the static scheme.

SWiSSSE.{**Srch**, **Insert**, **Delete**}. We now describe the keyword query, insert and

Algorithm 7.9 Dynamic SWiSSSE.{**Srch**, **Insert**, **Delete**}: Encrypted Document Array Address Retrieval

1: **procedure** Clt.**TokenGen**(kw)
2: $L \leftarrow \{\}$
3: **for** $j \in 0, \ldots,$ Clt.G(kw) $- 1$ **do**
4: $L \leftarrow L \cup \{F(kw||j||3 * \texttt{Clt.KWCtr}[kw])\}$
5: $L \leftarrow L \cup \{F(kw||j||3 * \texttt{KWCtr}[kw] + 1)\}$
6: $L \leftarrow L \cup \{F(kw||j||3 * \texttt{KWCtr}[kw] + 2)\}$
7: /* *Roll forward the counter for the next query* */
8: $\texttt{Clt.KWCtr}[kw] \leftarrow \texttt{Clt.KWCtr}[kw] + 1$
9: Send L to the server

10: **procedure** Svr.**Index_Lookup**(L)
11: Send Svr.EI.**get**(L) to the client

delete procedures for dynamic SWiSSSE. For ease of representation, these procedures broken up into smaller sub-routines described subsequently.

Encrypted Document Array Address Retrieval. This sub-routine is the same for a search query, document insertion or document deletion, and is described in Algorithm 7.9, and is very similar to the corresponding sub-routine for document array address retrieval in the static version of SWiSSSE.**Srch**, except that the client now fetches three sets of addresses - normal, auxiliary and insert. For document insertion, the client simply queries the least frequent keyword in the document he wants to insert. For document deletion, the client queries the least frequent keyword in the document he wants to delete. As the index for the inserted keywords are stored by the server, the client has to compute some additional virtual addresses to retrieve the documents.

Encrypted Document Retrieval. The sub-routine is again identical for a search query, document insertion and document deletion, and works in the same way as the corresponding sub-routine for the static version of SWiSSSE (see Algorithm 7.3 in Section 7.4.2 for the details of how this sub-routine works).

The final set of sub-routines are the write-back sub-routines corresponding to search queries, insertions and deletions. Unlike the previous sub-routines, write-backs are executed differently for each query type. We describe these next.

Write-Back for search query. The write-back sub-routine under dynamic SWiSSSE.**Srch** is described in Algorithm 7.10. Technically, it is very similar to that under static SWiSSSE.**Srch** (Algorithm 7.4, Section 7.4.2), except that the client has to perform some maintenance on the lookup index for the queried keyword to relocate the addresses for the document insertions to the ones used for fake documents.

More explicitly, recall that during the encrypted document array address retrieval phase, we have obtained all the normal write-back addresses of the form $F(kw||j||3*\texttt{KWCtr}[kw])$, the auxiliary write-back addresses of the form $F(kw||j||3 * \texttt{KWCtr}[kw] + 1)$ and insertion write-back addresses of the form $F(kw||j||3 * \texttt{KWCtr}[kw] + 2)$. Our goal is to remove the additional insertion addresses of the form $F(kw||j||\texttt{KWCtr}[kw] + 2)$ by making use of the fake documents that contain the keyword kw. In terms of the documents, this means for each newly inserted document, we find a fake document that contains kw, remove

Algorithm 7.10 Dynamic SWiSSSE.**Srch**: Write-Back Sub-Routine

procedure Clt.**Write_Back_Keyword_Query**$(M, \bar{kw})$
  Replace the lookup addresses of the newly inserted documents which contain $\bar{kw}$ with the addresses used for the fake documents.
  $UA \leftarrow \{\}$
  /* *Get random documents from the stash* */
  $D \leftarrow$ `Clt.A`.**pop**($|$`Clt.A`$|$)
  for $(\{(kw_i, j_i, b_i)\}, doc) \in UA$ **do**
    /* *Encrypt the new document addresses and documents* */
    $UA \leftarrow UA \cup \{(F(\mathsf{id}(doc)||$`Clt.ArrCtr`$[\mathsf{id}(doc)]),$ **Enc**$(\{(kw_i, j_i)\}||doc))\}$
    /* *Update the stash for the lookup index* */
    for $(kw, j, b) \in \{(kw_i, j_i, b_i)\}$ **do**
      `Clt.I`.**put**$((F(kw||j||3 *$ `Clt.KWCtr`$[kw] + b),$ **Enc**$(\mathsf{id}(doc))))$
  /* *Decrypt the documents retrieved and insert them into the document array* */
  `Clt.A`.**put**(**Dec**(M))
  Send (`Clt.I`.**pop**($\lfloor|$`Clt.I`$|/2\rfloor), UA$)

procedure Svr.**Write_Back**$((UI, UA))$
  `Svr.EI`.**put**(UI)
  `Svr.EA`.**put**(UA)

the keyword from the fake document, and allocate it to the newly inserted document. We omit the low-level details of the procedure for readability.

Write-Back for Document Insertion. The write-back sub-routine under dynamic SWiSSSE.**Insert** is described in Algorithm 7.11. Technically, it is essentially identical to the corresponding sub-routine under dynamic SWiSSSE.**Srch** except that the client has to insert the document locally. This is done by scanning the query response for fake documents, and replace one of them by the document that is to be inserted. The keyword pointers are updated so as to maintain correctness of future searches.

Write-Back for Document Deletion. The write-back sub-routine under dynamic SWiSSSE.**Delete** is described in Algorithm 7.12. Technically, it is again essentially identical to the corresponding sub-routine under dynamic SWiSSSE.**Srch** except that the client has to overwrite the target document to a fake document in the stash.

Supporting uncapped number of insertions. As one can clearly see from the bucketization strategy and the fake document generation procedure in our construction, there is a limit on how many documents the client can insert into the database. One possible work-around is to instantiate a new encrypted database every time the maximum quota is hit. This may not be practical for some systems as the client storage grows linearly in the number of instances of encrypted databases.

As an alternative, we can extend our dynamic construction to support uncapped document insertions at the cost of additional leakages. Without loss of generality, suppose that the client wants to store a documents more. He can simply insert a fake documents in the stash and redirect some of the pointers of the fake keywords (which he can obtain from normal queries) to these new fake documents. These new fake documents can then be written back to the server just like the normal documents. If the client wants to store

Algorithm 7.11 Dynamic SWiSSSE.**Insert**: Write-Back Sub-Routine

procedure Clt.**Write_Back_Insertion**($M, \{\bar{kw}_j\}, \bar{doc}$)
  Replace the lookup addresses of the newly inserted documents which contain $\bar{kw}$ with the addresses used for the fake documents.
  $UA \leftarrow \{\}$
  / Get random documents from the stash */*
  $D \leftarrow$ Clt.A.**pop**(|Clt.A|)
  for $(\{(kw_i, j_i, b_i)\}, doc) \in UA$ **do**
    / Encrypt the new document addresses and documents */*
    $UA \leftarrow UA \cup \{(F(\mathsf{id}(doc)||\texttt{Clt.ArrCtr}[\mathsf{id}(doc)]), \textbf{Enc}(\{(kw_i, j_i)\}||doc))\}$
    / Update the stash for the lookup index */*
    for $(kw, j, b) \in \{(kw_i, j_i, b_i)\}$ **do**
      Clt.I.**put**$((F(kw||j||3 * \texttt{Clt.KWCtr}[kw] + b), \textbf{Enc}(\mathsf{id}(doc))))$
  / Decrypt the documents retrieved and insert them into the document array */*
  Clt.A.**Insert**(**Dec**(M))
  Insert document $\bar{doc}$ with keywords $\{\bar{kw}_j\}$ into Clt.A
  Send (Clt.I.**pop**($\lfloor$|Clt.I| /2$\rfloor$), UA)

procedure Svr.**Write_Back**((UI, UA))
  Svr.EI.**put**(UI)
  Svr.EA.**put**(UA)

Algorithm 7.12 Dynamic SWiSSSE.**Delete**: Write-Back Sub-Routine

procedure Clt.**Write_Back_Deletion**($M, \bar{doc}$)
  Replace the lookup addresses of the newly inserted documents which contain $\bar{kw}$ with the addresses used for the fake documents.
  $UA \leftarrow \{\}$
  / Get random documents from the stash */*
  $D \leftarrow$ Clt.A.**pop**(|Clt.A|)
  for $(\{(kw_i, j_i, b_i)\}, doc) \in UA$ **do**
    / Encrypt the new document addresses and documents */*
    $UA \leftarrow UA \cup \{(F(\mathsf{id}(doc)||\texttt{Clt.ArrCtr}[\mathsf{id}(doc)]), \textbf{Enc}(\{(kw_i, j_i)\}||doc))\}$
    / Update the stash for the lookup index */*
    for $(kw, j, b) \in \{(kw_i, j_i, b_i)\}$ **do**
      Clt.I.**put**$((F(kw||j||3 * \texttt{Clt.KWCtr}[kw] + b), \textbf{Enc}(\mathsf{id}(doc))))$
  / Decrypt the documents retrieved and insert them into the document array */*
  Clt.A.**put**(**Dec**(M))
  Turn $\bar{doc}$ into a fake document in Clt.A
  Send (Clt.I.**pop**($\lfloor$|Clt.I| /2$\rfloor$), UA)

procedure Svr.**Write_Back**((UI, UA))
  Svr.EI.**put**(UI)
  Svr.EA.**put**(UA)

a additional documents for a particular keyword kw, he can make a search query on kw to retrieve the documents associated to kw, increase the address space of kw by a keywords, and generate a fake documents and point the newly generated keyword pointers to the new fake documents. These pointers and documents can then be written-back to the server with normal write-back operations. On a side note, the client should choose

a such that the new bucket size of kw corresponds to the bucket size of some other keyword, so that the volume leakage does not trivially leak the identity of kw in the future queries.

We leave it as an interesting future work to formalize the storage expansion process, and to analyse the additional leakages thereof.

7.6.3 Correctness

Similar to the static case, there is a possibility for our dynamic construction to fail if the client generates repeated addresses. We provide an upper bound of the failure probability of our dynamic construction with adversarially chosen queries below. As the proof is almost identical to the static case, we omit the proof from the paper.

Theorem 7.3. [Correctness of Dynamic SWiSSSE]

Let $|\mathbf{DB}|$ and $|KW\{\mathbf{DB}\}|$ denote the total number of documents and document-keyword pairs, respectively, in the database $\mathbf{DB}$ at any given point of time, and let l denote the output length of the PRF F used in static SWiSSSE. Then the advantage of any adversary $\mathcal{A}$, which issues at most k queries, in breaking the correctness of static SWiSSSE over the database $\mathbf{DB}$ is at most:

$$\frac{\left(|\mathbf{DB}|^2 + 4t_0\,|\mathbf{DB}| + 9\,|KW\{\mathbf{DB}\}|^2 + 18t_1\,|KW\{\mathbf{DB}\}|\right)}{2^{l+1}} + \mathsf{Adv}_{F,\mathcal{B}}^{PRF,|\mathbf{DB}|+2t_0} + \mathsf{Adv}_{F,\mathcal{C}}^{PRF,2|KW\{\mathbf{DB}\}|+2t_1},$$

where $t_0 = k \cdot \max_k w\,|\mathbf{DB}(kw)|$, $t_1 = k \cdot \max_k w\,|kw\{\mathbf{DB}(kw)\}|$ and $\mathcal{B}$ and $\mathcal{C}$ denote probabilistic polynomial-time adversaries in independent security experiments against the PRF F.

7.6.4 Security

Setup. At setup, the client offloads the encrypted lookup index and the encrypted document array to the server. These data structures are essentially key-value stores with pseudorandomly generated keys/addresses and values/entries that are encrypted under an IND-CPA secure encryption scheme. Hence, at setup, the server learns no information about the original database $\mathbf{DB}$ other than the number of documents in the padded database $\mathbf{DB}'$ (including both real and fake documents), and the total number of keyword-document pairs post-bucketization. Formally, we have:

$$\mathcal{L}_{\Sigma}^{\mathbf{Setup}}(\mathbf{DB}, \mathtt{G}) = (\left|\mathbf{DB}'\right|, \left|KW\left\{\mathbf{DB}'\right\}\right|, \mathbf{St}_{\mathcal{L}}).$$

Keyword Queries. As we have introduced virtual addresses for the inserted documents, the insertion history will be revealed by the keyword queries. As in the static case, we capture these leakages using a probabilistic and stateful leakage function, described formally in Algorithm 7.13.

Document Insertion. The leakage of a document insertion is identical to a single-keyword query except that the inserted document is processed in the state of the leakage.

Algorithm 7.13 Dynamic SWiSSSE: Leakage Function for Keyword Queries

procedure $\mathcal{L}^{\mathbf{Srch}}(\mathbf{q}, \mathbf{St}_{\mathcal{L}})$
  $(\mathbf{Srch}, \bar{kw}) \leftarrow \mathbf{q}$
  $\mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr} \leftarrow \mathbf{St}_{\mathcal{L}}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1]), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1] + 1), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1] + 2), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathtt{KWCtr}[\bar{kw}_1] \leftarrow \mathtt{KWCtr}[\bar{kw}] + 1$
  $L \leftarrow \mathtt{I}[\bar{kw}]$
  while $|L| < 2 \cdot \mathtt{Clt}.\mathtt{G}(kw)$ **do**
    $id \leftarrow \mathbf{Rand}(|\mathtt{A}|)$
    if $id \notin \{\mathsf{id}(doc) \mid doc \in \mathtt{A}'\}$ **then**
      $L \leftarrow L \cup id$
  $\mathtt{ArrCtr}[L] \leftarrow \mathtt{ArrCtr}[L] + 1$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(l, \mathtt{ArrCtr}[l]), 0, k) \mid l \in L\}$

  $UI \leftarrow \mathtt{I}'.\mathbf{pop}(|\mathtt{I}'|/2)$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(i, 1, k) \mid i \in UI\}$
State $UA \leftarrow \mathtt{A}'.\mathbf{Pop}(\lfloor |UA|/2 \rfloor)$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(\mathsf{id}(doc), \mathtt{ArrCtr}[\mathsf{id}(doc)]), 1, k) \mid (\{w_i, j_i, b_i\}, doc) \in UA\}$
  $\mathtt{A}' \leftarrow \mathtt{A}' \cup \mathbf{Merge\_Index}(\mathtt{A}[L], \bar{kw})$
  $\mathbf{St}_{\mathcal{L}} \leftarrow (\mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr})$
  Return $(\mathbf{IndHist}, \mathbf{ArrHist}), \mathbf{St}_{\mathcal{L}}$

We capture these leakages using a probabilistic and stateful leakage function, described formally in Algorithm 7.14.

Document Deletion. The leakage of a document deletion is identical to a single-keyword query except that the target document to be deleted is marked as fake in the state of the leakage.We capture these leakages using a probabilistic and stateful leakage function, described formally in Algorithm 7.15.

Finally, we are ready to state the security of our dynamic construction and prove it.

Theorem 7.4 (Security of Dynamic SWiSSSE)**.** *Let Σ be our proposed dynamic SSE scheme. Let $\mathcal{L}_{\Sigma}^{\boldsymbol{Setup}}$ and $\mathcal{L}_{\Sigma}^{\boldsymbol{Srch}}$, $\mathcal{L}^{\boldsymbol{Insert}}$, and $\mathcal{L}^{\boldsymbol{Delete}}$ be the leakage functions defined above, then Σ is $(\mathcal{L}_{\Sigma}^{\boldsymbol{Setup}}, \mathcal{L}_{\Sigma}^{\boldsymbol{Srch}}, \mathcal{L}^{\boldsymbol{Insert}}, \mathcal{L}^{\boldsymbol{Delete}})$-secure.*

Proof. We use a game-based argument to prove the security of the dynamic construction.

(**Game 0**) Let the real execution of the scheme on the database **DB** with queries $\mathbf{q}_1, \ldots, \mathbf{q}_k$ be game G_0. Then we have that for any adversary $\mathcal{A}$, $\Pr[\mathbf{Real}_{\Sigma,\mathcal{A}}(1^{\lambda}) = 1] = \Pr[G_0 = 1]$.

(**Game 1**) Let game G_1 be the same game as G_0 except that the execution of the setup step is replaced by the simulator. Clearly the simulator works the same way as the static case, so the difference in advantages between G_0 and G_1 is upper-bounded by $\mathsf{Adv}_F^{PRF,t_0} + t_0 \cdot \mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$, where $t_0 = 2\sum_{kw} \mathtt{G}(kw) + |\mathbf{DB}|$.

Algorithm 7.14 Dynamic SWiSSSE: Leakage Function for Insertion Queries

procedure $\mathcal{L}^{\mathbf{Insert}}(\mathbf{q}, \mathbf{St}_{\mathcal{L}})$
  $(\mathbf{Insert}, \{\bar{kw}_i\}, \bar{doc}) \leftarrow \mathbf{q}$
  $\mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr} \leftarrow \mathbf{St}_{\mathcal{L}}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1]), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1] + 1), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1] + 2), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathtt{KWCtr}[\bar{kw}_1] \leftarrow \mathtt{KWCtr}[\bar{kw}_1] + 1$
  $L \leftarrow \mathtt{I}[kw]$
  while $|L| < 2 \cdot \mathtt{Clt}.\mathtt{G}(kw)$ **do**
    $id \leftarrow \mathbf{Rand}(|\mathtt{A}|)$
    if $id \notin \{\mathsf{id}(doc) \mid doc \in \mathtt{A}'\}$ **then**
      $L \leftarrow L \cup id$
  $\mathtt{ArrCtr}[L] \leftarrow \mathtt{ArrCtr}[L] + 1$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(l, \mathtt{ArrCtr}[l]), 0, k) \mid l \in L\}$

  $UI \leftarrow \mathtt{I}'.\mathbf{pop}(|\mathtt{I}'|/2)$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(i, 1, k) \mid i \in UI\}$
  $UA \leftarrow \mathtt{A}'.\mathbf{Pop}(\lfloor |UA|/2 \rfloor)$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(\mathsf{id}(doc), \mathtt{ArrCtr}[\mathsf{id}(doc)]), 1, k) \mid (\{w_i, j_i, b_i\}, doc) \in UA\}$
  $\mathtt{I}' \leftarrow \mathtt{I}' \cup \mathbf{Index}(UA, \mathtt{KWCtr})$
  $M \leftarrow \mathbf{Insert}(\mathtt{A}[L], \{\bar{kw}_i\}, doc))$
  $\mathtt{A}' \leftarrow \mathtt{A}' \cup \mathbf{Merge\_Index}(M, \bar{kw}_1)$
  $\mathbf{St}_{\mathcal{L}} \leftarrow (\mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr})$
  Return $(\mathbf{IndHist}, \mathbf{ArrHist}), \mathbf{St}_{\mathcal{L}}$

(**Game 2**) In game G_2, we replace the query algorithms with the simulator. The algorithms look the same for all query types so we only show the one for the single-keyword query. The simulator looks the same as game G_2 in the proof of security for the static case, but the lookup index tokens in the dynamic construction includes the addresses generated by the insertion queries too.

The number of addresses the algorithm has to generate is upper-bounded by $t_1 = 2\sum_i \mathtt{G}(KW(\mathbf{q}_i)) + 2\sum_i |\mathbf{DB}(KW(\mathbf{q}_i))|$, and the number of encryptions needs to be created is upper-bounded by the same t_1. This means the difference in advantages between G_1 and G_2 is upper-bounded by $\mathsf{Adv}_F^{PRF,t_1} + t_1 \cdot \mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$.

(**Conclusion**.) By combining the two games above, we see that the difference in advantages between G_0 and G_2 is at most $\mathsf{Adv}_F^{PRF,t_0+t_1} + (t_0 + t_1) \cdot \mathsf{Adv}_{\Sigma'}^{IND-CPA}(\lambda)$. □

7.6.5 Oblivious Operations

We introduce here a new notion of security for dynamic SSE schemes called "oblivious operations". Informally, a dynamic SSE scheme supports oblivious operations if document updates and keyword searches are computationally indistinguishable to an adversarial server. The formal definition is presented below.

Definition 7.2 (Oblivious Operations)**.** *Let Σ be a dynamic SSE scheme. Let $\boldsymbol{DB}$ be a database, $\mathtt{G}$ be the bucketization parameter, $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k-1}$ be a sequence of queries, and*

Algorithm 7.15 Dynamic SWiSSSE: Leakage Function for Deletion Queries

procedure $\mathcal{L}^{\mathbf{Delete}}(\mathbf{q}, \mathbf{St}_{\mathcal{L}})$
  $(\mathbf{Delete}, \{\bar{kw}_i\}, \bar{doc}) \leftarrow \mathbf{q}$
  $\mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr} \leftarrow \mathbf{St}_{\mathcal{L}}$

  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1]), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1] + 1), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(\mathbf{T}(\bar{kw}, i, 3 * \mathtt{KWCtr}[\bar{kw}_1] + 2), 0, k) \mid i \in 0, \ldots, \mathtt{G}(\bar{kw}_1) - 1\}$
  $\mathtt{KWCtr}[\bar{kw}_1] \leftarrow \mathtt{KWCtr}[\bar{kw}_1] + 1$
  $L \leftarrow \mathtt{I}[kw]$
  while $|L| < 2 \cdot \mathtt{Clt.G}(kw)$ **do**
    $id \leftarrow \mathbf{Rand}(|\mathtt{A}|)$
    if $id \notin \{\mathsf{id}(doc) \mid doc \in \mathtt{A}'\}$ **then**
      $L \leftarrow L \cup id$
  $\mathtt{ArrCtr}[L] \leftarrow \mathtt{ArrCtr}[L] + 1$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(l, \mathtt{ArrCtr}[l]), 0, k) \mid l \in L\}$

  $UI \leftarrow \mathtt{I}'.\mathbf{pop}(|\mathtt{I}'|/2)$
  $\mathbf{IndHist} \leftarrow \mathbf{IndHist} \cup \{(i, 1, k) \mid i \in UI\}$
  $UA \leftarrow \mathtt{A}'.\mathbf{Pop}(\lfloor |UA|/2 \rfloor)$
  $\mathbf{ArrHist} \leftarrow \mathbf{ArrHist} \cup \{(\mathbf{T}(\mathsf{id}(doc), \mathtt{ArrCtr}[\mathsf{id}(doc)]), 1, k) \mid (\{w_i, j_i, b_i\}, doc) \in UA\}$
  $\mathtt{I}' \leftarrow \mathtt{I}' \cup \mathbf{Index}(UA, \mathtt{KWCtr})$
  $M \leftarrow \mathbf{Delete}(\mathtt{A}[L], \bar{doc}))$
  $\mathtt{A}' \leftarrow \mathtt{A}' \cup \mathbf{Merge\_Index}(M, \bar{kw}_1)$
  $\mathbf{St}_{\mathcal{L}} \leftarrow (\mathtt{I}', \mathtt{A}', \mathtt{KWCtr}, \mathtt{ArrCtr})$
  Return $(\mathbf{IndHist}, \mathbf{ArrHist}), \mathbf{St}_{\mathcal{L}}$

Algorithm 7.16 Game G_1 (dynamic construction). Only the setup step is changed.

procedure $\textsc{Clt}.\mathbf{Setup}(\mathbf{DB})$
  $(N, p, \mathbf{St}_{\mathcal{L}}) \leftarrow \mathcal{L}_{\Sigma}^{\mathbf{Setup}}(\mathbf{DB}, \mathtt{G})$
  $\mathtt{EI}, \mathtt{EA} \leftarrow [\,]$
  /* *Generate the encrypted documents* */
  for $i = 0, \ldots, N - 1$ **do**
    $\mathtt{EA}.\mathbf{Insert}(\mathbf{RF}(2i), \mathbf{Enc}(0^{l_0}))$
  /* *Generate the encrypted document addresses* */
  for $i = 0, \ldots, 2p$ **do**
    $\mathtt{EI}.\mathbf{Insert}(\mathbf{RF}(2i + 1), \mathbf{Enc}(0^{l_1}))$
  Send $(\mathtt{EI}, \mathtt{EA})$ to the server

$\boldsymbol{q}_k$ and $\boldsymbol{q}'_k$ be two queries such that $KW(\boldsymbol{q}_k) = KW(\boldsymbol{q}'_k)$. Let $\ell_0, \boldsymbol{St}^0_{\mathcal{L}} \leftarrow \mathcal{L}_{\Sigma}^{\boldsymbol{Setup}}(\boldsymbol{DB})$ and $\ell_i, \boldsymbol{St}^i_{\mathcal{L}} \leftarrow \mathcal{L}^*_{\Sigma}(\boldsymbol{q}_i, \boldsymbol{St}^{i-1}_{\mathcal{L}})$ for $0 < i \leq k$ where $\mathcal{L}^*_{\Sigma}$ is the appropriate leakage function for the query $\boldsymbol{q}_i$, and $\ell'_k, \boldsymbol{St}'^k_{\mathcal{L}} \leftarrow \mathcal{L}^*_{\Sigma}(\boldsymbol{q}'_k, \boldsymbol{St}^{k-1}_{\mathcal{L}})$.

We say that Σ supports oblivious operations if ℓ_k is computationally indistinguishable from ℓ'_k for any choice of $\boldsymbol{DB}$, $\mathtt{G}$, $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_k$ and $\boldsymbol{q}'_k$.

Note that the definition of oblivious operations only requires the outputs of the leakage functions (at the point where the query is executed) to be indistinguishable. It does not,

Algorithm 7.17 Game G_2 (dynamic construction).

procedure CLT.**Srch**(**q**)
  $(\mathbf{IndHist}, \mathbf{ArrHist}), \mathbf{St}_{\mathcal{L}} \leftarrow \mathcal{L}_{\Sigma}^{\mathbf{Srch}}(\mathbf{DB}, \mathbf{q}, \mathbf{St}_{\mathcal{L}})$

  /* *Encrypted document array address retrieval* */
  $L \leftarrow \{\}$
  $t' \leftarrow$ the number of single-keyword queries executed
  for $i \in \{i \mid (i, b, t) \in \mathbf{IndHist}, b = 0, t = t'\}$ **do**
    $L \leftarrow L \cup \mathbf{RF}(2i + 1)$
  Send L to the server

  /* *Encrypted document retrieval* */
  $L \leftarrow \{\}$
  for $i \in \{i \mid (i, b, t) \in \mathbf{ArrHist}, b = 0, t = t'\}$ **do**
    $L \leftarrow L \cup \mathbf{RF}(2i)$
  Send L to the server

  /* *Write-back* */
  $UI, UA \leftarrow \{\}$
  for $i \in \{i \mid (i, b, t) \in \mathbf{IndHist}, b = 1, t = t'\}$ **do**
    $UI \leftarrow UI \cup (\mathbf{RF}(2i + 1), \mathbf{Enc}(0^{l_0}))$
  for $i \in \{i \mid (i, b, t) \in \mathbf{ArrHist}, b = 1, t = t'\}$ **do**
    $UA \leftarrow UA \cup (\mathbf{RF}(2i), \mathbf{Enc}(0^{l_1}))$
  Send (UI, UA) to the server

however, require the states of the leakage function to be indistinguishable. This makes sense because the leakage output is available to the adversary as soon as the corresponding operation is executed, which makes for an easy mapping task. On the other hand, the information contained in the state of the leakage function may be revealed to the adversary at a later point of time (for instance, via delayed pseudorandom write-backs in our scheme), and it is computationally hard for the adversary to map it back in time to the exact query it corresponds to.

Our dynamic SSE scheme naturally satisfies the aforementioned definition of oblivious operations. Both keyword searches and document updates involve reading a set of entries from the encrypted data structures, followed by delayed write-backs. The only functional differences between searches and updated are reflected in how the client locally manages/updates its stash. From the point of view of the server, the output of the leakage function at the point of query are simply the accesses made to the encrypted data structures, which is unconditionally indistinguishable for searches and updates. This allows us to state the following theorem.

Theorem 7.5 (Oblivious Operations)**.** *The dynamic variant* SWiSSSE *described above supports oblivious operations.*

7.6.6 Forward and Backward Privacy of Dynamic SWiSSSE

In this subsection, we describe the notions of forward and backward privacy achieved by dynamic SWiSSSE, and compare these with the notions of forward and backward privacy achieved by existing SSE constructions. In particular, we stress the fact that

SWiSSSE is the first dynamic SSE scheme in the literature to achieve strong backward privacy guarantees against system-level leakages.

Forward Privacy. Forward private SSE was introduced by Chang and Mitzenmacher in [39], and has been subsequently studied in [169, 28, 25, 74, 109, 29, 68, 167]. An SSE scheme is said to be forward private if insertion and deletion operations computationally hide the set of keywords in the underlying document. Forward privacy has received much attention in light of leakage-abuse and file injection attacks [33, 200], which are potentially devastating for SSE schemes that try to support updates without being forward private.

Observe that combining Theorems 7.4 and 7.5 allows us to claim that our dynamic SSE scheme achieves stronger forward privacy guarantees than existing constructions in the literature, *including* those based on computation/communication-intensive techniques such as ORAM [74, 25, 29, 37]. In particular, existing definitions of forward privacy do not hide the *number of keywords* an inserted/deleted document contains, which is potentially sensitive information. Our construction, on the other hand, achieves the stronger notion of forward privacy in which we also hide from the server the number of keywords in a document which is inserted/deleted.

We now present a more detailed argument. By Theorem 7.5, our dynamic SSE scheme satisfies indistinguishability of operations. Hence, the output of the leakage function for updates is computationally indistinguishable from the output of the leakage function for keyword searches. Next, by Theorem 7.4, the leakage function output for searches is the set of accesses made to the encrypted data structures at the server, which reveals no information to a computationally bounded adversary about the underlying keywords and documents Hence, at the point of an update operation, our dynamic scheme not only computationally hides the actual keywords in the target document, but also the number of keywords. As discussed later, this has important repercussions with respect to security against leakage-abuse and file-injection attacks.

Backward Privacy. The notion of backward privacy for dynamic SSE is comparatively more recent, and was first formalized by Bost *et al.* in [29]. Subsequently, Chamani *et al.* [37] and Sun *et al.* [174] proposed SSE schemes supporting single-keyword search that are backward private under various leakage profiles. The strongest notion of backward privacy formalized in [29] is called Type-1 backward privacy [29]. A dynamic SSE scheme is said to be Type-1 backward private if a search query on a keyword kw reveals no information to the adversary beyond result pattern for kw and the timestamps at which the documents containing kw were inserted into the database. The only constructions to achieve this strong notion of backward privacy adopt ORAM-style techniques and require polylogarithmically many communication rounds for searches [29, 37].

Once again, Theorem 7.4 allows us to claim that our dynamic SSE scheme achieves stronger than Type-1 backward privacy guarantees. This is particularly notable given that our construction only require two rounds of communication between the client and the server for searches.

To begin with, observe that as per Theorem 7.4 the leakage function output at the point of searches in our construction hides the result pattern and the update history for the underlying keyword from the server. If the adversary could monitor the state of the leakage function from the beginning of time up until the point of query, it could potentially learn the update history associated with a keyword. However, in the actual

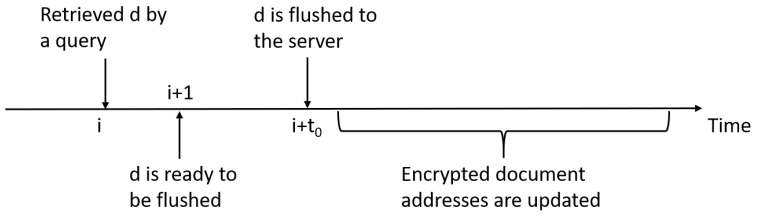


Figure 7.5: An illustration of the operations related to a document *doc* in our construction. At time i, the document *doc* is retrieved by a query. The document is in the stash and ready to be written back at time $i + 1$. The document itself is written back at time $i + t_0$, where t_0 is a random delay due to randomised write-backs. The encrypted document addresses associated to *doc* will be updated randomly in time later than $i + t_0$.

scheme, the adversary can only glean this through observing the delayed write-backs. However, given that the write-backs are mixed and matched and target pseudorandom locations, it is difficult for a computationally bounded adversary to trace each encrypted document it accesses during a search query at timestamp t back to the timestamp $t' < t$ when the document was originally inserted.

In the discussion below, we expand some more on delayed write-backs and their impact on the leakage of our dynamic scheme using an example. An illustration of the same is presented in Figure 7.5.

Encrypted document write-back. Without loss of generality, let **DB** be the database and $(\mathbf{q}_1, \ldots, \mathbf{q}_k)$ be the sequence of queries on the database. Let *doc* be one of the documents retrieved in query $\mathbf{q}_i$ where $1 \leq i < k$. We are interested in the distribution of t_0 for which query $\mathbf{q}_{i+t_0}$ triggers the write-back of document *doc*. As half of the documents are written back from the stash after each query, t_0 clearly follows a shifted geometric distribution with parameter $\frac{1}{2}$, unless that there is a query $\mathbf{q}_{i+j}$ that retrieves *doc*. In the latter case, the write-back of document *doc* will happen at query $\mathbf{q}_{i+j+t_0}$ with t_0 following a shifted geometric distribution with parameter $\frac{1}{2}$.

Encrypted document address write-back. Under the same setting as above, let *doc* be one of the documents retrieved in query $\mathbf{q}_i$ where $1 \leq i < k$ and kw be one of the keywords of *doc*. We are interested in the distribution of t_1 for which query $\mathbf{q}_{i+t_1}$ triggers the write-back of the encrypted document address associated to keyword kw. Recall that the write-backs for the encrypted document addresses are scheduled after the respective documents are written back to the server, and half of the encrypted document addresses are written back to the server in each query, this means that t_1 follows the sum of a shifted geometric distribution and a geometric distribution both parameterized by $\frac{1}{2}$, or equivalently, one plus a negative binomial distribution with parameter $(2, \frac{1}{2})$. As before, if the document *doc* is retrieved by another query $\mathbf{q}_{i+j}$ before it is written back, then we will write the encrypted document address in query $\mathbf{q}_{i+j+t_1}$.

Resistance to Cryptanalysis. Finally, for appropriate parameter choices (e.g., bucket sizes and bucketization strategies), dynamic SWiSSSE achieves strong enough backward privacy guarantees in practice to resist a wide range of cryptanalytic attacks based on system-wide leakages, such as access pattern and query equality pattern based attacks [94, 33], file injection attacks [200], and attacks based on highly refined leakages (such as the correlation-leakage based attack described in Section 7.5). Also noteworthy is the fact that SWiSSSE achieves such strong guarantees without compromising

Storage	Stash	$\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} \lvert KW\{\mathbf{DB}(kw)\}\rvert)$
	EDB/DB	$\mathcal{O}(\sum_k w\mathtt{G}(kw) + \lvert\mathbf{DB}\rvert)$
Time complexity	Document retrieval	$\mathcal{O}(\mathtt{G}(kw))$
	Write-back	$\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} \lvert KW\{\mathbf{DB}(kw)\}\rvert)$
Communication volume	Document retrieval	$\mathcal{O}(\mathtt{G}(kw))$
	Write-back	$\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} \lvert KW\{\mathbf{DB}(kw)\}\rvert)$

Figure 7.6: A summary of the performance parameters. Here, kw denotes the leading keyword of a query, $\mathtt{G}(kw)$ is the bucket size of keyword kw, $|KW\{\mathbf{DB}\}|$ is the total number of keyword-document pairs, and $|KW\{\mathbf{DB}(kw)\}|$ is the total number of keyword-document pairs for the documents that contain kw.

significantly on search/update performance and communication overheads. This makes it an attractive candidate for deployment in typical applications involving outsourced databases.

7.7 Performance Analysis

In this section, we provide an asymptotic performance analysis of SWiSSSE in both the static and dynamic cases. A summary of the key performance characteristics can be found in Figure 7.6 and some concrete numbers are presented subsequently.

Size of the stash. For search queries, recall that the half of the stash is flushed every iteration and filled with the response from the latest query. Since the number of documents retrieved by any query is less than $2 \cdot \max_{kw} \mathtt{G}(kw)$ (half of that comes from randomly generated document addresses), there are at most $4 \cdot \max_{kw} \mathtt{G}(kw)$ documents in the stash. The documents are padded to a constant size, which means the storage of the documents in the stash requires $\mathcal{O}(\max_{kw} \mathtt{G}(kw))$ space. For document insertion queries, the documents to be inserted are processed with the responses, so the same analysis on the space complexity applies.

The stash also stores a local lookup index. For a search query on keyword kw, the number of lookup index locations that need to be updated is equal to the number of keyword-document pairs in the query response, or $|KW\{\mathbf{DB}(kw)\}|$. Consider a document insertion query with document doc and keywords $\{kw_1, \ldots, kw_k\}$. Without loss of generality, assume that kw_1 is the leading keyword. Then the number of lookup index locations that need to be updated is at most $|KW\{\mathbf{DB}(kw_1)\}| + k - 1$. Since the number of keywords in the document is much smaller than $|KW\{\mathbf{DB}(kw_1)\}|$, it is reasonable to treat k as a constant in the asymptotic analysis. Recalling that half of the lookup index stored in the stash is flushed to the server after each query, it is not hard to see that the maximum number of lookup index locations stored by the client is $\mathcal{O}(\max_{kw} |KW\{\mathbf{DB}(kw)\}|)$.

In addition, the client needs to store three arrays of integers, namely an array for the groupings of the keywords, an array for the number of insertions of the keywords, and an array for the counters used to generate the document array addresses. These arrays are all small and of constant size, so they do not contribute to the asymptotic size of the stash. Combining everything together, we get that the size of the stash is $\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} |KW\{\mathbf{DB}(kw)\}|)$.

Size of the encrypted database. The server stores an encrypted lookup index and

an encrypted document array. The size of the encrypted lookup index is proportional to the total number of keyword-document pairs whereas the size of the encrypted document array is proportional to the number of documents. Hence, the size of the encrypted database is $\mathcal{O}(\sum_k w\mathtt{G}(kw) + |\mathbf{DB}|)$. Note that this order-of-magnitude calculation ignores the overhead from padding all documents to a constant size,

Time complexity and communication volume of a query. Suppose that the leading keyword for the query is kw. In our construction, a query consists of three rounds of interaction. In the first round, the client computes the encrypted lookup index addresses for the query. This involves $\mathcal{O}(\mathtt{G}(kw))$ computation and communication, as there are at most $3 \cdot \mathtt{G}(kw)$ addresses involved. The server then takes $\mathcal{O}(\mathtt{G}(kw))$ time to retrieve the encrypted document array addresses and send them to the client. This means that the overall communication volume is $\mathcal{O}(\mathtt{G}(kw))$ for the first round.

Upon receiving the $\mathcal{O}(\mathtt{G}(kw))$ encrypted document array addresses, the client processes them and retrieves $2 \cdot \mathtt{G}(kw)$ encrypted documents from the server. The client decrypts the documents and filters the results locally to obtain the query response. The time complexity for the overall process is $\mathcal{O}(\mathtt{G}(kw))$. The communication volume and the time complexity for the server are straight-forwardly $\mathcal{O}(\mathtt{G}(kw))$.

Finally, after receiving the encrypted documents from the previous step, the client decrypts them in $\mathcal{O}(\mathtt{G}(kw))$ time. If the query is a document insertion query, the client has to do at most $\mathcal{O}(\mathtt{G}(kw))$ amount of work to turn one of the fake documents into the document intended for insertion. After that, the client randomly picks at most $2 \cdot \max_{kw} \mathtt{G}(kw)$ documents from the stash, encrypts them and uploads them to the server. He also randomly picks half of the lookup indices stored in the stash, encrypts them, and uploads them to the server. Using the analysis of the stash size above, we conclude that the time complexity and communication volume for this step is $\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} |KW\{\mathbf{DB}(kw)\}|)$. This means the overall time complexity of this step for the client and the communication volume is $\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} |KW\{\mathbf{DB}(kw)\}|)$. Similarly, we conclude that the time complexity of this step for the server is $\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} |KW\{\mathbf{DB}(kw)\}|)$.

Combining the analyses above together, we conclude that the time complexity of a query for both the client and the server is $\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} |KW\{\mathbf{DB}(kw)\}|)$, while the communication volume of a query is $\mathcal{O}(\max_{kw} \mathtt{G}(kw) + \max_{kw} |KW\{\mathbf{DB}(kw)\}|)$. We note that stash handling is not relevant to the retrieval of documents and it can be performed whenever the client is free. With regards to document retrieval only, the time complexity for the client and the server is $\mathcal{O}(\mathtt{G}(kw))$ and the communication volume is $\mathcal{O}(\mathtt{G}(kw))$.

Alternative Parameters. There are three parameters we have fixed when we first introduced the construction, namely the number of fake documents, the number of dummy documents retrieved per query and the write back rate. These parameters can be changed to achieve different trade-off between efficiency and security.

The number of fake documents we initialise the database with affects the number of insertion queries we can make on the database. By using a larger number, the database can support more insertion queries, but as the fake documents have to be initialised with dummy documents during the setup, the time required to initialise the database is longer and the storage on the server takes more space. Maybe unexpectedly, more fake documents implies less noise in the co-occurrence leakage, as each document contains

fewer keywords. This may mean that a more aggressive padding parameter or padding strategy has to be used to prevent attacks based on co-occurrence leakage.

Recall that in our construction, every time when the client wants to retrieve some documents, he retrieves an equal number of dummy documents by generating addresses at random. This helps to introduce noise in the co-occurrence leakage and prevents an attacker from identifying the set of retrieved documents exactly. However, if the user is willing to leak the document access pattern then he can set the number of dummy document retrievals to zero, as long as the resultant keyword co-occurrence leakage does not lead to a query recovery attack.

In the write-back phase, we set the number of keyword-document pairs and documents to be written back to half of the size of the stash. The client can opt to write back a different fraction of the stash, whenever the stash is full, or after a random number of queries each time. The choice affects the client storage, the probability of leaking the search pattern and the level of noise in the keyword co-occurrence leakage. For example, if the client decides to flush everything in the stash every 4 iterations, the worst-case client storage doubles, the probability of leaking the search pattern is upper-bounded by 25%, and the signal-to-noise ratio in the keyword co-occurrence leakage quarters.

7.8 Experimental Results

Overview. In this section, we benchmark a prototype implementation of dynamic SWiSSSE and compare it to a plaintext database and other state-of-the-art SSE schemes.

As target database we choose the Enron email corpus [194] (see Section 6.1.1.2). It contains over 500K emails and over 30M keyword-document pairs which makes it a perfect database to experiment with the scalability of SWiSSSE. We run experiment on 1K to 400K emails and report the performance numbers.

7.8.1 Experimental Setup

Choice of Primitives. We instantiate the PRF in our construction with HMAC-SHA-256 [112]. Only the first 16 bytes of the output are used as keys to reduce storage. We use AES-GCM [138, 127] as the encryption function.

Implementation. We implement the client in Java [175], using the Java Cryptography Extension [141] as the underlying cryptographic library; AES-NI was enabled in our implementation. We choose to use a single-thread implementation as it provides the most accurate measurements of performance. We use Redis [1] as the underlying database system on the server. To reduce memory consumption of Redis, the inverted indices are stored in hashes [1] as opposed to a direct key-value store. This choice leads to over 5 times of reduction on memory consumption but has a penalty on the time complexity of insertion and deletion operations. On the other hand, the documents are stored in a key-value store directly.

[1] A hash in Redis is a hash table where each entry in the table is a list of key-value pairs. An access with a "master key" is very efficient but the retrieval of an actual key-value pair requires a linear scan on the list for which the key-value pair is in.

We also implement a plaintext database in Java. The database uses an inverted index for fast lookup as well, except that the keys of the inverted index are simply the keywords, and the values are lists of document identifiers associated to the keywords.

Bucketization Strategy. We use a bucket size of 400 for the most frequent 2400 keywords and a bucket size of 200 for the remaining keywords. This offers reasonable practical security and efficiency as shown in Section 7.5.2.

Padding of Documents. The real documents of SWiSSSE are padded to the smallest multiple of 2 thousand bytes before encryption. The fake documents are generated as strings of 2 thousand bytes.

Experimental Environment. We run our experiments on an Intel i7-7700K CPU clocked at 4.7 GHz and 32 GB DDR4 memory clocked at 2400 MHz. The server and client are run on the same machine.

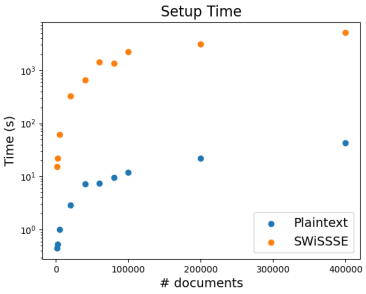
7.8.1.1 Benchmarks

Setup Time. Figure 7.7a shows the setup time of the plaintext database and SWiSSSE. SWiSSSE is two orders of magnitudes slower than a plaintext implementation which is expected due to encryption.

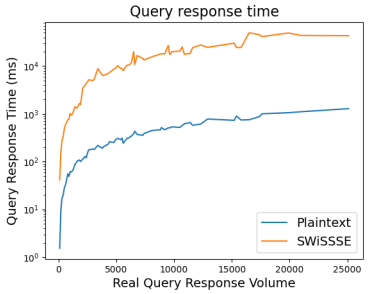
Query Response Time. Figure 7.7b shows the query response time of the plaintext database and SWiSSSE for the experiment with 400K documents. One thousand keywords are queried in each experiment. Here, real query response volume refers to the actual number of documents containing the keywords and query response time is defined to be the time from the start of a query to the point of time for which the client obtains the plaintext documents. SWiSSSE is about an order of magnitude slower than a plaintext database. This can be attributed to several reasons. Firstly, SWiSSSE deploys a bucketization strategy which leads to more documents retrieved than the true query response volume. Secondly, SWiSSSE uses the duplication technique on the inverted index, which means the amount of time required for index retrieval become linear in the bucket size of the queried keyword (as opposed to a single query for the plaintext database). Thirdly, SWiSSSE has to engage the stash to retrieve locally stored documents. Finally, SWiSSSE needs to perform cryptographic operations.

We note that the query response time of SWiSSSE can be improved significantly with parallelisation and multi-threading. For example, the computation of search keys and decryption of documents can be parallelised. Computation of search tokens and decryption of documents can be separated from interactions with the server into different threads to reduce unnecessary blocking time between different commands.

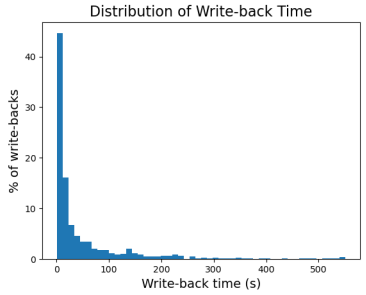
As insertion and deletion operations of SWiSSSE essentially calls the search operation and performs insertion/deletion locally, the query response time in those cases are similar to the search operation. Therefore, we omit the experimental results on these operations.



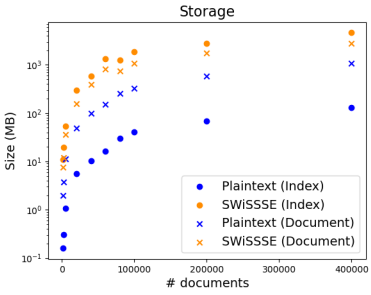
(a) Setup time of the plaintext database and SWiSSSE.



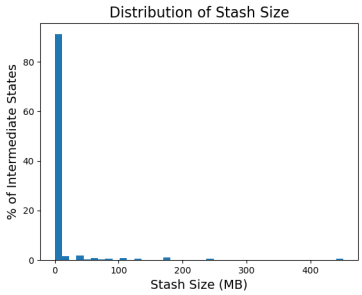
(b) Query response time of the plaintext database and SWiSSSE on 400K documents.



(c) Distribution of write-back time of SWiSSSE on 400K documents.



(d) Storage of the plaintext database and SWiSSSE .



(e) Stash size of SWiSSSE .

Figure 7.7: Performance comparison between the plaintext database and SWiSSSE.

Write-back Efficiency. We report write-back efficiency for the experiment with 400K documents in Figure 7.7c. As write-back time depends on the number of documents retrieved in previous queries, reporting an average value is not very informative. Here, we choose to show the distribution of write-back time with the queries we have made in the experiment. We observe that over 70% of the write-backs are completed under 30 seconds but there exists very long write-back times occasionally.

The major bottle-neck of the write-back operation (over 80% of the execution time) comes from inserting the key-value pairs into the Redis database – as we are using hashes as opposed to a direct key-value store, this is expected. In practice, the client can simply upload all the key-value pairs it wants to update and go offline; the server can then insert these key-value pairs on its own. That will reduce the write-back time for the client by a factor of 5.

Storage. Storage of the plaintext database and SWiSSSE is reported in Figure 7.7d. The main source of overhead for SWiSSSE comes from the inverted index as can be seen clearly from the graph. This is because the duplication technique is used in SWiSSSE and a lot more keys needs to be created for the inverted index. On the other hand, the overhead on document storage is only about a factor of two as expected.

Stash. The distribution of the stash size is shown in Figure 7.7e. The stash size was kept under 10 MB for over 80% of the queries. There were several times for which the stash size grows to over 100 MB. However, that is due to queries on keywords with high frequencies, which are rare in practice. Furthermore, as half of the documents are written back to the server after each query, the stash size will only be high for a few queries. Furthermore, in practice, the client can issue dummy queries to help speeding the process up.

7.8.2 Comparison to Other Searchable Encryption Schemes

In this section, we give a comparison between the state-of-the-art SSE schemes and SWiSSSE. As concrete performance numbers depend on the optimality of the implementations, our comparison is based on numbers that are independent of the implementation.

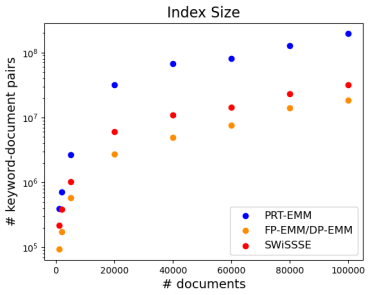
State-of-the-art SSE schemes. We pick **PRT-RMM** [98], **FP-EMM** and **DP-EMM** [149] as the schemes to compare to SWiSSSE. See Section 6.4 for a high-level description of these schemes. These schemes have only specified how document indices should be retrieved but not how documents should be retrieved. So for a fair comparison, we constrain ourselves to document indices only.

Setup Time and Storage. A comparison of the size of the search index is shown in Figure 7.8a. The index size is shown in terms of the number of keyword-document pairs.

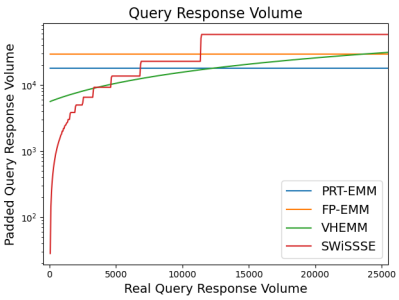
The index size of SWiSSSE shown here is the initial size of the index. It can grow to up to 2 times in size in the static setting and 3 times in the dynamic setting. Nonetheless, its size is on par with other schemes and reasonable in practice.

It is clear that the size of index of SWiSSSE is on par with the state-of-the-art SSE schemes.

Query Response Time. A comparison of query response volume is shown in Figure 7.8b. Here, we use real query response volume to refer to the actual number of documents containing the keywords and padded query response volume to refer to the number of documents retrieved by the schemes. As it can be seen, SWiSSSE has the smallest query response volume for all keywords with frequency less than 3300. This corresponds to over 95% of the keywords of the database (see Figure 6.2). SWiSSSE is outperformed by the other two schemes on more frequent keywords, but these keywords are less likely to be queried in practice, so we believe that it should not be seen as a limitation of SWiSSSE.



(a) Comparison of index size between the state-of-the-art SSE schemes and SWiSSSE.



(b) Comparison of query response volume between the state-of-the-art SSE schemes and SWiSSSE.

Figure 7.8: Performance comparison between the state-of-the-art SSE schemes and SWiSSSE.

7.9 Discussion

In this Chapter, we describe SWiSSSE, the first system-wide secure SSE scheme that can be instantiated in both the static and dynamic settings. Through experiments, we demonstrate that SWiSSSE is practically efficient not only as a search index, but as a full-fledged document retrieval system. An interesting feature of SWiSSSE is that it achieves oblivious queries, where an adversary cannot distinguish search, insertion and deletion queries. However, SWiSSSE has also raised many research questions.

Size of Inverted Index. SWiSSSE uses the duplication technique on the inverted index just like all other SSE schemes. This leads to three sources of overhead. Firstly, every keyword has to be masked by a PRF as many times as its number of occurrence. Given millions of occurrences of keywords, the keys of the inverted index takes up gigabytes of storage. Secondly, the document identifiers, which are just a few bytes in size, need to be encrypted by a standard encryption scheme. Just like the keys, encryption of the document identifiers leads to gigabytes of storage overhead. Finally, a key-value store requires additional overhead for its internal data structure to support efficient operations. For our experiment on 400K Enron emails for instance, a naive key-value store of the inverted index requires over 30 gigabytes of memory, which is far from scalable. Clearly, the inverted index is a bottleneck for scalability.

As a future research direction, we plan to investigate other techniques on the design of the inverted index. We also want to find more efficient representations of the encrypted database.

Size of Stash. As can be seen from our experiments in Section 7.8, the stash size can grow without control occasionally. This can be problematic for low-storage devices. In the future, we plan to investigate more efficient ways to use the stash and other write-back strategies.

Multi-client Setting. SWiSSSE is a single-client SSE scheme by design. It will be interesting to extend the ideas in SWiSSSE to the multi-client setting.

Richer Search Functionality. SWiSSSE currently only supports basic keyword search and update functionalities. We leave as future work richer search functionalities

such as conjunctive/disjunctive keyword search and keyword search with wildcards.

Security Proof. We prove the security of SWiSSSE in the standard security model and argue for its security with cryptanalysis on the leakage. In fact, cryptanalysis is the only tool available for arguing about security right now. This is certainly not enough as cryptanalysis is not always complete.

In the next chapter, we motivate a new security notion which captures security against classes of attacks by measuring the gain of the attacker with respect to particular attack goals.

Security vs Efficiency Trade-off. Unlike most of the works which focus on building efficient solutions with (close to) no leakage, SWiSSSE is an adventure in building an efficient solution with practically acceptable leakage. This allows SWiSSSE to bypass the theoretical lower bound on efficiency set for schemes with (close to) no leakage. We believe that it is an important step towards building practical cryptosystems as the leakage-free alternative is not efficient enough for practical deployment (yet) as demonstrated in Section 6.1.

Our work is very similar to other "leaky" schemes in the literature [197, 178, 185, 38] and we hope that SWiSSSE can inspire more works of this kind in the field of structured encryption and beyond.

Chapter 8

Foundations: Towards a Better Security Notion

The community takes an all-or-nothing approach to tackle leakage-abuse attacks. Instead of searching for acceptable leakage and design schemes with those leakage, many recent works [98, 149, 57] have focused on completely suppressing the information leakage that is known to be harmful. As a result, these schemes are forced to use expensive techniques such as full padding or ORAM, which defeats the purpose of encrypted databases.

We, on the other hand, believe that leakage is at the heart of structured encryption. It is unavoidable if we want to achieve efficiency. But at the same time, leakage can be problematic as demonstrated by leakage-abuse attacks (see Section 4.5, and Chapter 5 and 6). Hence, in order to design an efficient and secure scheme, we need to be able to answer the ultimate question: *how much leakage can we accept before it is too much*?

The standard security notion [53, 40] is certainly not the answer to the question as leakage-abuse attacks are still possible on a provably secure scheme. Leakage-abuse attacks themselves are not sufficient to answer the question either as they only show how to break schemes. What we need is a hybrid of the two: a security notion which features leakage-abuse attacks. If we can prove that a scheme is secure in this notion, we want to say that the scheme is secure with respect to the featured class of leakage-abuse attacks under certain assumptions of adversarial power.

In this chapter, we describe a novel security notion which achieves the goal above. Our notion is inspired by g-leakage [5] and it allows one to measure the security of a scheme with respect to security goals and adversarial powers on a relative scale. We show how previous constructions can be adapted to our security notion. We also propose three new searchable encryption constructions and demonstrate the usefulness of our new security notion. Finally, we show how our notion can be applied to other schemes in the literature.

Contents

8.1 Introduction

In this section, we revisit the standard security notion presented in [53, 40]. We discuss its drawbacks and argue that it is insufficient as a security notion. We also review other notions which attempted to improve on the standard notion. Finally, we motivate our own notion and highlight its advantages over the standard notion.

8.1.1 The Standard Security Notion

Recall from Section 4.3 that the standard security notion for structured encryption presented proposed in [53] and refined in [40] is a simulation-based game where security is parametrised by a leakage profile. The leakage profile is supposed to quantify what an adversary is allowed to learn. So if a scheme has a leakage profile that is deemed to be *harmless* by the community, we say that the scheme is *practically secure*.

This has changed since the discovery of leakage-abuse attacks [94, 122, 200, 102, 155, 115, 87, 126, 89, 17, 143]. In a leakage-abuse attack, the attacker is assumed to have some background information on the database, and the leakage from the schemes are used with the background information to recover private information about the database, such as the keywords of the queries and the keywords in the documents.

The key reason why these leakage-abuse attacks exist, even though there is a security proof, is that the security proof only shows an upper bound of information leakage, and it does not tell the security impacts of the leakage. For example, $DP - EMM$ [149] pads query response volumes by a small Laplace random variable (see Section 6.4) so the resultant leakage is noisy. Of course, one can say that the leakage is differential-private volume-hiding because the true query response volume is not directly leaked, but a simple frequency analysis [143] is able to break query privacy of the scheme.

There is also a problem with the quantification of leakage profiles as pointed out by Kerschbaum and Tueno [104]. Consider an encrypted database for range queries where the encrypted documents are ordered by their labels. Since the ordering mechanism is public information, it is not a part of the leakage profile. However, this ordering information can be used to locate the documents and launch attacks on them. So in some sense, the leakage profile based notion does not even capture the information leakage.

Some may argue that a security proof is still useful as it allows for cryptanalysis. This is certainly true, but it is problematic in several ways. Firstly, most of the leakage-abuse attacks in the literature make very strong assumptions on adversarial power and background information. For instance, the IKK attack [94] requires the entire database and a small portion of the queries to be known by the attacker before being able to recover a significant amount of queried keywords. This does not reflect a realistic scenario and it does not show quantitatively how much information is the attacker allowed to obtain before he breaches security. Secondly, the leakage-abuse attacks use specific techniques. For instance, our query reconstruction attacks in Chapter 6 relies on likelihood functions and simulated annealing. A failed attack does not necessarily mean that the underlying scheme is secure against query reconstruction attacks, but rather, it could be the case that the attack techniques are not suitable for the leakage profile. In that sense, leakage-abuse attacks offer very limited insights into practical security implications of leakages. Finally, many constructions in the literature [98, 149, 57] decide to suppress

a leakage completely if it is shown to be vulnerable to a leakage-abuse attack. This is of limited value for the development of encrypted databases, as many of these leakages, exemplified by access-pattern leakage, are expensive to suppress. A leakage-free scheme is inevitably going to be inefficient which defeats the purpose of an encrypted database.

8.1.2 Other Notions

We briefly mention some previous attempts at addressing the issues with the standard notion. In [104], Kerschbaum and Tueno pointed out that the standard notion does not capture the leakage entirely, as explained above. They proposed a new security notion for encrypted databases that support range queries to fix the problems. Their idea is to capture the (lack of) ordering information of the documents with an indistinguishability game, where the goal of the adversary is to guess the label of a document of his choice. A scheme is said to be secure with respect to the notion if the adversary cannot do better than random guessing.

Bost and Fouque [26] proposed to use constraints to fix the problems. Their idea is to use an indistinguishability game where in the two experiments, the adversary generates two databases and two sets of queries just like the indistinguishability game in [53]. However, the databases and the queries must satisfy certain constraints. For example, to capture the KKNO attack [102], the databases generated must have the same co-occurrence counts, as otherwise there is an adversary who can distinguish the two experiments.

These notions are still insufficient to capture leakage-abuse attacks. For [104], the notion is very specific to encrypted databases that support range queries, and it only captures storage privacy. We want a notion that is more general and captures all classes of attacks. With regards to [26], the security game is indistinguishability based, and it is not clear how to use it to measure the security impact. Furthermore, the constraints are rather arbitrary and it is hard to use them to model practical attacks.

In a related work, Boldyreva et al. [21] took a completely different approach to this problem. Their idea is to use a security notion that measures the success rate of an attacker with a specific goal directly. Informally, their security game works as follows. The adversary is given a set of ciphertexts of uniformly random messages and he has to output an interval of a given size so that at least one plaintext falls into it. The outcome of the game is 1 if the adversary has succeeded and 0 otherwise. The (windowed one-way) security of an order-preserving scheme with respect to an adversary is then defined to be the probability of the game returning 1.

This notion is very similar to the notion we are about to present except three key differences. Firstly, the notion in [21] is instantiated with uniformly distributed plaintexts. For encrypted databases, this is not enough, as databases follow a certain distribution, and we may design a scheme that is only secure for a certain distribution of databases. Secondly, the security of a scheme is measured by a probability in [21]. For encrypted databases, maybe only some of the ciphertexts/queries are interesting so looking at probabilities are not enough. Finally, the security proofs in [21] can use the idealised notion directly. We will see later in this chapter that it is not the case for our notion.

8.1.3 G-leakage and Our Notion

Our new security notion is inspired by g-leakage [5]. G-leakage is an information-theoretical model to measure the information gain of an adversary through an information channel. More formally, let X be a set of random events and $\mathcal{C}$ a channel which takes as input an event $x \in X$ and outputs another event $y \in Y$. The adversary observes y and outputs a guess $\omega \in \mathcal{W}$ with the goal to maximise the expectation of a public function (known as *gain function*) $g : \mathcal{W} \times X \to [0, 1]$. In this formulation, g can be thought of as an attack goal. For instance, g can be 1 if the guess is the same as the input, and 0 otherwise; or it can be 1 if the guess matches the first bit of the input, and 0 otherwise. The expectation of the gain function then allows one to measure how good the channel $\mathcal{C}$ is at defeating attack g.

We apply this idea to the context of encrypted databases. In particular, we model the database and the set of queries as X, the setup protocol and query protocols as the channel $\mathcal{C}$, the transcripts as Y, and the attack goal as g. To allow for auxiliary information as an input to the attacker, we modify X to output some auxiliary information which the adversary can use to produce his guess. By encoding an attack goal, such as query reconstruction, as a gain function g, one can compute the expectation over g for a specific adversary. For example, we can let g be 1 if the adversary is able to guess the keyword associated to a query, and 0 otherwise. Then, if the expectation over g is small for all adversaries, we know that the chance for the adversary to guess the keyword associated to a query is small, and the scheme is secure with respect to query reconstruction attacks.

Our notion is similar to the simulation-based security game in [53] except for a few key differences. Firstly, we let the environment generate the database and the queries and the adversary receives some auxiliary information, as opposed to letting the adversary picking the database and the queries himself. The auxiliary information is used to model background information in actual attacks. Secondly, upon execution of the setup protocol and the queries, the adversary outputs a *guess*. The guess, together with the database and the queries are used as the inputs to a gain function, and the output of the gain function is the output of the experiments. The goal of the adversary is to maximise the gain function given what he observes. Finally, as opposed to claiming that the scheme is secure if the output of the real game is indistinguishable from the ideal game, we use the expectation of the output from the real game as a measure of security impact, and the ideal game is only there to bound the expectation.

There are three main advantages of our security notion. Firstly, our notion is flexible in terms of auxiliary information. This allows one to quantify security impacts with respect to adversarial information. Secondly, our notion is relative in the sense that the output of the notion is no longer a single bit, but rather a number which measures how good a scheme is against a certain class of attacks. Finally, the attacker is not constrained to use a specific attack technique. He can use whichever technique that maximises his *gain* on the database. That way, we bypass the problem of needing to argue about security with respect to specific leakage-abuse attacks.

8.2 Preliminary Results

8.2.1 Notation

We use the shorthand $(x_i)_{i=0}^N$ to represent a list $(x_0, \ldots, x_N)$. If the indices are clear from the context, we may simply write (x_i) for the list.

We use the notation $(\text{output}_A, \text{output}_B) \leftarrow [f_A, f_B]$ to represent an execution of the interactive function f, where A and B are the two parties involved. In the notation, f_A is the part of the interactive function run by party A and f_B is the part of the interactive function run by party B. By the end of the execution, party A obtains output output_A and party B obtains output output_B.

8.2.2 Our Security Notion

Our Notion. Informally, our security notion on a scheme Σ can be described as follows. There is an environment $\mathcal{Z}$ that generates some data $\mathbf{D}$, polynomially many queries $(\mathbf{q}_i)$ and some auxiliary information **aux** on the data and the queries. The data is encrypted under some key key and is given to the adversary $\mathcal{A}$ with the auxiliary information **aux**. The environment then initialises the encrypted data **ED** and makes queries $(\mathbf{q}_i)$. The transcript of the setup phase and the queries are observed by the adversary. After polynomially many queries, the adversary outputs a guess ω which tries to maximise the expectation of the gain function $g(\omega, \mathbf{D}, (\mathbf{q}_i))$. We simplify the notation and write $g(\omega, \mathbf{D})$ to mean a gain function that only uses guess ω and data $\mathbf{D}$ as the input, or $g(\omega, (\mathbf{q}_i))$ if we are interested in query privacy. Security of a scheme with respect to a gain function g is defined to be supremum of expectation of the gain function over all adversaries.

Our security notion relies on a clever choice of the gain function g. The gain function g should reflect the goal of the adversary. For example, in a query reconstruction attack where the gain function takes shape $g(\omega, (\mathbf{q}_i))$, a guess ω consists of an index j and a keyword kw, we may set the gain function g to be:

$$g((j, kw), (\mathbf{q}_i)) = \mathbb{1}(W(\mathbf{q}_j) = kw).$$

The gain function is 1 if the adversary correctly guesses one of the queried keywords, and 0 otherwise. Then, the expectation over this gain function equals to the success rate of a specific query reconstruction attack $\mathcal{A}$. Finally, we take supremum of expectation over all adversaries as we care for security of scheme Σ with respect to all attacks that try to maximise the information gain represented by g, regardless of the technique used.

Our security game is shown formally in Definition 8.1.

Definition 8.1 (Our Security notion)**.** Let 1^λ be a security parameter. Consider the following probabilistic experiment for structured encryption scheme Σ where $\mathcal{A}$ is a stateful semi-honest adversary, $\mathcal{S}$ is a stateful simulator, `Clt` is the client played by the environment in the real game, and `Svr` is the server played by the adversary.

$\mathbf{Real}^g_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)$

$(\mathbf{D}, (\mathbf{q}_i), \mathbf{aux}) \leftarrow \mathcal{Z}(1^\lambda)$
$((\mathsf{key}, \mathbf{ED}), \mathbf{ED}, \mathcal{T}_0) \leftarrow \left[\mathbf{Setup}_{\mathtt{Clt}}(1^\lambda, \mathbf{D}), \mathbf{Setup}_{\mathtt{Svr}}(1^\lambda)\right]$
for $i \leftarrow 1, \ldots, |(\mathbf{q}_i)|$ **do**
    $(\mathbf{rsp}_i, \mathbf{ED}, \mathcal{T}_i) \leftarrow [\mathbf{Query}_{\mathtt{Clt}}(\mathsf{key}, \mathbf{q}_i), \mathbf{Query}_{\mathtt{Svr}}(\mathbf{ED})]$
$\omega \leftarrow \mathcal{A}((\mathcal{T}_i), \mathbf{aux})$
return $g(\omega, \mathbf{D}, (\mathbf{q}_i))$

The security of scheme Σ with respect to gain function g is defined to be the expectation of the experiment, i.e. $\mathbf{E}\left[\mathbf{Real}^{g}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right]$.

In addition to the use of a gain function in our notion as highlighted in the motivation, there are a few key differences between our notion and **SS-CQA-B**. First of all, the goal of the adversary in our notion is to learn something about the underlying data and/or queries, so it is unreasonable to let the adversary pick them. Instead, the environment $\mathcal{Z}$ does it. Secondly, we are able to control the auxiliary information **aux** given to the adversary with the environment $\mathcal{Z}$, hence, analysing the security of the underlying schemes under different prior knowledges. Finally, security of our notion is relative: a scheme is more secure if the supremum of the expectation of the gain function is smaller.

One can easily generalise our notion to other adversarial models. For example, the adversary may be allowed to pick the queries after he receives the encrypted data. This corresponds to an adversary who can execute non-adaptive queries on the server. A natural extension from there is to make the adversary adaptive, meaning that the adversary can choose the next query based on what he has seen so far. It is also possible to allow the adversary to corrupt some of the queries so that the decrypted responses are revealed to it.

It is, however, difficult to work with our security notion directly as the objects in the notion are real instances. Instead, we take a similar approach to the standard notion [98] by moving to an idealised notion. The idealised notion captures information leakage with leakage functions which allows us to argue about security using information-theoretic tools. We find that security in our idealised notion does not trivially translate to security in our notion, and we find conditions for which they do. We also show the relation between the standard notion and our notion.

8.2.3 Computational Indistinguishability of Expectations of Gain Functions

Motivation. In order to bound the expectation of our real game, we want to use the idealised game in Figure 8.1 with some leakage profile $\mathcal{L} = (\mathcal{L}_{\mathbf{Setup}}, \mathcal{L}_{\mathbf{Srch}})$, as it is easier to argue security with a leakage profile. Our idea is to show that the output by the adversary together with the data and queries in the real and ideal worlds are computationally indistinguishable, so that the expectation of the gain function in the real world is close to that in the ideal world. However, interestingly, this is not the case.

Indistinguishability does not Imply Equality in Expectation. For simplicity, we consider the following example where the environment generates a database **DB** with

$\mathbf{Ideal}^{g}_{\Sigma,\mathcal{A},\mathcal{Z},\mathcal{S}}(1^\lambda)$

1: $(\mathbf{D}, (\mathbf{q}_i), \mathbf{aux}) \leftarrow \mathcal{Z}(1^\lambda)$
2: $((\mathsf{key}, \mathbf{ED}), \mathbf{ED}, \mathcal{T}_0) \leftarrow \left[\mathcal{S}(1^\lambda, \mathcal{L}_{\mathbf{Setup}}(\mathbf{D})), \mathbf{Setup}_{\mathtt{Svr}}(1^\lambda)\right]$
3: **for** $i \leftarrow 1, \ldots, |(\mathbf{q}_i)|$ **do**
4: $\quad (\mathbf{rsp}_i, \mathbf{ED}, \mathcal{T}_i) \leftarrow [\mathcal{S}(\mathcal{L}_{\mathbf{Query}}(\mathbf{q}_i, \mathbf{D}), \mathbf{Query}_{\mathtt{Svr}}(\mathbf{ED})]$
5: $\omega \leftarrow \mathcal{A}((\mathcal{T}_i), \mathbf{aux})$
6: **return** $g(\omega, \mathbf{D}, (\mathbf{q}_i))$

Figure 8.1: Our idealised security game.

only one file and the file content is a binary string of length n, distributed uniformly in the domain. n is also the security parameter. The environment generates no query. The setup algorithm **Setup** from the client uses an IND-CPA encryption scheme to encrypt the file except when the content of the document in the file is the string 0^n. In the later case, the encryption is 0^n. It is straightforward to see that a binary leakage function which attains 1 if the document is 0^n and attains 0 otherwise can be used to quantify the setup leakage. For our example, we modify this leakage function slightly as follows:

- if the document is 0^n, return 1 with probability $1 - 2^{-n}$,
- return 0 otherwise.

This new leakage function still works for a **SS-CQA-B** proof as the chance of returning a wrong leakage is negligible.

Consider the following gain function:

$$g(\omega, \mathbf{DB}) = 2^{2n} \cdot \mathbb{1}\{\omega = f\} \text{ where } \mathbf{DB} = \{f\}.$$

In short, the gain function g attains 2^n when the adversary guesses correctly the content of the document in the database and is zero otherwise.

In the real world, when the document is 0^n, the adversary can guess the document content correctly all the times; when the document is anything else, he can guess the content correctly with probability $\frac{1}{2^n-1}$. This makes the expected gain $2^n + \frac{2^{2n}}{(2^n-1)^2}$. In the ideal world, when the document is 0^n, there is a 2^{-n} chance for the document to be encrypted as a random string that is different from 0^n, so the expected gain is reduced to $2^n - 1 + \frac{2^{2n}}{(2^n-1)^2}$. The difference between the expected gains is not negligible in n.

Computationally Indistinguishable Gain Functions. However, for reasonable gain functions, we do have that the expectation in the real world differs from the expectation in the ideal world by at most a negligible constant. Let $X = (\Omega, \mathcal{F}, P_X)$ be a probability space (See Section 2.1 for the definition of probability space). Let $|X|$ to be the number of outcomes of X, i.e. $|\Omega|$. For a function $g : \Omega \to \mathbb{R}$, we write $g(X)$ to mean a real-valued random variable which maps the outcomes of X to $g(\cdot)$ of it. We abuse the notation $z \in g(X)$ to mean elements in the image of $g(X)$. We formalise the criterion on the gain function in Theorem 8.1.

The reader may refer to Section 2.1 for the definition of probability space which is used below.

Theorem 8.1 (Bound on Expectations with respect to g)**.** Let $X = (\Omega, \mathcal{F}, P_X)$ and $Y = (\Omega, \mathcal{F}, P_Y)$ be probability spaces and $g : \Omega \to \mathbb{R}$ be a function which maps the outcomes of the events to real numbers. Then

$$\mathbf{E}[g(X)] \leq \mathbf{E}[g(Y)] + |g(X)| \cdot \sup_z |g(z)| \cdot \Delta(g(X), g(Y)),$$

where $\Delta(g(X), g(Y))$ denotes the statistical distance (or total variation distance) between $g(X)$ and $g(Y)$, i.e. $\sup_{z \in g(X)} |\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z]|$.

Proof of Theorem 8.1. We start by expressing the difference between the expectations.

$$\begin{aligned}
&|\mathbf{E}[g(X)] - \mathbf{E}[g(Y)]| \\
&= \sum_{z \in g(X)} z \cdot \mathsf{Pr}[g(X) = z] - \sum_{z \in g(Y)} z \cdot \mathsf{Pr}[g(Y) = z] \\
&= \sum_{z \in g(X \cup Y)} z \cdot (\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z]) \\
&= \sum_{z \in g(X)} z \cdot (\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z]) .
\end{aligned}$$

So in particular, there is a z such that

$$|z \cdot (\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z])| \geq \frac{|\mathbf{E}[g(X)] - \mathbf{E}[g(Y)]|}{|g(X)|}.$$

We can take a weaker bound on the right hand side by replacing z on the left by $\sup_z |g(z)|$, and we get

$$\begin{aligned}
\left| \sup_z |g(z)| \cdot (\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z]) \right| &\geq \frac{|\mathbf{E}[g(X)] - \mathbf{E}[g(Y)]|}{|g(X)|} \\
\sup_z |g(z)| \cdot |\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z]| &\geq \frac{|\mathbf{E}[g(X)] - \mathbf{E}[g(Y)]|}{|g(X)|} \\
|\mathsf{Pr}[g(X) = z] - \mathsf{Pr}[g(Y) = z]| &\geq \frac{|\mathbf{E}[g(X)] - \mathbf{E}[g(Y)]|}{|g(X)| \cdot \sup_z |g(z)|}.
\end{aligned}$$

The left hand side is bounded from above by statistical distance $\Delta(g(X), g(Y))$. Finally, by rearranging the terms, we have the desired result.

$$\begin{aligned}
\Delta(g(X), g(Y)) &\geq \frac{|\mathbf{E}[g(X)] - \mathbf{E}[g(Y)]|}{|g(X \cup Y)| \cdot \sup_z |g(z)|} \\
\mathbf{E}[g(X)] &\leq \mathbf{E}[g(Y)] + |g(X)| \cdot \sup_z |g(z)| \cdot \Delta(g(X), g(Y)).
\end{aligned}$$

□

It is straightforward to see that if two probability spaces are computationally indistinguishable, and a function g behaves nicely on the probability spaces, we get Corollary 8.1.

Corollary 8.1 (Bound on Computationally Indistinguishable Probability Spaces with respect to g)**.** Let 1^λ be a security parameter. Let $X = (\Omega, \mathcal{F}, P_X)$ and $Y = (\Omega, \mathcal{F}, P_Y)$ be probability spaces and $g : \Omega \to \mathbb{R}$ be a function which maps the outcomes of the events to real numbers. Assume that the following conditions hold:

1. X and Y are computationally indistinguishable,
2. $|\{\omega \mid g(\omega) \neq 0, \omega \in \Omega\}| \in \mathtt{poly}(\lambda)$,
3. $\sup_z |g(z)| \in \mathtt{poly}(\lambda)$.

Then

$$\mathbf{E}\left[g(X)\right] \leq \mathbf{E}\left[g(Y)\right] + \mathtt{negl}(\lambda).$$

8.2.4 SS-CQA-B Revisited

In this section, we bridge the gap between the standard notion and our security notion. In particular, we show that the black-box security of **SS-CQA-B** implies the outputs $(\omega, \mathbf{D}, (\mathbf{q}_i))$ in our real and ideal games are computationally indistinguishable. Therefore, the corresponding leakage profile and simulator can be used to bound the expected gain in our real experiment using reasonable gain functions. Our proof requires an intermediate security notion which is the same as **SS-CQA-B** except that the adversary outputs a string instead. We call this notion **SS-CQA-S** and its definition is given below.

Definition 8.2 (Modified Adaptive Security of Interactive STE (**SS-CQA-S**))**.** Let 1^λ be a security parameter. Consider the following probabilistic experiments for structured encryption scheme Σ where $\mathcal{A}$ is a stateful semi-honest adversary which outputs a string in the end, $\mathcal{S}$ is a stateful simulator, $\mathtt{Clt}$ is the client played by the environment in the real game, $\mathtt{Svr}$ is the server played by the adversary, $\mathcal{L} = (\mathcal{L}_{\mathbf{Setup}}, \mathcal{L}_{\mathbf{Query}})$ is the leakage profile:

$\mathbf{Real}^{\text{SS-CQA-S}}_{\Sigma,\mathcal{A}}(1^\lambda)$

$\mathbf{D} \leftarrow \mathcal{A}(1^\lambda)$
$((\mathsf{key}, \mathbf{ED}), \mathbf{ED}, \mathcal{T}_0) \leftarrow \left[\mathbf{Setup}_{\mathtt{Clt}}(1^\lambda, \mathbf{D}), \mathbf{Setup}_{\mathtt{Svr}}(1^\lambda)\right]$
for $i \leftarrow 1, \ldots, N$ **do**
  $\mathbf{q}_i \leftarrow \mathcal{A}(\mathbf{D}, (\mathbf{q}_j), (\mathcal{T}_j))$
  $(\mathbf{rsp}_i, \mathbf{ED}, \mathcal{T}_i) \leftarrow [\mathbf{Query}_{\mathtt{Clt}}(\mathsf{key}, \mathbf{q}_i), \mathbf{Query}_{\mathtt{Svr}}(\mathbf{ED})]$
$\omega \leftarrow \mathcal{A}(\mathbf{D}, (\mathbf{q}_i), (\mathcal{T}_i))$
return ω

$\mathbf{Ideal}^{SS-CQA-S}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda)$

$\mathbf{D} \leftarrow \mathcal{A}(1^\lambda)$
$((\mathsf{key}, \mathbf{ED}), \mathbf{ED}, \mathcal{T}_0) \leftarrow \left[\mathcal{S}(1^\lambda, \mathcal{L}_{\mathbf{Setup}}(\mathbf{D})), \mathbf{Setup}_{\mathtt{Svr}}(1^\lambda)\right]$
for $i \leftarrow 1, \ldots, N$ **do**

4: $\quad \mathbf{q}_i \leftarrow \mathcal{A}(\mathbf{D}, (\mathbf{q}_j), (\mathcal{T}_j))$
5: $\quad (\mathbf{rsp}_i, \mathbf{ED}, \mathcal{T}_i) \leftarrow [\mathcal{S}(\mathcal{L}_{\mathbf{Query}}(\mathbf{q}_i, \mathbf{D}), \mathbf{Query}_{\mathtt{Svr}}(\mathbf{ED})]$
6: $\omega \leftarrow \mathcal{A}(\mathbf{D}, (\mathbf{q}_i), (\mathcal{T}_i))$
7: **return** ω

We say that scheme Σ is string-wise adaptive $\mathcal{L}$-CQA secure in the black-box model if there is a simulator $\mathcal{S}$ such that for every PPT adversary $\mathcal{A}$ and distinguisher $\mathcal{D}$,

$$\mathsf{Pr}\left[\mathcal{D}(\mathbf{Real}^{\mathbf{SS\text{-}CQA\text{-}S}}_{\Sigma,\mathcal{A}}(1^\lambda)) = 1\right] - \mathsf{Pr}\left[\mathcal{D}(\mathbf{Ideal}^{\mathbf{SS\text{-}CQA\text{-}S}}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda)) = 1\right] \leq \mathtt{negl}(\lambda).$$

We say that scheme Σ is string-wise adaptive $\mathcal{L}$-CQA secure in the non-black-box model if for every PPT adversary $\mathcal{A}$ and distinguisher $\mathcal{D}$, there is a simulator $\mathcal{S}$ such that,

$$\mathsf{Pr}\left[\mathcal{D}(\mathbf{Real}^{\mathbf{SS\text{-}CQA\text{-}S}}_{\Sigma,\mathcal{A}}(1^\lambda)) = 1\right] - \mathsf{Pr}\left[\mathcal{D}(\mathbf{Ideal}^{\mathbf{SS\text{-}CQA\text{-}S}}_{\Sigma,\mathcal{A},\mathcal{S}}(1^\lambda)) = 1\right] \leq \mathtt{negl}(\lambda).$$

If the adversary $\mathcal{A}$ in the above experiments has to pick all the queries beforehand, we say that scheme Σ is string-wise non-adaptive $\mathcal{L}$-CQA secure.

For simplicity, we denote the black-box and non-black-box version of **SS-CAQ-B** as **B-BB** and **B-NBB** respectively; we denote the black-box and non-black-box version of **SS-CAQ-S** as **S-BB** and **S-NBB** respectively. The implications between the four notions are in Figure 8.2.

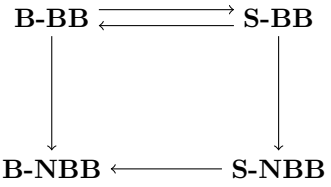


Figure 8.2: Implications between the notions

Proof of Implications in Figure 8.2. The implications **B-BB** $\Rightarrow$ **B-NBB** and **S-BB** $\Rightarrow$ **S-NBB** are trivial.

(**S-NBB** $\Rightarrow$ **B-NBB**) Suppose that the underlying scheme Σ is not secure in **B-NBB**, then there is an adversary $\mathcal{A}$ for all simulators $\mathcal{S}$, the output of the real and ideal experiments differs with non-negligible probability. We construct an adversary against **S-NBB** as follows. At the beginning of the experiment, the adversary outputs some data $\mathbf{D}$ and a sequence of queries $(\mathbf{q}_i)$. The queries are then executed to generate a sequence of transcripts $(\mathcal{T}_i)$. We run adversary $\mathcal{A}$ on input $(\mathbf{D}, (\mathbf{q}_i), (\mathcal{T}_i))$ to obtain a bit b and set the output of the **S-NBB** experiment to be $(b, \mathbf{D}, (\mathbf{q}_i))$. Since the probability of obtaining b in the real world and the ideal world is different in **B-NBB**, there must be a distinguisher who can distinguish $(b, \mathbf{D}, (\mathbf{q}_i))$ from the real world and the ideal world, hence, scheme Σ must be insecure in **S-NBB**.

(**S-BB** $\Rightarrow$ **B-BB**) The proof is similar to **S-NBB** $\Rightarrow$ **B-NBB** so we do not repeat it here.

(**B-BB** $\Rightarrow$ **S-BB**) For any simulator $\mathcal{S}$ in **B-BB**, we construct an adversary that generates a non-negligible difference in probability using adversaries for **S-BB** as follows. Let $\mathcal{A}$ and $\mathcal{D}$ be some adversary and distinguisher that breaks scheme Σ in **S-BB** for the simulator $\mathcal{S}$. The adversary against **B-BB** works as follows. It uses adversary $\mathcal{A}$ to generate some data $\mathbf{D}$ and a sequence of queries $(\mathbf{q}_i)$. The queries are executed to generate a sequence of transcripts $(\mathcal{T}_i)$. The adversary $\mathcal{A}$ is called again with input $(\mathbf{D}, (\mathbf{q}_i), (\mathcal{T}_i))$ to generate a string ω. This string is passed into distinguisher $\mathcal{D}$ and a bit b is produced. Since we assumed that **S-BB** is insecure, the probability of getting b in the real world and ideal world is different, and scheme Σ is insecure in **B-BB**. □

We are now ready to state Proposition 8.1 which connects **SS-CQA-B** to our notion. Informally, the proposition states that if a scheme is secure in the black-box model of **SS-CQA-B** (**B-BB**) with leakage $\mathcal{L}$, the same leakage can be used to compute the expected gain in our notion.

Proposition 8.1 (**B-BB** Security implies Security in our Notion)**.** Let $\mathcal{L}$ be a leakage profile and Σ be **B-BB** secure. Let g be a function which takes as input a string, some data and a list of queries, and outputs a positive real number such that:

1. $||g|| \in \texttt{poly}$,
2. $\sup_z |g(z)| \in \texttt{poly}$,

where $||g||$ denotes the size of the support of g where g is non-zero. Then for all environments $\mathcal{Z}$, there is a simulator $\mathcal{S}$, for every PPT adversary $\mathcal{A}$,

$$\mathbf{E}\left[\mathbf{Real}^{g}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right] \leq \mathbf{E}\left[\mathbf{Ideal}^{g}_{\Sigma,\mathcal{A},\mathcal{Z},\mathcal{S}}(1^\lambda)\right] + \texttt{negl}(\lambda). \tag{8.1}$$

Proof of Proposition 8.1. It suffices to show that the two experiments in our notion are computationally indistinguishable if the underlying scheme is secure in the **B-BB** model. Using Figure 8.2, we know that security in the **B-BB** model implies security in the **S-BB** model. Now, we modify the experiments in the **S-BB** model by replacing the first line of the real and ideal worlds with that of our notion, i.e. we let the environment sample the data, queries and auxiliary information:

$$(\mathbf{D}, (\mathbf{q}_1, \ldots, \mathbf{q}_N), \mathbf{aux}) \xleftarrow{R} \mathcal{Z}.$$

It is straightforward to see that security in the **S-BB** model implies security in this modified model. The only difference between this model and our notion is that the gain function has not been applied to the output of the experiment yet. It can be easily shown that the two experiments in our notion are indistinguishable if the modified model is secure with a simple reduction.

We can then use Proposition 8.1 to show the desired result. □

We note that all of the constructions in the literature use **B-NBB** as their security notion but the actual proofs are for **B-BB**. Hence, our security notion can be applied to those schemes.

8.3 New Constructions

In this section, we devise three searchable encryption schemes to show the usefulness of our notion. We note that these schemes are nowhere as efficient as SWiSSSE but their leakages are easier to analyse with our notion. We leave the analysis of the security of SWiSSSE and other schemes with complex leakage profiles as a future work.

8.3.1 Syntax of Searchable Encryption

The syntax of structured encryption we use in this chapter can be found in Section 4.1. We use searchable encryption as a case study, and we use search protocol **Srch** in place of **Query** and encrypted search protocol $\mathbf{Srch}_e$ in place of $\mathbf{Query}_e$ to emphasis that the query type supported by the database is keyword search query. We omit the subscription when it is clear that we are referring to an encrypted protocol.

We refine the structure of the documents as follows to make our constructions later more readable. A keyword-based database **DB** consists of a set of files, so $\mathbf{DB} = \{f_i\}$. Each file $f \in \mathbf{DB}$ contains a unique file identifier, some file content and some keywords specified by the user so $f = (id, doc, W)$. For simplicity, we denote file identifier of a file as $id = id(f)$, the document associated to a file $doc = doc(f)$, and the keywords associated to a file as $W = KW(f)$.

An keyword-based encrypted database **EDB** consists of a collection of encrypted files and an encrypted index **EI** for encrypted queries, so $\mathbf{EDB} = (\{ef_i\}, \mathbf{EI})$. Each encrypted file ef contains a file identifier and some encrypted file content so $ef = (id, edoc)$. For simplicity of notation, we denote file identifier of a file as $id = id(ef)$ and the set of keywords associated to an encrypted file by $W = KW(ef)$. For a search query $\mathbf{q}$, we write $KW(\mathbf{q})$ to mean the keyword associated to the query.

8.3.2 Additional Notation

Let $KW(\cdot)$ denotes a function that takes in a file and returns the set of keywords associated to it. We write $P_a^{KW(\mathbf{DB})}$ to mean a permutation on $KW(\mathbf{DB})$, i.e. a bijection between the keywords in the database, where a is a prime number and P_a is a product of $\frac{|KW(\mathbf{DB})|}{a}$ disjoint cycles of length a. When the keyword set is well-understood, we omit it from the notation. Let W be a set of keywords, we abuse the notation $P_a(W)$ to mean $P_a(W) = \{P_a(kw) \mid kw \in W\}$.

8.3.3 Typical Leakage Functions

We recall and formalise two typical leakage functions, namely search pattern and access pattern formally in this section, as our constructions have leakage functions derived from these.

Search Pattern. Search pattern refers to query equality leakage, meaning that given two search queries $\mathbf{q}_1$ and $\mathbf{q}_2$, an attacker is able to tell if the keyword associated to query $\mathbf{q}_1$ is the same as that of query $\mathbf{q}_2$. This leakage is typically represented as an l-by-l matrix, where l is the number of search queries and the (i, j)-th entry of the matrix is 1 if and only if the i-th search query and the j-th search query have the same

underlying keyword; the (i, j)-th entry is 0 otherwise. We denote search pattern as $\mathsf{SP}((\mathbf{q}_1, \ldots, \mathbf{q}_l), \mathbf{DB})$ in this thesis.

Access Pattern. Access pattern refers to equality leakage during file retrieval. Consider the case where file f contains keyword kw_1 and kw_2, and a query on each is executed. For a scheme that does not suppress access pattern leakage, an encrypted version of file f will be retrieved with both queries, hence, leaking the fact that the two keywords are present in the same file. The leakage is typically expressed as a $|\mathbf{DB}|$-by-l matrix, where $|\mathbf{DB}|$ is the number of files in the database and l is the number of queries. An order on the file identifiers is assumed so the indices make sense. The (i, j)-th entry of the matrix is 1 if and only if the i-th file is retrieved by the j-th query. We denote access pattern as $\mathsf{AP}((\mathbf{q}_1, \ldots, \mathbf{q}_l), \mathbf{DB})$ in this thesis.

We note that there are ways to represent search and access pattern. In the matrix representation above, each row of the matrix represents a query (or a document), and the order of rows is independent from the underlying queried keywords (or documents). Later in the thesis, we represent the access pattern leakage with the actual document identifiers to simplify the notation, and it should be noted that the actual document identifiers are not leaked by the constructions.

8.3.4 Motivation

As discussed in Section 4.5, access pattern from search queries can be abused by an adversary to recover keywords in documents or queries. The obvious approach to suppress the leakage is to use ORAM-based constructions [74] to eliminate access pattern leakage altogether. Given that ORAM is computationally costly, we seek for a scheme with greater leakage but which preserves enough privacy for the underlying scheme, that is, the scheme may not be completely leakage-free, but the information gains with respect to the important classes of leakage-abuse attacks in our framework are low.

All of our constructions rely on a permutation P_a. The idea is that the databases can be permuted by P_a in some way such that the encrypted view of the database stays the same. Therefore, the permuted databases and the original database are indistinguishable by an honest-but-curious adversary. We show that different ways of applying the permutation P_a leads to different security properties.

On a high-level, we propose three schemes, all based on some searchable encryption scheme that leaks only the number of keywords and the number of documents in the initialisation algorithm, and search pattern and access pattern in the searching phase. For simplicity, we only demonstrate how our schemes work in the vanilla setting where the underlying scheme supports keyword search and no update or deletion are allowed. One can easily extend our schemes to support these two operations. The first scheme uses the permutation P_a to generate fake files such that the keywords in the same cycle of P_a generate the same leakage pattern. The second scheme is a modified version of the first scheme by adding permutations of keywords to the real files instead of creating fake files. The third scheme adds fake keywords to the documents by using permutations of keywords from other files.

8.3.5 Our Constructions

Permutation by Adding Fake Documents. Let $\overline{\Sigma} = (\overline{\textbf{Setup}}, \overline{\textbf{Srch}})$ be a searchable encryption scheme with the number of documents and number of keyword-document pairs as the setup leakage, and search pattern and access pattern as the search leakage. Our construction $\Sigma_1 = (\textbf{Setup}, \textbf{Srch})$ pre-processes the plaintext database by adding fake documents and then uses $\overline{\Sigma}$ as the underlying searchable encryption scheme.

To add fake documents, the scheme Σ_1 begins by generating a random permutation P_a of $KW(\textbf{DB})$ with cycles of size a, where a is a prime. To pre-process the plaintext database, for every file f in the database, Σ_1 generates $a-1$ fake files with function **FGen**. **FGen** takes as input a set of keywords W and output a fake file with W as the keywords associated to the file. Here, the sets of keywords used to generate the fake files are $P_a(KW(f)), P_a^2(KW(f)), \ldots, P_a^{a-1}(KW(f))$. After the pre-processing step, scheme Σ_1 calls $\overline{\textbf{Setup}}$ to setup the database. To answer queries, scheme Σ_1 simply invokes $\overline{\textbf{Srch}}$ and filters fake documents locally. The pseudocode of the setup algorithm is shown in Algorithm 8.1.

Algorithm 8.1 **Setup** algorithm of Σ_1.

```
1: procedure Setup(1^λ, a, DB)
2:     Generate permutation P_a for keyword set KW(DB)
3:     DB' ← {}
4:     for f ∈ DB do
5:         DB' ← DB' ∪ f
6:         for i ← 1, ..., a − 1 do
7:             DB' ← DB' ∪ FGen(P_a^i(KW(f)))
8:     return Setup(1^λ, DB')
```

Permutation with Adding Fake Keywords. Let $\overline{\Sigma} = (\overline{\textbf{Setup}}, \overline{\textbf{Srch}})$ be a searchable encryption scheme with the number of documents and number of keyword-document pairs as the setup leakage, and search pattern and access pattern as the search leakage. Our construction $\Sigma_2 = (\textbf{Setup}, \textbf{Srch})$ pre-processes the plaintext database by adding fake keywords to the existing documents and then uses $\overline{\Sigma}$ as the underlying searchable encryption scheme.

To add fake documents, the scheme Σ_2 begins by generating a random permutation P_a of $KW(\textbf{DB})$ with cycles of size a, where a is a prime. To pre-process the plaintext database, for every file f in the database, $a-1$ new sets of keywords $P_a(KW(f))$, $P_a^2(KW(f)), \ldots, P_a^{a-1}(KW(f))$ are generated and added to file f. The resultant database is then encrypted and uploaded with $\overline{\textbf{Setup}}$. Similar to Σ_1, Σ_2 uses search protocol $\overline{\textbf{Srch}}$ to retrieve documents and filter false-positives locally. The pseudocode of the setup algorithm is shown in Algorithm 8.2.

Permutation with a Documents. Let using the same scheme $\overline{\Sigma} = (\overline{\textbf{Setup}}, \overline{\textbf{Srch}})$ as the underlying searchable encryption scheme, our last scheme $\Sigma_3 = (\textbf{Setup}, \textbf{Srch})$ obfuscates access pattern by using a permutation P_a on batches of documents as follows. During the setup phase, Σ_3 generates a permutation P_a of $KW(\textbf{DB})$ with cycles of size a, where a is a prime. Σ_3 then randomly batches the files into groups of a. Without loss of generality, let $f_0, \ldots, f_{a-1}$ be the files in a batch. Σ_3 sets the keywords associated to

Algorithm 8.2 Setup algorithm of Σ_2.

procedure Setup$(1^\lambda, a, \mathbf{DB})$
  Generate permutation P_a for keyword set $KW(\mathbf{DB})$
  $\mathbf{DB}' \leftarrow \{\}$
  for $f \in \mathbf{DB}$ **do**
    $W \leftarrow KW(f)$
    for $i \leftarrow 1, \ldots, a-1$ **do**
      $W \leftarrow W \cup P_a^i(KW(f))$
    $\mathbf{DB}' \leftarrow \mathbf{DB}' \cup (id(f), doc(f), W)$
  return $\overline{\mathbf{Setup}}(1^\lambda, \mathbf{DB}')$

file f_i to

$$\bigcup_{j=0}^{a-1} P_a^{i-j}(KW(f_j)).$$

It is easy to see that the keywords associated to f_i are contained in this set for all i, and the padded keyword sets for the files in the same batch are permutations of each other. The pseudocode of the setup algorithm is shown in Algorithm 8.3.

Algorithm 8.3 Setup algorithm of Σ_3.

procedure Setup$(1^\lambda, a, \mathbf{DB})$
  Generate permutation P_a for keyword set $KW(\mathbf{DB})$
  $\mathbf{DB}' \leftarrow \{\}$
  for $f_0, \ldots, f_{a-1}$ sampled from $\mathbf{DB}$ without replacement **do**
    $W \leftarrow \bigcup_{j=0}^{a-1} P_a^{-j}(KW(f_j))$
    for $i \leftarrow 0, \ldots, a-1$ **do**
      $\mathbf{DB}' \leftarrow \mathbf{DB}' \cup (id(f_i), doc(f_i), P_a^i(W))$
  return $\overline{\mathbf{Setup}}(1^\lambda, \mathbf{DB}')$

8.4 Security Analysis of Our Constructions

In this section, we prove the security of the three schemes we have devised earlier with respect to **SS-CQA**, and analyse their security with respect to three security goals in our security model. We demonstrate the usefulness of our framework in measuring the level of privacy a scheme offers on different aspects, including query privacy and data privacy.

8.4.1 Leakage of our Constructions

In this section, we present the leakages of the three schemes and prove their security in the **SS-CQA** model. All of our constructions are built from a searchable encryption scheme $\overline{\Sigma} = (\overline{\mathbf{Setup}}, \overline{\mathbf{Srch}})$ with leakage profile:

$$\mathcal{L}_{\overline{\mathbf{Setup}}}(\mathbf{DB}) = \left(|\mathbf{DB}|, \sum_i |KW(\mathbf{DB}[i])| \right),$$

$$\mathcal{L}_{\overline{\mathbf{Srch}}}(\mathbf{q}, \mathbf{DB}; \mathbf{q}_1, \ldots, \mathbf{q}_l) = (\mathsf{SP}_{\overline{\Sigma}}((\mathbf{q}_1, \ldots, \mathbf{q}_l, \mathbf{q}), \mathbf{DB}), \mathsf{AP}_{\overline{\Sigma}}((\mathbf{q}_1, \ldots, \mathbf{q}_l, \mathbf{q}), \mathbf{DB})).$$

As the only difference between $\overline{\Sigma}$ and our schemes is the pre-processing step, our schemes have the same leakage profile as $\overline{\Sigma}$, except that the input database $\mathbf{DB}$ is replaced with a padded version $\mathbf{DB}'$ according to the pre-processing algorithms.

Leakage Profile of Σ_1. Recall that our first construction simply generates $a-1$ fake files for every real file, with the keywords permuted by different powers of P_a. This means that even though the encrypted database contains $a-1$ times more encrypted files and keyword-file pairs, the setup leakage of the construction does not change, as an attacker can simply divide the size of the database and the number of keyword-file pairs to recover the original size of the database and the number of keyword-file pairs.

The search protocol of the construction leaks search pattern just as before. In terms of the access pattern, fake files are introduced to mask the real access pattern. As we have discussed before, access pattern leakage can be represented as a matrix, where the rows of the matrix represent the files and the columns of the matrix represent the queries. The order of rows is independent from the order of files in the original database. This however, adds additional complications to the representation and we decide to represent the access pattern leakage with the file identifiers even though they are not leaked directly. One can think of the real access pattern leakage as a row-permuted version of our representation.

For simplicity, we assume $\mathbf{DB}[i]$ is the i-th file in the original database, and it is mapped to the $(ai-a+1)$-th file in the encrypted file in the encrypted database with permutation P_a. We also assume that the fake files generated from the i-th file are mapped to the $(ai-a+2)$-th to (ai)-th files in the encrypted database. Then the access pattern leakage of the construction can be written as:

$$\mathsf{AP}_{\Sigma_1}((\mathbf{q}_1,\ldots,\mathbf{q}_l,\mathbf{q}),\mathbf{DB})_{i,j} = \begin{cases} 1 \text{ if } KW(\mathbf{q}_j) \in P_a^{(i-1 \bmod a)}\left(KW\left(\mathbf{DB}[\lceil \tfrac{i}{a} \rceil]\right)\right), \\ 0 \text{ otherwise.} \end{cases}$$

Leakage Profile of Σ_2. Recall that for scheme Σ_2, the permuted keywords are padded directly to the real files using permutation P_a. The resultant files may contain less than a times of keywords as the permutations may contain the same keywords as the original files. This means that the setup leakage of Σ_2 is slightly different from $\overline{\Sigma}$. We can describe the setup leakage of Σ_2 formally with

$$\mathcal{L}_{\mathbf{Setup}}(\mathbf{DB}) = \left(|\mathbf{DB}|, \sum_i \left|\bigcup_{j=0}^{a-1} P_a^j(KW(\mathbf{DB}[i]))\right|\right).$$

In terms of leakage from search queries, Σ_2 leaks search pattern and a modified access pattern. The access pattern leakage can be expressed as:

$$\mathsf{AP}_{\Sigma_2}((\mathbf{q}_1,\ldots,\mathbf{q}_l,\mathbf{q}),\mathbf{DB})_{i,j} = \begin{cases} 1 \text{ if } KW(\mathbf{q}_j) \in \bigcup_{k=0}^{a-1} P_a^k\left(KW(\mathbf{DB}[i])\right), \\ 0 \text{ otherwise.} \end{cases}$$

Leakage Profile of Σ_3. Our scheme Σ_3 batches files together randomly and apply padding with permutation P_a. This means that the leakage is randomised by how the files are batched. In our description below, we simplify the leakage profile by assuming a fixed ordering of files.

The setup leakage of the scheme includes the number of files and the total number of keyword-file pairs after padding.

$$\mathcal{L}_{\mathbf{Setup}}(\mathbf{DB}) = \left(|\mathbf{DB}|, \sum_{i=0}^{\frac{|\mathbf{DB}|}{a}-1} \left| \bigcup_{j=1}^{a} P_a^{j-1}(KW(\mathbf{DB}[ai+j])) \right| \right).$$

The search leakage of the scheme includes search pattern and access pattern induced by the padded database. The access pattern leakage can be expressed as:

$$\mathsf{AP}_{\Sigma_3}((\mathbf{q}_1, \ldots, \mathbf{q}_l, \mathbf{q}), \mathbf{DB})_{i,j} = \begin{cases} 1 \text{ if } KW(\mathbf{q}_j) \in \bigcup_{k=a\lceil \frac{i}{a} \rceil}^{a\lceil \frac{i}{a} \rceil + a} P_a^{i-k}\left(KW(\mathbf{DB}[k])\right), \\ 0 \text{ otherwise.} \end{cases}$$

The setup leakage stays the same if we have not made the assumption on the ordering of files. The access pattern, on the other hand, will be a row-permuted version of the matrix above if the assumption is not made.

8.4.2 Invariant Properties of our Constructions

In this section, we show an important property of our constructions which we need in our security analysis with gain functions later.

Indistinguishable Leakage Profiles under Permutation. Let P_a be the permutation used to pre-process the plaintext database $\mathbf{DB}$. Our schemes are constructed in a way that the keywords of the files in the database can be permuted by P_a and generate the same leakage profile. It is also possible to permute the keywords in the queries and generate the same search pattern and access pattern leakages.

Taking scheme Σ_1 as an example. Consider a simple database with a single file f_1. During the pre-processing step, fake files $f_2, \ldots, f_a$ are generated such that the keywords associated to the files are $P_a^j(KW(f_1))$ for $j = 1, \ldots, a-1$. By permuting the keywords associated to file f_1 with P_a, we get that the new set of keywords is $P_a(KW(f_1))$. If that is the set of keywords associated to f_1 instead, we will get the sets of keywords associated to the fake files as $P_a^{j+1}(KW(f_1))$ for $j = 1, \ldots, a-1$. However, the set of sets of keywords from the real file and the fake files together does not change, as it stays as $P_a^j(KW(f_1))$ for $j = 0, \ldots, a-1$. This means that an attacker who can only observe the leakages from the setup step and the subsequent queries is not able to tell if the set of keywords associated to the encrypted version of f_1 is $KW(f_1)$ or any other $P_a^j(KW(f_1))$ for $j = 1, \ldots, a-1$, as any of these choices generate the same leakage profile. We call this property invariant under keyword sets permutation.

From now on, we abuse the notion $P_a(\mathbf{DB})$ to mean permuting the keyword sets of the files in $\mathbf{DB}$, or

$$P_a(\mathbf{DB}) = \{(id(f_i), P_a(KW(f_i)), f_i)\}.$$

Invariant under Keyword Sets Permutation. We are now ready to formally define invariant under keyword sets permutation.

Definition 8.3 (Invariant under Keyword Sets Permutation)**.** Let Σ be a searchable encryption scheme with leakage profile $(\mathcal{L}_{\mathbf{Setup}}, \mathcal{L}_{\mathbf{Srch}})$. We say that Σ is invariant under

keyword sets permutation if for all databases $\mathbf{DB} = \{f_i\}$, there exists a permutation P on the database $\mathbf{DB}$ which may depend on the particular instance of the setup procedure, such that

$$(\mathcal{L}_{\mathbf{Setup}}(\mathbf{DB}), \mathcal{L}_{\mathbf{Srch}}(\cdot, \mathbf{DB})) = (\mathcal{L}_{\mathbf{Setup}}(P(\mathbf{DB})), \mathcal{L}_{\mathbf{Srch}}(\cdot, P(\mathbf{DB}))).$$

We state the invariant under keyword sets permutation property of our constructions as a theorem below.

Theorem 8.2 (Invariant under Keyword Sets Permutation Property of our Constructions)**.** Schemes Σ_1, Σ_2 and Σ_3 are invariant under keyword sets permutation.

Proof. (Scheme Σ_1) Assume P_a is the permutation used in the setup step. It should be immediately clear that the setup leakage does not change after an application of permutation P_a, as

$$\begin{aligned}
\mathcal{L}_{\mathbf{Setup}}(P_a(\mathbf{DB})) &= \left(\frac{|P_a(\mathbf{DB})|}{a}, \frac{\sum_i \sum_{j=0}^{a-1} \left| P_a^{j+1}(KW(\mathbf{DB}[i])) \right|}{a} \right) \\
&= \left(\frac{|\mathbf{DB}|}{a}, \frac{\sum_i \sum_{j=0}^{a-1} \left| P_a^{j}(KW(\mathbf{DB}[i])) \right|}{a} \right) \\
&= \mathcal{L}_{\mathbf{Setup}}(\mathbf{DB})
\end{aligned}$$

Search query leakage of scheme Σ_1 consists of two parts, namely search pattern and access pattern. Search pattern is not affected by the keyword sets permutation so it stays the same after the permutation. In terms of access pattern, the permutation of keywords induces a permutation on the document identifiers. If we can prove that, it is straightforward to see that the access-pattern stays the same as access pattern is invariant under document identifier permutation.

To prove the permutation, consider without loss of generality a database $\mathbf{DB} = \{f_1\}$, where $KW(f_1) = W$. The setup algorithm generates files $f_2, \ldots, f_a$ such that $KW(f_i) = P_a^{i-1}(W)$.By applying permutation P_a on database $\mathbf{DB}$ and running the setup algorithm, we get files $\{f_i'\}$, such that $KW(f_i') = P_a^i(W)$. In other worlds, $KW(f_i) = KW(f_{i-1}')$ for $i = 2, \ldots, a$ and $KW(f_1) = KW(f_a')$. This can be seen as a permutation on the document identifiers in terms of access pattern leakage. By generalising the argument to a database with multiple files, we get the desired result.

(Scheme Σ_2) Assume P_a is the permutation used in the setup step. If f is a file in the database $\mathbf{DB}$, the pre-processing step changes its keywords to $\bigcup_{i=0}^{a-1} P_a^i(KW(f))$. It should be clear that a permutation with P_a on the keywords of f does not change the keywords:

$$\begin{aligned}
& P_a \left(\bigcup_{i=0}^{a-1} P_a^i(KW(f)) \right) \\
&= \bigcup_{i=1}^{a} P_a^i(KW(f)) \\
&= \bigcup_{i=0}^{a-1} P_a^i(KW(f)).
\end{aligned}$$

So the setup and search leakage are not affected by a keyword sets permutation.

(Scheme Σ_3) The invariant property of scheme Σ_3 can be proven in a similar way to Σ_1 by treating some of the pre-processed documents as the original documents in Σ_1. Suppose $f_1, \ldots, f_a$ is a batch of files chosen in the setup step of Σ_3. Then the keywords associated to file f_1 after pre-processing is $\bigcup_{i=1}^{a} P_a^{1-i} KW(f_i)$, and it can be verified easily that $P_a(KW(f_i')) = KW(f_{i+1}')$ for all $i = 1, \ldots, a-1$, and $P_a(KW(f_a')) = KW(f_1')$. This means files $f_2, \ldots, f_a$ can be treated as the fake files in Σ_1, and the proof follows immediately. □

Invariant under Queried Keywords Permutation. Similarly, the queried keywords can be permuted without changing the query leakage. In a naive construction, this is certainly not the case: a query on keyword kw_1 and a query on keyword kw_2 are going to return different numbers of files as long as the number of times these two keywords appear in the database are different. This is not the case in our constructions. Suppose P_a is the permutation used to pre-process a database $\mathbf{DB}$. In scheme Σ_1, for every instance of keyword kw_1, the scheme generates a fake file containing keyword $P_a(kw_1)$. There are also fake documents generated containing keywords $P_a^2 * (kw_1)$ and so on. That means the query response volume for the pre-processed database is the total number of documents containing $kw_1, P_a(kw_1), \ldots, P_a^{a-1}(kw_1)$. By the same argument, if keyword kw_2 is in the same cycle as keyword kw_1 in permutation P_a, the query response volume associated to a query on keyword kw_2 is $kw_2, P_a(kw_2), \ldots, P_a^{a-1}(kw_2)$. But since we know $kw_2 = P_a^j(kw_1)$ for some j, the query response volume of a query with keyword kw_2 is the same as that of keyword kw_1.

The indistinguishability property does not stop at query response volumes. In fact, the access pattern matrix will not change after a permutation of queried keywords. This can be seen easily as a permutation on the queried keywords corresponds directly to a permutation of file identifiers. Using scheme Σ_1 as an example. If file f_1 contains keyword kw_1 and kw_2, the pre-processing step introduces fake files $f_2, \ldots, f_a$ such that the keyword in f_i is $P_a^{i-1}(\{kw_1, kw_2\})$. If a permutation of P_a is applied on the queried keywords, then a query on keyword kw_1 is changed to a query on $P_a(kw_1)$. So f_2 is returned instead of file f_1. Similarly, a query on keyword kw_2 is changed to a query on $P_a(kw_2)$ and file f_2 is returned instead of file f_1. In fact, P_a fixes a permutation on the file identifiers in terms of the access pattern. This corresponds to a row permutation of the access pattern matrix, and since it does not change the information carried by the access pattern matrix, we say that Σ_1 is invariant under queried keywords permutation.

By abuse of notation, we write $P_a(\mathbf{q})$ to mean a permutation on the queried keyword associated to $\mathbf{q}$, and the resultant queried keyword is $P_a(KW(\mathbf{q}))$.

We are now ready to formally define invariant under queried keywords permutation.

Definition 8.4 (Invariant under Queried Keywords Permutation)**.** Let Σ_1 be a searchable encryption scheme with leakage profile $(\mathcal{L}_{\mathbf{Setup}}, \mathcal{L}_{\mathbf{Srch}})$. We say that Σ_1 is invariant under keyword sets permutation if for all databases $\mathbf{DB} = \{f_i\}$ and all sequences of search queries $\mathbf{q}_1, \ldots, \mathbf{q}_l$, there exists a permutation P which may depend on the particular instance of the setup procedure, such that

$$\mathcal{L}_{\mathbf{Srch}}(\mathbf{q}_1, \ldots, \mathbf{q}_l, \mathbf{DB}) = \mathcal{L}_{\mathbf{Srch}}(P(\mathbf{q}_1), \ldots, p(\mathbf{q}_l), \mathbf{DB}).$$

We state the invariant under queried keywords permutation property of our constructions as a theorem below.

Theorem 8.3 (Invariant under Queried Keywords Permutation Property of our Constructions)**.** Schemes Σ_1, Σ_2 and Σ_3 are invariant under queried keywords permutation.

Proof. (Scheme Σ_1) Assume P_a is the permutation used in the setup step. As before, the search pattern is not affected by the permutation on the queried keywords. In terms of the access pattern, as before, let f_1 be a file in the original database, and $f_2, \ldots, f_a$ be the fake files generated by the setup step. Without loss of generality, let kw be a keyword in f_1. After an application of permutation P_a on a query associated to kw, we get a query on $P_a(kw)$. This leads to retrieval of f_2 instead of f_1. We get the same permutation on document identifiers just as before, so Σ_1 is invariant under queried keywords permutation.

(Scheme Σ_2) Assume P_a is the permutation used in the setup step. We note that a pre-processed file f contains kw if and only if it contains $P_a(kw)$ too. This means a query on kw retrieves the exact set of files as a query on $P_a(kw)$. This implies that Σ_2 is invariant under queried keywords permutation.

(Scheme Σ_3) The proof for scheme Σ_3 follows from the argument in proof of Theorem 8.2 so it is omitted here. □

8.4.3 Security Analysis through Gain Functions

In this section, we demonstrate how to use gain functions to analyse security of our constructions. We consider three gain functions. The first gain function g_1 is used to measure query privacy, whereas the other two are used to measure data privacy.

Measuring Query Privacy with g_1. An important class of attacks on searchable encryption is query reconstruction attacks. These attacks [94, 33, 17] assume that the adversary has access to the underlying database or a noisy version of it, and the goal of the adversary is to recover the keywords associated to the queries. In our security notion, this can be modelled as an environment $\mathcal{Z}$ which outputs auxiliary information $\mathbf{aux} = \mathbf{DB}_0$, where $\mathbf{DB}_0$ is the underlying database. The goal of the adversary is to maximise its gain with respect to the function:

$$g_1((j, kw), (\mathbf{DB}_i), (\mathbf{q}_i)) = \mathbb{1}(KW(\mathbf{q}_j) = kw).$$

As the auxiliary information does not include any information on the actual queries, it is easy to see that no adversary can achieve an expected gain of more than $\frac{1}{a}$ on our constructions, as the constructions are invariant under queried keywords permutation. We state the security of our constructions with respect to gain function g_1 in Theorem 8.4.

Theorem 8.4 (Security of our Constructions with respect to Gain Function g_1)**.** Let scheme Σ be either Σ_1, Σ_2 or Σ_3, and a the size of the cycles in the permutation P_a generated by the schemes. Let environment $\mathcal{Z}$ be one that generates a database and a sequence of queries uniformly randomly, with auxiliary information **aux** the same as the generated database, on a polynomial-size support. Then

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathrm{Real}^{g_1}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right] \leq \frac{1}{a} + \mathtt{negl}(\lambda).$$

Proof. We know from Theorem 8.3 that schemes Σ_1, Σ_2 and Σ_3 are invariant under queried keywords permutation. Since the queries are uniformly generated, and no information on query distribution is passed to the adversary, a sequence of queries $\mathbf{q}_1, \ldots, \mathbf{q}_l$ generates the same leakage as $P_a^i(\mathbf{q}_1), \ldots, P_a^i(\mathbf{q}_l)$ for $i = 1, \ldots, a$. These possibilities are equally likely. Hence for scheme Σ which is either Σ_1, Σ_2 or Σ_3, and any simulator $\mathcal{S}$ which satisfies the **B-BB** notion,

$$
\begin{aligned}
&\sup_{\mathcal{A}} \mathbf{E}\left[\mathcal{S}^{g}_{\Sigma,\mathcal{A},\mathcal{Z},\mathcal{S}}(1^\lambda)\right] \\
\leq &\sum_{(\mathcal{T}_i)} \mathsf{Pr}\left[(\mathcal{T}_i)\right] \max_{(j,kw)} \sum_{((\mathbf{DB}_i),(\mathbf{q}_i),\mathbf{aux})} g_1((j,kw),(\mathbf{DB}_i),(\mathbf{q}_i)) \cdot \mathsf{Pr}\left[((\mathbf{DB}_i),(\mathbf{q}_i)) \mid (\mathcal{T}_i)\right] \\
\leq &\sum_{(\mathcal{T}_i)} \mathsf{Pr}\left[(\mathcal{T}_i)\right] \cdot \frac{1}{a} \\
= &\frac{1}{a}.
\end{aligned}
$$

Since the environment $\mathcal{Z}$ samples from a polynomial-sized sample space as assumed by Theorem 8.4, and the image of g_1 contains only two values, we get

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathrm{Real}^{g_1}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right] \leq \frac{1}{a} + \mathtt{negl}(\lambda)$$

using Theorem 8.1. □

Measuring Data Privacy with g_2. Another important class of attacks on searchable encryption is data reconstruction attacks. These attacks assume that the adversary has some information on the database and queries, and its goal is to recover one or more keywords of the encrypted files. This attack is devastating as the recovered keywords can be used to decipher the content of the encrypted files and completely defeat the purpose of file encryption. The query reconstruction attacks in the literature [94, 33, 17] are data reconstruction attacks too on the schemes that leak access pattern.

We can capture data reconstruction attacks with an environment $\mathcal{Z}$ that generates a database and a sequence of queries at random, and set the auxiliary information to the access pattern of the database (with known queries) in the naive construction. The reason for this choice of auxiliary information is because a data reconstruction attack is pointless if the attacker is given the database to begin with. The difference between the access pattern and the actual database is that the file identifiers are hidden, so there is uncertainty in the plaintext database. We note that a data reconstruction attack can be sensitive to the underlying data. For example, if all of the files in the database contain keyword kw, then an adversary can easily guess a keyword in any file. To make it more interesting, we restrict the environment $\mathcal{Z}$ to generate databases with files such that for all files f, if $kw \in f$, then $P_a^j(kw) \notin KW(f)$, for $j = 1, \ldots, a-1$, where P_a is the permutation generated by the underlying scheme.

We propose to use

$$g_2((ef, kw), (\mathbf{DB}_i), (\mathbf{q}_i)) = \mathbb{1}\left(kw \in KW(\mathbf{DB}_0[id(ef)])\right)$$

as the gain function to measure the success of the adversary, where ef in the adversary's guess is an encrypted file which comes from one of the transcripts. Intuitively, gain

function g_2 measures the ability of an adversary at guessing a single keyword within a file correctly.

There is a slight complication with gain function g_2 on scheme Σ_1 as not all encrypted files are real files. Hence, for scheme Σ_1, if the guess of the adversary targets a fake file, the gain function assumes 0 immediately.

We are now ready to give an upper bound of the expected gain for our constructions in Theorem 8.5.

Theorem 8.5 (Security of our Constructions with respect to Gain Function g_2)**.** Let $\overline{\Sigma}$ be the underlying searchable encryptions scheme used by our constructions. Let scheme Σ be either Σ_1, Σ_2 or Σ_3, and a the size of the cycles in the permutation P_a generated by the schemes. Let environment $\mathcal{Z}$ be one that generates a database $\mathbf{DB}$ and a sequence of queries uniformly randomly, such that for all files f, if $kw \in f$, then $P_a^j(kw) \notin KW(f)$, for $j = 1, \ldots, a-1$. Let the auxiliary information output by environment $\mathcal{Z}$ be $\mathsf{AP}_{\overline{\Sigma}}(\cdot, \mathbf{DB})$. If the support of $\mathcal{Z}$ is polynomial-sized, then

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathrm{Real}^{g_2}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right] \leq \frac{1}{a} + \mathtt{negl}(\lambda).$$

Proof. In the theorem statement, we assume that for all files f, if $kw \in KW(f)$, then $P_a^j(kw) \notin KW(f)$, for $j = 1, \ldots, a-1$. This implies that the maximum number of occurrences of a keyword is less than or equal to $\frac{|\mathbf{DB}|}{a}$. This is to prevent naive keyword guessing attacks.

Consider a guess (ef, kw) from the adversary. Assume without loss of generality that $KW(ef) = W$. Given that Σ_1, Σ_2 and Σ_3 are invariant under keyword sets permutation, and the auxiliary information only includes access pattern of the database, the leakage observed by the adversary is equally likely to be generated from $P_a^i(W)$ for $i = 1, \ldots, a-1$. Hence, the chance of the adversary making the right guess is less than or equal to $\frac{1}{a}$ for any encrypted file and keyword. Formally, this can be expressed as

$$\begin{aligned}
&\sup_{\mathcal{A}} \mathbf{E}\left[\mathcal{S}^{g}_{\Sigma,\mathcal{A},\mathcal{Z},\mathcal{S}}(1^\lambda)\right] \\
\leq &\sum_{(\mathcal{T}_i)} \mathsf{Pr}\left[(\mathcal{T}_i)\right] \max_{(ef,kw)} \sum_{((\mathbf{DB}_i),(\mathbf{q}_i),\mathbf{aux})} g_1((ef,kw),(\mathbf{DB}_i),(\mathbf{q}_i)) \cdot \mathsf{Pr}\left[((\mathbf{DB}_i),(\mathbf{q}_i)) \mid (\mathcal{T}_i)\right] \\
\leq &\sum_{(\mathcal{T}_i)} \mathsf{Pr}\left[(\mathcal{T}_i)\right] \cdot \frac{1}{a} \\
= &\frac{1}{a},
\end{aligned}$$

for scheme Σ which is either Σ_1, Σ_2 or Σ_3, and any simulator $\mathcal{S}$ which satisfies the **B-BB** notion.

Since the environment $\mathcal{Z}$ samples from a polynomial-sized sample space as assumed by Theorem 8.5, and the image of g_2 contains only two values, we get

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathrm{Real}^{g_1}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right] \leq \frac{1}{a} + \mathtt{negl}(\lambda)$$

using Theorem 8.1. □

MEASURING DATA PRIVACY WITH g_3. Under the same conditions as above, there are other ways we can measure data privacy. Here, we consider an adversary whose goal is to recover all keywords of a file. And instead of a one-or-nothing gain function, we can use a more expressive gain function. For example, we set gain function g_3 to be:

$$g_3((ef, W), (\mathbf{DB}_i), (\mathbf{q}_i)) = 1 - \frac{|W \triangle KW(\mathbf{DB}_0[id(ef)])|}{|KW(\mathbf{DB}_0[id(ef)])|}.$$

Intuitively, this gain function measures security impact of a guess by looking at the symmetric difference between the keywords in the guess and the actual keywords in the target file. If the symmetric difference is large, it means that either the attacker missed some keywords in the target file or included some keywords in his guess that are not in the target file. In those cases, the gain function is small. On the other hand, if the adversary has guessed the keywords of his target file perfectly, the gain function attains 1.

It should be immediately clear that the upper bound of the expected gain cannot exceed $\frac{1}{a}$, due to Theorem 8.5. We state this as a theorem below.

Theorem 8.6 (Security of our Constructions with respect to Gain Function g_3)**.** Let $\overline{\Sigma}$ be the underlying searchable encryptions scheme used by our constructions. Let scheme Σ be either Σ_1, Σ_2 or Σ_3, and a the size of the cycles in the permutation P_a generated by the schemes. Let environment $\mathcal{Z}$ be one that generates a database $\mathbf{DB}$ and a sequence of queries uniformly randomly, such that for all files f, if $kw \in KW(f)$, then $P_a^j(kw) \notin KW(f)$, for $j = 1, \ldots, a-1$. Let the auxiliary information output by environment $\mathcal{Z}$ be $\mathsf{AP}_{\overline{\Sigma}}(\cdot, \mathbf{DB})$. If the support of $\mathcal{Z}$ is polynomial-sized, then

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathrm{Real}^{g_3}_{\Sigma,\mathcal{A},\mathcal{Z}}(1^\lambda)\right] \leq \frac{1}{a} + \mathtt{negl}(\lambda).$$

8.5 Application of Our Notion to Other Schemes

Our security notion is flexible to analyse security of different schemes and with respect to different security goals. To illustrate this, we revisit the attacks we have presented in Chapter 5, 6 and 7 in this new framework. We also show how our framework can be applied to prove (in)security of other schemes in the literature.

8.5.1 Security of Encrypted Range Queries

In Chapter 5, we demonstrate how to exploit access-pattern and volume leakage of encrypted range queries and achieve database reconstruction and distribution reconstruction. These attack goals can be easily converted into a gain function and the corresponding attacks show lower bounds for the expected gains. Whenever the expected gain is large, we can conclude that the scheme is insecure with respect to the attack goal.

An interesting attack we have in Chapter 5 is the partial (distribution) reconstruction in Section 5.3.4.1. There, the goal of the adversary is to reconstruct the database as much as possible right before the partial solution becomes ambiguous. There are many gain functions we can use in this case to measure the success of an adversary (or the security of a scheme). For example, the gain function may measure the size of the solution

(how many indices does it cover) or the coverage of the solution (how many records does it contain). Our attacks in the HCUP dataset does moderately well in terms of the first gain function and exceptionally well in terms of the second one. In a broader view, understanding different gain functions and their implications can lead to different security and efficiency trade-offs for structured encryption.

8.5.2 Security of Searchable Encryption

In Chapter 6, we show how system-wide leakage from searchable encryption schemes can be exploited. From our experiments in Section 6.5, it can be seem clearly that the query recovery rate varies with the experimental setting. This suggests that some of the schemes may be secure in certain use-cases. However, to prove it concretely, we need to use the security notion we proposed in this chapter.

One important research direction is to understand query distributions in the real world and how valuable each query is. This allows for design of better informed gain functions and more efficient searchable encryption scheme with respect to those gain functions. As an example, if the keywords with high frequency are not so valuable, we can reduce the gain of the guesses on those queried keywords in the gain function. Then a scheme does not need to use expensive techniques to protect the corresponding queries which can lead to better efficiency.

8.5.3 Security of SWiSSSE

In Chapter 7, we propose SWiSSSE and demonstrate its security via attacks. Typically, one can only use an attack to show a lower bound on a particular gain function on a scheme. However, in the case of SWiSSSE, the attack we propose is statistically optimal, meaning that it is the best possible attack given the input. Hence, our attack, in fact, can be used as a tool to show a tight bound on the expected gain.

A caveat is that although the attack is statistically optimal, its convergence is not guaranteed. That is, the weighted gain (over different databases and queries generated from the specified distribution) output by our attack may be smaller than the true expected gain. To tackle this problem, we can use approximation algorithms with performance guarantees [101, 10] whenever possible to derive a concrete upper bound on the expected gain.

8.5.4 Security of the IKK construction

Without going into too much detail, the IKK construction [94] pads keywords to the documents so that there is a partition of keywords, and for each partition, if a keyword in the partition appears in some document, all other keywords in the same partition must appear in the same document. The padding is done statically for the minimal expansion of the index size and the authors have not proposed how insertions and deletions can be done. We refer to the IKK construction as `IKK` in our proofs.

The scheme is very comparable to bucketisation and `scheme2` and has very similar security properties with respect to keyword guessing attacks. In particular, if kw_1 and kw_2 forms a bucket but there is only one document with those two keywords, then the padding suggested by IKK does not hide the inclusion of kw_1 and kw_2 in that document.

On the other hand, by controlling the size of the partitions, one can achieve different levels of security with respect to query guessing attacks. We show the security statement formally below.

Definition 8.5 (Keyword Partition)**.** Let W be the set of keywords over all documents in the database. Let $\{S_i\}_i$ be sets such that

1. $\forall i \neq j, S_i \cap S_j = \emptyset$,
2. $\bigcup_i S_i = W$,
3. $\forall i, |S_i| \geq a$.

We call $\{S_i\}_i$ an a-partition of W.

Definition 8.6 (Leakage of the `IKK` construction)**.** Let $\mathbf{DB}$ be a database and W be the set of keywords. Then the `IKK` construction produces a database $\mathbf{DB}'$ with some a-partition $\{S_i\}_i$ that satisfies

1. $\forall j, KW(\mathbf{DB}[j]) \subseteq KW(\mathbf{DB}'[j])$,
2. $\forall kw \in W, \forall doc \in \mathbf{DB}', kw \in KW(doc) \implies S_j \subseteq KW(doc)$ where $kw \in S_j$.

Leakage of the `IKK` construction includes search pattern leakage and access pattern leakage induced by $\mathbf{DB}'$.

Theorem 8.7 (Security of the IKK construction with respect to query guessing attack)**.** Let $\{S_i\}_i$ be an a-partition. Let $\mathcal{Z}$ be the environment in our security notion. Suppose that $\mathbf{DB}$ generated by $\mathcal{Z}$ has polynomial size. Assume that the distribution of database $\mathbf{DB}$ generated by $\mathcal{Z}$ is uniform from its support. We define the auxiliary information $\mathbf{aux}$ generated by the environment as follows:

$$\mathbf{aux} = \{(W, |\mathbf{DB}(W)|) \mid W \subseteq KW(\mathbf{DB})\},$$

where $\mathbf{DB}(W)$ denotes the number of files contain exactly keywords W. We assume $\mathcal{Z}$ select queries uniformly randomly and all keywords have been queried at least once before the adversary $\mathcal{A}$ is required to output an answer.

Let g_1 be the gain function defined in Section 8.4.3. Then

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathbf{Real}^{g_1}_{\mathtt{IKK},\mathcal{A},\mathcal{Z}}(n)\right] \leq \frac{1}{a} + \mathtt{negl}(\lambda). \tag{8.2}$$

Proof. Without loss of generality, let kw_1 and kw_2 be two keywords from some partition S_i. Let $(\mathbf{q}_i)$ be a sequence of queries on some padded database $\mathbf{DB}'$ specified by the IKK construction. If queries for kw_1 and kw_2 are swapped, the access pattern stays the same, as $\forall f \in \mathbf{DB}', kw_1 \in KW(f) \iff kw_2 \in KW(f)$. Therefore, for any guess (j, kw) the adversary makes, it is equally likely for $KW(\mathbf{q}_j)$ to be any keyword in the partition containing kw. Given that $\{S_i\}_i$ is an a-partition, it must be the case that $\sup_{\mathcal{A}} \mathbf{E}\left[\mathrm{Ideal}^{g_1}_{\mathtt{IKK},\mathcal{A},\mathcal{Z}}(n)\right] \leq \frac{1}{a}$ and the desired result follows. □

8.5.5 Security of GBC

Unlike padding the database with fake keywords like IKK has suggested, Liu et al. [122] proposed to group queries together so that when the user issues some query, other queries in the same group are issued at the same time. As the queries in the same group appears together all the times, it is trivial that no adversary with no auxiliary information can guess the identity of the queries with probability greater than the inverse of the group size. For this reason, we omit the proof in the paper.

On the other hand, GBC is vulnerable to keyword guessing attacks shown in the previous section. This can be shown with a data-processing inequality [13, 4] argument. The authors of GBC proved that the leakage of the scheme includes the access pattern leakage and group pattern. Previous attacks, including the IKK attack [94] have demonstrated that access pattern leakage can be used to recover the keywords in the documents, so GBC with a larger leakage cannot be secure against these attacks.

8.5.6 Security of Differentially-private Volume Hiding

Differential privacy [5] is a popular concept in statistics to hide information about individuals in a dataset. It was adopted by Patal et al. [149] in their volume-hiding multi-map construction. We have shown that the differentially-private volume-hiding multi-map in [149] is insecure in Chapter 6. Recall that in the construction, if n_{kw} is the true response length of a query on keyword kw, the construction pads the response length to $\hat{n}_{kw} = n_{kw} + n^* + \mathbf{Lap}_{\mathsf{sk}}(2/\epsilon)$, where n^* is a parameter set by the client to offset the query response length in case the latter random variable is negative, and $\mathbf{Lap}_{\mathsf{sk}}(\cdot)$ is a Laplace distribution with secret key sk as the seed.

In this section, we show that differentially-private volume hiding as a technique is vulnerable to query reconstruction attacks under certain conditions, even without the presence of system-wide leakage which is exploited in Chapter 6. In this attack setting, we assume that the auxiliary information consists of some background information of the database, namely the keywords, their frequencies, and the associated uncertainties, i.e. $\{(kw_i, n_i, N_i)\}$. Upon observing a sequence of k uniformly randomly picked queries $\mathbf{q}_1, \ldots, \mathbf{q}_k$, the goal of the adversary is to guess the keyword of one of the queries of his choice, that is, his guess takes the form (j, kw). We use gain function g_1 defined in Section 8.4.3 to measure the success of the adversary.

Consider the following likelihood-based adversary $\mathcal{A}$ who, picks at random an observed query $\mathbf{q}_i$, computes the likelihood of $KW(\mathbf{q}_i)$ being every kw_j, and uses the keyword kw_j that has the largest likelihood as his guess.

The likelihood of $KW(\mathbf{q}_i)$ being kw_j can be expressed as

$$\mathsf{Pr}\left[\hat{l}(KW(\mathbf{q}_i)) \mid n_j, N_j\right] = \mathsf{Pr}\left[\hat{l}(KW(\mathbf{q}_i)) = n_j + N_j + n^* + \mathbf{Lap}(2/\epsilon)\right].$$

Define event $\omega_j(l)$ as the event that keyword kw_j produces the largest likelihood given observed query response volume l. The probability that adversary $\mathcal{A}$ guesses the keyword

kw_j associated to query $\mathbf{q}$ correctly can be expressed as:

$$\sum_{l=1}^{\infty} \mathsf{Pr}\left[\hat{l}(KW(\mathbf{q})) = l\right] \mathbb{1}(\omega_j(l)) \tag{8.3}$$

$$= \sum_{l=1}^{\infty} \mathsf{Pr}\left[\hat{l}(kw_j)) = l\right] \mathbb{1}(\omega_j(l)). \tag{8.4}$$

As the queries are uniformly randomly distributed, the expected gain can be bounded as

$$\sup_{\mathcal{A}} \mathbf{E}\left[\mathbf{Real}^{g_4}_{\mathtt{DPVH},\mathcal{A},\mathcal{Z}}(n)\right] \geq \frac{1}{|KW(\mathbf{DB})|} \sum_{j} \sum_{l=1}^{\infty} \mathsf{Pr}\left[\hat{l}(kw_j)) = l\right] \mathbb{1}(\omega_j(l)) + \mathtt{negl}(\lambda). \tag{8.5}$$

Concretely, for a database where the true query response volumes $l(kw_i)$ are far apart and the noise $\mathbf{Lap}(2/\epsilon)$ is small, it is easy to see that Equation 8.4 is large. For example, if $l(kw_i) = 100^i$ and $\epsilon = 0.2$, it is easy to check that $\mathbb{1}(\omega_j(l)) >> 0.99$ when l is close to the expectation of $\hat{l}(kw_j))$, and the whole summation is close to 1. This means that the likelihood-based adversary described above has a high chance of guessing the keyword associated to a query correctly.

8.6 Discussion

In this chapter, we propose a new security notion which measures security in a relative way. It allows for flexible auxiliary information and specific attack goals. We propose three new constructions and demonstrate the merits of using our security notion over the standard notion.

Although presented as a passive security notion where the adversary has no control over the generation of the data and the queries, our notion can be extended to an active setting easily by letting the adversary influence the generating of the data and the queries in some way. It is also straightforward to extend our notion to an adaptive setting where the adversary is allowed to pick queries one at a time. It is also possible to change what the adversary receives to model other adversaries, such as a man-in-the-middle adversary.

We have only showed a few gain functions as examples and we believe that there are many more interesting ones. For instance, we can have a gain function which targets keyword recovery of documents, where each keyword is weighted differently. We can also have a gain function which targets query recovery, where some queries are more valuable than the others. Our idea can also be applied to other cryptosystems where there is leakage. It may also allow for development of weaker but more efficient cryptographic primitives to be used in cryptosystems.

Chapter 9

Conclusion and Discussion

This thesis presents new results on cryptanalysis, constructions and security notions for structured encryption. In this chapter, we summarise these results, and suggest new research directions to look at.

Contents

9.1 Cryptanalysis

In Chapters 5 and 6, we have presented new attacks on encrypted range queries and searchable encryption respectively. These attacks aim to be as practical as possible. For example, our attacks on encrypted range queries in Section 5.3.2 assume that only range queries from small windows are made and some form of countermeasure is used by the client. Still, we manage to show that attacks are possible in these scenarios.

In addition, in Chapter 6, we identify a new class of leakage we called *system-level* leakage. This does not only lead to devastating attacks, but also motivates system-wide secure structured encryption such as the ones we devise in Chapter 7.

On the other hand, the attacks we have explored are just the tip of the iceberg. There are many research challenges that are still open. These include the need for a better understanding of distribution of databases and queries, a more methodical way of picking auxiliary information, and a more refined way of measuring the success of the attacker – all of which have strong implications on design of structured encryption schemes.

Furthermore, many of our attacks and other attacks in the literature [94, 115, 87, 111, 17] do not have an optimality proof, meaning that there can possibly be better attacks given the same leakage and auxiliary information. Of course, this does not invalidate the attacks, but it makes them inappropriate as cryptanalysis to show security of a construction – a scheme that is secure against one of these attacks may still be vulnerable to an attack in the future. In that light, we need to better our understanding in designing of attacks, and search for optimal attacks whenever possible.

9.2 Constructions

In Chapter 7, we present the first constructions of searchable encryption that are system-wide secure. We show security of our constructions through the cryptanalysis techniques we have developed in Chapter 6. We demonstrate practical efficiency of our construction through experiments on a real-world dataset using a Java implementation [175] which uses Redis [1] as the underlying database system.

It is worth noting that our constructions are secure against all known attacks, including query reconstruction attacks [94, 33, 155, 17, 143] exploiting various leakages and file-injection attacks [200, 17].

However, our construction is only the first step towards designing a full-fledged structured encryption scheme. The schemes we have proposed have clear limitations, including a hard upper bound on the number of possible insertions, a relatively slow insertion/deletion operation, and an oversized search index[1].

Furthermore, our constructions only support simple keyword searches and updates. More research is needed to add more functionality to it. On a related note, all structured encryption schemes in the literature support a particular set of queries, and the apparent way to expand on query expressiveness is to "merge" different schemes. This

[1]The search index has size comparable to the actual database as document identifier duplication is used. This is a common problem for almost all searchable encryption constructions and there is no known solutions for this problem.

can be problematic as the combined leakage from different schemes may lead to new attack surfaces. In that light, it is interesting to look into leakage cryptanalysis of this kind and the possibility of designing "composable" schemes.

In the bigger picture, SWiSSSE can be considered as a particular information retrieval scheme with non-trivial leakage. In Section 7.8, we show that SWiSSSE is an order of magnitude slower than an insecure plaintext solution. This however, means that it is a few orders of magnitudes faster than the state-of-the-art zero-knowledge ORAM/PIR-based solutions. This opens interesting research directions in non-zero-knowledge information retrieval schemes and understanding the trade-off between efficiency and security.

9.3 Foundation

In Chapter 8, we point out shortcomings in the standard security definition for structured encryption [53, 40] and develop a new security notion which aims to capture leakage-abuse attacks. We show how our proposed notion can be used to prove (in)security of schemes with respect to classes of leakage-abuse attacks.

There are two main challenges in the use of our security notion. Firstly, a proof of security in our notion is significantly harder than that in the standard notion. It is easy to prove the leakage profile of a scheme, but a proof of security in our notion may require development of new information-theoretic tools. We believe that more research efforts should be put into this direction.

Secondly, we need to further our understanding of the uses of databases and the adversarial power of an attacker in the real world, so as to fully utilise our security notion. In particular, current models of distribution of database and adversarial power of an attacker are mainly synthetic, so we are not able to prove practical security of a scheme in our notion. If we were able to do so, we can potentially design much more efficient schemes as these schemes only need to be secure under a specific set of settings. We believe that more research in leakage cryptanalysis can help us in this direction.

9.4 Leakage vs Efficiency in Related Fields

Our constructions presented in Chapter 7 and security notion presented in Chapter 8 are efforts towards building more efficient schemes than the perfectly secure ones. This is the first work of this kind in the field and we wish more research can be done in this direction.

Our work falls under a broad class of works [197, 178, 185, 38] that focus on building efficient schemes with non-trivial leakage. We hope that the techniques we used in our work can be generalised to these related fields too.

Bibliography

[1] Redis. `https://redis.io/`. Accessed: 2020-10-15. [Cited on pages 181 and 216.]

[2] Hime Aguiar e Oliveira Junior, Lester Ingber, Antonio Petraglia, Mariane Rembold Petraglia, and Maria Augusta Soares Machado. *Adaptive Simulated Annealing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. [Cited on pages 117, 124, and 125.]

[3] Adi Akavia, Dan Feldman, and Hayim Shaul. Secure search via multi-ring fully homomorphic encryption. Cryptology ePrint Archive, Report 2018/245, 2018. `https://eprint.iacr.org/2018/245`. [Cited on page 42.]

[4] M. S. Alvim, K. Chatzikokolakis, C. Palamidessi, and G. Smith. Measuring information leakage using generalized gain functions. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 265–279, June 2012. [Cited on page 213.]

[5] Mário S. Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring Information Leakage using Generalized Gain Functions. In *Computer Security Foundations*, pages 265–279, Cambridge MA, United States, 2012. IEEE. [Cited on pages 187, 191, and 213.]

[6] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. [Cited on pages 13, 109, 110, 112, and 113.]

[7] Sanjeev Arora and Boaz Barak. *Computational complexity. A modern approach*. Cambridge: Cambridge University Press, 2009. [Cited on page 28.]

[8] R. Arratia, L. Goldstein, and L. Gordon. Two moments suffice for poisson approximations: The chen-stein method. *Ann. Probab.*, 17(1):9–25, 01 1989. [Cited on page 76.]

[9] Dmitri Asonov and Johann Christoph Freytag. Almost optimal private information retrieval. In Roger Dingledine and Paul F. Syverson, editors, *PET 2002: 2nd International Workshop on Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 209–223, San Francisco, CA, USA, April 14–15, 2002. Springer, Heidelberg, Germany. [Cited on page 42.]

[10] G. Ausiello. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer, 2020. [Cited on page 211.]

[11] L Babai. Trading group theory for randomness. *Proceedings of the seventeenth annual ACM symposium on Theory of computing - STOC 85*, 1985. [Cited on page 28.]

[12] Michael Backes, Rainer W. Gerling, Sebastian Gerling, Stefan Nürnberger, Dominique Schröder, and Mark Simkin. WebTrust - A comprehensive authenticity and integrity framework for HTTP. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS 14: 12th International Conference on Applied Cryptography and Network Security*, volume 8479 of *Lecture Notes in Computer Science*, pages 401–418, Lausanne, Switzerland, June 10–13, 2014. Springer, Heidelberg, Germany. [Cited on page 13.]

[13] Normand J. Beaudry and Renato Renner. An intuitive proof of the data processing inequality. *Quantum Info. Comput.*, 12(5-6):432–441, May 2012. [Cited on page 213.]

[14] Oliver Berthold and Heinrich Langos. Dummy traffic against long term intersection attacks. In Roger Dingledine and Paul F. Syverson, editors, *PET 2002: 2nd International Workshop on Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 110–128, San Francisco, CA, USA, April 14–15, 2002. Springer, Heidelberg, Germany. [Cited on page 106.]

[15] Patrick Billingsley. *Probability and Measure*. John Wiley and Sons, second edition, 1986. [Cited on page 24.]

[16] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. The tao of inference in privacy-protected databases. *Proc. VLDB Endow.*, 11(11):1715–1728, July 2018. [Cited on page 139.]

[17] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. In *ISOC Network and Distributed System Security Symposium – NDSS 2020*, San Diego, CA, USA, February 23–26, 2020. The Internet Society. [Cited on pages 13, 14, 15, 54, 111, 115, 116, 117, 131, 132, 139, 189, 207, 208, and 216.]

[18] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970. [Cited on page 51.]

[19] Alexandra Boldyreva and Nathan Chenette. Efficient fuzzy search on encrypted data. In Carlos Cid and Christian Rechberger, editors, *Fast Software Encryption – FSE 2014*, volume 8540 of *Lecture Notes in Computer Science*, pages 613–633, London, UK, March 3–5, 2015. Springer, Heidelberg, Germany. [Cited on pages 12 and 52.]

[20] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 224–241, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany. [Cited on pages 13 and 59.]

[21] Alexandra Boldyreva, Nathan Chenette, and Adam O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 578–595, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. [Cited on pages 59 and 190.]

[22] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany. [Cited on page 12.]

[23] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. Private database queries using somewhat homomorphic encryption. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13: 11th International Conference on Applied Cryptography and Network Security*, volume 7954 of *Lecture Notes in Computer Science*, pages 102–118, Banff, AB, Canada, June 25–28, 2013. Springer, Heidelberg, Germany. [Cited on page 13.]

[24] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. `http://toc.cryptobook.us/`. Accessed: 2021-01-10. [Cited on page 31.]

[25] Raphael Bost. $\Sigma o\phi o\varsigma$: Forward secure searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1143–1154, Vienna, Austria, October 24–28, 2016. ACM Press. [Cited on pages 12, 51, and 177.]

[26] Raphael Bost and Pierre-Alain Fouque. Thwarting leakage abuse attacks against searchable encryption – A formal approach and applications to database padding. Cryptology ePrint Archive, Report 2017/1060, 2017. `https://eprint.iacr.org/2017/1060`. [Cited on pages 15, 146, and 190.]

[27] Raphael Bost and Pierre-Alain Fouque. Security-efficiency tradeoffs in searchable encryption. *Proceedings on Privacy Enhancing Technologies*, 2019(4):132–151, October 2019. [Cited on page 56.]

[28] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. Cryptology ePrint Archive, Report 2016/062, 2016. `https://eprint.iacr.org/2016/062`. [Cited on page 177.]

[29] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1465–1482, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. [Cited on pages 12, 51, 54, 55, and 177.]

[30] Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 62–89, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. [Cited on page 13.]

[31] Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. Range query integrity in cloud data streams with efficient insertion. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 719–724, Milan, Italy, November 14–16, 2016. Springer, Heidelberg, Germany. [Cited on pages 12 and 52.]

[32] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 694–726, Baltimore, MD, USA, November 12–15, 2017. Springer, Heidelberg, Germany. [Cited on page 42.]

[33] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 668–679, Denver, CO, USA, October 12–16, 2015. ACM Press. [Cited on pages 13, 14, 15, 111, 115, 117, 131, 132, 139, 146, 160, 177, 178, 207, 208, and 216.]

[34] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. Cryptology ePrint Archive, Report 2016/718, 2016. `https://eprint.iacr.org/2016/718`. [Cited on page 115.]

[35] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *ISOC Network and Distributed System Security Symposium – NDSS 2014*, San Diego, CA, USA, February 23–26, 2014. The Internet Society. [Cited on pages 12, 51, and 146.]

[36] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. [Cited on pages 12, 51, 52, 55, 146, and 156.]

[37] Javad Ghareh Chamani, Dimitrios Papadopoulos, Charalampos Papamanthou, and Rasool Jalili. New constructions for forward and backward private symmetric searchable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1038–1055, Toronto, ON, Canada, October 15–19, 2018. ACM Press. [Cited on page 177.]

[38] T.-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2448–2467, San Diego, CA, USA, January 6–9, 2019. ACM-SIAM. [Cited on pages 16, 186, and 217.]

[39] Yan-Cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05: 3rd International Conference on Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455, New York, NY, USA, June 7–10, 2005. Springer, Heidelberg, Germany. [Cited on pages 12, 51, and 177.]

[40] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. [Cited on pages 12, 13, 48, 51, 52, 187, 189, and 217.]

[41] G. Chen, T. Lai, M. K. Reiter, and Y. Zhang. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 810–818, 2018. [Cited on pages 12, 15, 116, 117, 119, 121, 123, 126, 127, 129, 130, 132, 133, 134, 136, 137, 138, and 139.]

[42] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. Logistic regression over encrypted data from fully

homomorphic encryption. Cryptology ePrint Archive, Report 2018/462, 2018. https://eprint.iacr.org/2018/462. [Cited on page 42.]

[43] Louis H. Y. Chen. An approximation theorem for sums of certain randomly selected indicators. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 33(1):69–74, Mar 1975. [Cited on page 76.]

[44] Louis H. Y. Chen. Poisson approximation for dependent trials. *Ann. Probab.*, 3(3):534–545, 06 1975. [Cited on page 76.]

[45] Rongmao Chen, Yi Mu, Guomin Yang, Fuchun Guo, and Xiaofen Wang. A new general framework for secure public key encryption with keyword search. In Ernest Foo and Douglas Stebila, editors, *ACISP 15: 20th Australasian Conference on Information Security and Privacy*, volume 9144 of *Lecture Notes in Computer Science*, pages 59–76, Brisbane, QLD, Australia, June 29 – July 1, 2015. Springer, Heidelberg, Germany. [Cited on page 12.]

[46] Nathan Chenette, Kevin Lewi, Stephen A. Weis, and David J. Wu. Practical order-revealing encryption with limited leakage. In Thomas Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 474–493, Bochum, Germany, March 20–23, 2016. Springer, Heidelberg, Germany. [Cited on pages 13 and 59.]

[47] Xiang Cheng, Sen Su, Yiping Teng, and Ke Xiao. Enabling secure and efficient knn query processing over encrypted spatial data in the cloud. *Sec. and Commun. Netw.*, 8(17):3205–3218, November 2015. [Cited on pages 13 and 52.]

[48] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020. [Cited on page 42.]

[49] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press. [Cited on pages 13 and 42.]

[50] Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure ORAM with $\tilde{O}(\log^2 n)$ overhead. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 62–81, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. [Cited on page 146.]

[51] Thomas H. Cormen, Charles Eric. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. [Cited on page 19.]

[52] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 44–75, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. [Cited on page 13.]

[53] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 79–88, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. [Cited on pages 12, 48, 51, 54, 55, 146, 156, 187, 189, 190, 191, and 217.]

[54] Wladimir De la Cadena, Asya Mitseva, Jan Pennekamp, Jens Hiller, Fabian Lanze, Thomas Engel, Klaus Wehrle, and Andriy Panchenko. POSTER: Traffic splitting to counter website fingerprinting. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2533–2535, London, UK, November 11–15, 2019. ACM Press. [Cited on page 106.]

[55] Juan de Vicente, Juan Lanchares, and Román Hermida. Placement by thermodynamic simulated annealing. *Physics Letters A*, 317(5):415–423, 2003. [Cited on page 126.]

[56] Ioannis Demertzis, Javad Ghareh Chamani, Dimitrios Papadopoulos, and Charalampos Papamanthou. Dynamic searchable encryption with small client storage. In *ISOC Network and Distributed System Security Symposium – NDSS 2020*, San Diego, CA, USA, February 23–26, 2020. The Internet Society. [Cited on pages 51 and 55.]

[57] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. SEAL: Attack mitigation for encrypted databases via adjustable leakage. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020: 29th USENIX Security Symposium*, pages 2433–2450. USENIX Association, August 12–14, 2020. [Cited on pages 12, 15, 52, 60, 61, 96, 97, 100, 104, 116, 146, 187, and 189.]

[58] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 185–198, New York, NY, USA, 2016. Association for Computing Machinery. [Cited on pages 12, 52, 59, 60, 61, and 96.]

[59] Alexander W. Dent. A note on game-hopping proofs. Cryptology ePrint Archive, Report 2006/260, 2006. `https://eprint.iacr.org/2006/260`. [Cited on page 37.]

[60] Jeff Desjardins. How much data is generated each day? `https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/`. Accessed: 2020-11-19. [Cited on page 10.]

[61] Srinivas Devadas, Marten van Dijk, Christopher W. Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion ORAM: A constant bandwidth blowup oblivious RAM. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 145–174, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany. [Cited on page 14.]

[62] Whitfield Diffie and Martin E. Hellman. New directions in cryptography, 1976. [Cited on page 32.]

[63] Alexandros G. Dimakis, Brighten Godfrey, Martin J. Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *CoRR*, abs/cs/0702015, 2007. [Cited on page 121.]

[64] William F. Ehrsam, Carl H. W. Meyer, John L. Smith, and Walter L. Tuchman. Message verification and transmission error detection by block chaining, 1976. [Cited on pages 39 and 40.]

[65] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31:469–472, 1985. [Cited on page 42.]

[66] Y. Elmehdwi, B. K. Samanthula, and W. Jiang. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *2014 IEEE 30th International Conference on Data Engineering*, pages 664–675, 2014. [Cited on page 52.]

[67] Saba Eskandarian and Matei Zaharia. Oblidb: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, October 2019. [Cited on page 42.]

[68] Mohammad Etemad, Alptekin Küpçü, Charalampos Papamanthou, and David Evans. Efficient dynamic searchable encryption with forward privacy. *PoPETs*, 2018(1):5–20, 2018. [Cited on page 177.]

[69] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel-Catalin Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *ESORICS 2015: 20th European Symposium on Research in Computer Security, Part II*, volume 9327 of *Lecture Notes in Computer Science*, pages 123–145, Vienna, Austria, September 21–25, 2015. Springer, Heidelberg, Germany. [Cited on pages 52 and 59.]

[70] Niels Ferguson and Bruce Schneier. *Practical Cryptography*. John Wiley & Sons, Inc., USA, 1 edition, 2003. [Cited on page 60.]

[71] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992. [Cited on page 67.]

[72] Xinwen Fu, B. Graham, Riccardo Bettati, and Wei Zhao. Active traffic analysis attacks and countermeasures. pages 31– 39, 11 2003. [Cited on pages 60, 106, and 146.]

[73] M. R. Garey and D. S. Johnson. *Computers And Intractability A Guide To The Theory Of Np-Completeness*. Freeman & Company, New York, 1979. [Cited on page 28.]

[74] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. TWORAM: Efficient oblivious RAM in two rounds with applications to searchable encryption. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 563–592, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. [Cited on pages 14, 146, 177, and 200.]

[75] Romain Gay, Pierrick Méaux, and Hoeteck Wee. Predicate encryption for multi-dimensional range queries from lattices. In Jonathan Katz, editor, *PKC 2015: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 752–776, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany. [Cited on page 59.]

[76] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. [Cited on pages 13 and 42.]

[77] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005: 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815, Lisbon, Portugal, July 11–15, 2005. Springer, Heidelberg, Germany. [Cited on page 42.]

[78] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. [Cited on page 13.]

[79] Marilyn George, Seny Kamara, and Tarik Moataz. Structured encryption and dynamic leakage suppression. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 370–396, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. [Cited on page 113.]

[80] John T. Gill. Computational complexity of probabilistic turing machines. *Proceedings of the sixth annual ACM symposium on Theory of computing - STOC 74*, 1974. [Cited on page 28.]

[81] Christian Göge, Tim Waage, Daniel Homann, and Lena Wiese. Improving fuzzy searchable encryption with direct bigram embedding. In Javier Lopez, Simone Fischer-Hübner, and Costas Lambrinoudakis, editors, *TrustBus 2017: Trust and Privacy in Digital Business, 14th International Conference*, volume 10442 of *Lecture Notes in Computer Science*, pages 115–129, Lyon, France, August 30–31, 2017. Springer, Heidelberg, Germany. [Cited on page 12.]

[82] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. `https://eprint.iacr.org/2003/216`. [Cited on pages 12 and 51.]

[83] Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA, 2006. [Cited on page 31.]

[84] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996. [Cited on pages 14, 42, and 146.]

[85] Brian Gough. *GNU Scientific Library Reference Manual - Third Edition*. Network Theory Ltd., 3rd edition, 2009. [Cited on page 132.]

[86] S. L. Graham, R. L. Rivest, and Ralph C. Merkle. Secure communications over insecure channels, 1978. [Cited on page 32.]

[87] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 315–331, Toronto, ON, Canada, October 15–19, 2018. ACM Press. [Cited on pages 13, 54, 60, 61, 71, 189, and 216.]

[88] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy*, pages 1067–1083, San

Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press. [Cited on pages 13, 54, 60, 62, 64, 86, and 97.]

[89] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 361–378, London, UK, November 11–15, 2019. ACM Press. [Cited on pages 13, 16, and 189.]

[90] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Leakage perturbation is not enough: Breaking structured encryption using simulated annealing. Cryptology ePrint Archive, Report 2021/879, 2021. `https://ia.cr/2021/879`. [Cited on page 16.]

[91] Zichen Gui, Kenneth G. Paterson, Sikhar Patranabis, and Bogdan Warinschi. Swissse: System-wide security for searchable symmetric encryption. Cryptology ePrint Archive, Report 2020/1328, 2020. `https://ia.cr/2020/1328`. [Cited on page 16.]

[92] Warren He, Devdatta Akhawe, Sumeet Jain, Elaine Shi, and Dawn Xiaodong Song. ShadowCrypt: Encrypted web applications for everyone. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 1028–1039, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. [Cited on pages 115 and 117.]

[93] William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *Journal of Computer and System Sciences*, 65(4):695–716, 2002. Special Issue on Complexity 2001. [Cited on page 28.]

[94] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *ISOC Network and Distributed System Security Symposium – NDSS 2012*, San Diego, CA, USA, February 5–8, 2012. The Internet Society. [Cited on pages 13, 14, 15, 54, 111, 115, 117, 131, 132, 139, 160, 178, 189, 207, 208, 211, 213, and 216.]

[95] Malika Izabachène, Renaud Sirdey, and Martin Zuber. Practical fully homomorphic encryption for fully masked neural networks. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *CANS 19: 18th International Conference on Cryptology and Network Security*, volume 11829 of *Lecture Notes in Computer Science*, pages 24–36, Fuzhou, China, October 25–27, 2019. Springer, Heidelberg, Germany. [Cited on page 42.]

[96] Rothe Jorg. *Complexity Theory and Cryptology: An Introduction to Cryptocomplexity*. 2010. [Cited on pages 19 and 25.]

[97] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 94–124, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. [Cited on pages 12, 51, 52, 55, and 146.]

[98] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part II*, volume 11477 of *Lecture Notes in Computer Science*,

pages 183–213, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. [Cited on pages 12, 16, 109, 112, 113, 114, 116, 117, 119, 120, 121, 122, 123, 126, 127, 128, 130, 132, 133, 134, 136, 137, 138, 146, 184, 187, 189, and 193.]

[99] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 339–370, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. [Cited on page 15.]

[100] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 965–976, Raleigh, NC, USA, October 16–18, 2012. ACM Press. [Cited on pages 12, 51, 54, and 55.]

[101] Viggo Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology, Dept. of Numerical Analysis and Computing Science, 1992. [Cited on page 211.]

[102] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1329–1340, Vienna, Austria, October 24–28, 2016. ACM Press. [Cited on pages 54, 60, 61, 62, 64, 71, 72, 97, 189, and 190.]

[103] Florian Kerschbaum. Frequency-hiding order-preserving encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 656–667, Denver, CO, USA, October 12–16, 2015. ACM Press. [Cited on pages 13 and 59.]

[104] Florian Kerschbaum and Anselme Tueno. An efficiently searchable encrypted data structure for range queries. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019: 24th European Symposium on Research in Computer Security, Part II*, volume 11736 of *Lecture Notes in Computer Science*, pages 344–364, Luxembourg, September 23–27, 2019. Springer, Heidelberg, Germany. [Cited on pages 189 and 190.]

[105] Aggelos Kiayias, Nikos Leonardos, Helger Lipmaa, Kateryna Pavlyk, and Qiang Tang. Optimal rate private information retrieval from homomorphic encryption. *Proceedings on Privacy Enhancing Technologies*, 2015(2):222 – 243, 01 Jun. 2015. [Cited on page 42.]

[106] Eunkyung Kim, Hyang-Sook Lee, and Jeongeun Park. Towards round-optimal secure multiparty computations: Multikey FHE without a CRS. In Willy Susilo and Guomin Yang, editors, *ACISP 18: 23rd Australasian Conference on Information Security and Privacy*, volume 10946 of *Lecture Notes in Computer Science*, pages 101–113, Wollongong, NSW, Australia, July 11–13, 2018. Springer, Heidelberg, Germany. [Cited on page 13.]

[107] Eunkyung Kim and Mehdi Tibouchi. FHE over the integers and modular arithmetic circuits. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 435–450, Milan, Italy, November 14–16, 2016. Springer, Heidelberg, Germany. [Cited on page 13.]

[108] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. A secure knn query processing algorithm using homomorphic encryption on outsourced database. *Data & Knowledge Engineering*, 123:101602, 2019. [Cited on page 13.]

[109] Kee Sung Kim, Minkyu Kim, Dongsoo Lee, Je Hong Park, and Woo-Hwan Kim. Forward secure dynamic searchable symmetric encryption with efficient updates. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1449–1463, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. [Cited on pages 12 and 177.]

[110] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987. [Cited on page 42.]

[111] Evgenios M. Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. The state of the uniform: Attacks on encrypted databases beyond the uniform query distribution. In *2020 IEEE Symposium on Security and Privacy*, pages 1223–1240, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press. [Cited on pages 13, 60, 61, and 216.]

[112] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104 (Informational), February 1997. Updated by RFC 6151. [Cited on page 181.]

[113] Kaoru Kurosawa and Yasuhiro Ohtaki. UC-secure searchable symmetric encryption. In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 285–298, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer, Heidelberg, Germany. [Cited on pages 12 and 51.]

[114] Kaoru Kurosawa and Yasuhiro Ohtaki. How to update documents verifiably in searchable symmetric encryption. In Michel Abdalla, Cristina Nita-Rotaru, and Ricardo Dahab, editors, *CANS 13: 12th International Conference on Cryptology and Network Security*, volume 8257 of *Lecture Notes in Computer Science*, pages 309–328, Paraty, Brazil, November 20–22, 2013. Springer, Heidelberg, Germany. [Cited on page 51.]

[115] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy*, pages 297–314, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. [Cited on pages 13, 54, 60, 62, 64, 97, 189, and 216.]

[116] Shangqi Lai, Sikhar Patranabis, Amin Sakzad, Joseph K. Liu, Debdeep Mukhopadhyay, Ron Steinfeld, Shifeng Sun, Dongxi Liu, and Cong Zuo. Result pattern hiding searchable encryption for conjunctive queries. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 745–762, Toronto, ON, Canada, October 15–19, 2018. ACM Press. [Cited on pages 12, 51, 52, 55, and 146.]

[117] Shangqi Lai, Xingliang Yuan, Shifeng Sun, Joseph K. Liu, Yuhong Liu, and Dongxi Liu. GraphSE^2: An encrypted graph database for privacy-preserving social search. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin

Kirda, and Zhenkai Liang, editors, *ASIACCS 19: 14th ACM Symposium on Information, Computer and Communications Security*, pages 41–54, Auckland, New Zealand, July 9–12, 2019. ACM Press. [Cited on page 13.]

[118] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious RAM lower bound! In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 523–542, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. [Cited on page 42.]

[119] Billy Lau, Simon P. Chung, Chengyu Song, Yeongjin Jang, Wenke Lee, and Alexandra Boldyreva. Mimesis aegis: A mimicry privacy shield-A system's approach to data privacy on public cloud. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 33–48, San Diego, CA, USA, August 20–22, 2014. USENIX Association. [Cited on pages 115 and 117.]

[120] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1167–1178, Vienna, Austria, October 24–28, 2016. ACM Press. [Cited on page 59.]

[121] Yuan Li, Hongbing Wang, and Yunlei Zhao. Delegatable order-revealing encryption. In Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang, editors, *ASIACCS 19: 14th ACM Symposium on Information, Computer and Communications Security*, pages 134–147, Auckland, New Zealand, July 9–12, 2019. ACM Press. [Cited on page 59.]

[122] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu an Tan. Search pattern leakage in searchable encryption: Attacks and new construction. Cryptology ePrint Archive, Report 2013/163, 2013. `https://eprint.iacr.org/2013/163`. [Cited on pages 54, 189, and 213.]

[123] Steve Lu and Rafail Ostrovsky. Distributed oblivious RAM for secure two-party computation. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 377–396, Tokyo, Japan, March 3–6, 2013. Springer, Heidelberg, Germany. [Cited on page 14.]

[124] Wenjie Lu, Shohei Kawasaki, and Jun Sakuma. Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data. In *ISOC Network and Distributed System Security Symposium – NDSS 2017*, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society. [Cited on page 42.]

[125] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 2–10 vol.1, 1990. [Cited on page 28.]

[126] Evangelia Anna Markatou and Roberto Tamassia. Full database reconstruction with access and search pattern leakage. In Zhiqiang Lin, Charalampos Papamanthou, and Michalis Polychronakis, editors, *ISC 2019: 22nd International Conference on Information Security*, volume 11723 of *Lecture Notes in Computer Science*, pages 25–43, New York City, NY, USA, September 16–18, 2019. Springer, Heidelberg, Germany. [Cited on pages 54 and 189.]

[127] David A. McGrew and John Viega. The security and performance of the galois/counter mode of operation (full version). Cryptology ePrint Archive, Report 2004/193, 2004. https://eprint.iacr.org/2004/193. [Cited on page 181.]

[128] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. GRECS: Graph encryption for approximate shortest distance queries. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 504–517, Denver, CO, USA, October 12–16, 2015. ACM Press. [Cited on page 52.]

[129] Microsoft. Driving growth together: Small businesses and the cloud. https://info.microsoft.com/rs/microsoftdemandcenter/images/driving-growth-together-small-businesses-and-cloud-infographic.pdf. Accessed: 2020-11-23. [Cited on page 10.]

[130] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426, Santa Barbara, CA, USA, August 18–22, 1986. Springer, Heidelberg, Germany. [Cited on page 42.]

[131] Kimberly Mlitz. Revenue from big data and business analytics worldwide from 2015 to 2022. https://www.statista.com/statistics/551501/worldwide-big-data-business-analytics-revenue/#:~:text=The%20global%20big%20data%20and,(CAGR)%20of%2013.2%20percent. Accessed: 2020-11-23. [Cited on page 10.]

[132] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications, 1993. [Cited on page 36.]

[133] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2053–2069, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press. [Cited on page 60.]

[134] Muhammad Naveed. The fallacy of composition of oblivious RAM and searchable encryption. Cryptology ePrint Archive, Report 2015/668, 2015. https://eprint.iacr.org/2015/668. [Cited on pages 55 and 56.]

[135] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 644–655, Denver, CO, USA, October 12–16, 2015. ACM Press. [Cited on pages 13 and 16.]

[136] Jerzy Neyman. Outline of a theory of statistical estimation based on the classical theory of probability. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 236(767):333–380, 1937. [Cited on page 25.]

[137] NIST. Block cipher modes. https://csrc.nist.gov/Projects/block-cipher-techniques/BCM. Accessed: 2021-02-06. [Cited on pages 38 and 39.]

[138] National Institute of Standards and Technology Gaithersburg MD. Specification for the advanced encryption standard (aes). Federal Information Processin Standards Publication 197, 2001. [Cited on page 181.]

[139] Sota Onozawa, Noboru Kunihiro, Masayuki Yoshino, and Ken Naganuma. Inference attacks on encrypted databases based on order preserving assignment problem. In Atsuo Inomata and Kan Yasuda, editors, *IWSEC 18: 13th International Workshop on Security, Advances in Information and Computer Security*, volume 11049 of *Lecture Notes in Computer Science*, pages 35–47, Sendai, Japan, September 3–5, 2018. Springer, Heidelberg, Germany. [Cited on page 13.]

[140] OpenMP Architecture Review Board. OpenMP application program interface version 5.0, 2018. [Cited on page 132.]

[141] Oracle. Java cryptography architecture (jca) reference guide, 4 2020. [Cited on page 181.]

[142] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption, 2020. [Cited on page 60.]

[143] Simon Oya and Florian Kerschbaum. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *USENIX Security 2021 (To Appear)*, 2021. [Cited on pages 13, 16, 54, 116, 117, 139, 189, and 216.]

[144] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. Do dummies pay off? Limits of dummy traffic protection in anonymous communications. In Emiliano De Cristofaro and Steven J. Murdoch, editors, *PETS 2014: 14th International Symposium on Privacy Enhancing Technologies*, volume 8555 of *Lecture Notes in Computer Science*, pages 204–223, Amsterdam, The Netherlands, July 16–18, 2014. Springer, Heidelberg, Germany. [Cited on page 60.]

[145] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms — ESA 2001*, pages 121–133, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. [Cited on pages 110 and 121.]

[146] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press. [Cited on pages 12 and 52.]

[147] Sarvar Patel, Giuseppe Persiano, Mariana Raykova, and Kevin Yeo. PanORAMa: Oblivious RAM with logarithmic overhead. In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 871–882, Paris, France, October 7–9, 2018. IEEE Computer Society Press. [Cited on page 146.]

[148] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Lower bounds for encrypted multi-maps and searchable encryption in the leakage cell probe model. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 433–463, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. [Cited on page 56.]

[149] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 79–93, London, UK, November 11–15, 2019. ACM Press. [Cited on pages 12, 15, 16, 109, 110, 113, 114, 116, 117, 119, 121, 123, 126, 127, 128, 129, 132, 133, 134, 135, 136, 137, 138, 146, 148, 184, 187, 189, and 213.]

[150] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. Arx: A strongly encrypted database system. Cryptology ePrint Archive, Report 2016/591, 2016. `https://eprint.iacr.org/2016/591`. [Cited on page 52.]

[151] Raluca A. Popa, Frank H. Li, and Nickolai Zeldovich. An ideal-security protocol for order-preserving encoding. In *2013 IEEE Symposium on Security and Privacy*, pages 463–477, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press. [Cited on page 13.]

[152] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, page 85–100, New York, NY, USA, 2011. Association for Computing Machinery. [Cited on pages 52 and 59.]

[153] M.f. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980. [Cited on pages 111 and 136.]

[154] Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50(5):284–317, 1944. [Cited on page 29.]

[155] David Pouliot and Charles V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1341–1352, Vienna, Austria, October 24–28, 2016. ACM Press. [Cited on pages 13, 14, 54, 115, 117, 189, and 216.]

[156] NLTK Project. *Natural Language Toolkit*. `https://www.nltk.org/`. [Cited on pages 111, 132, and 136.]

[157] Ronald L. Rivest and M. Dertouzos. On data banks and privacy homomorphisms. 1978. [Cited on page 13.]

[158] Daniel S. Roche, Daniel Apon, Seung Geol Choi, and Arkady Yerukhimovich. POPE: Partial order preserving encoding. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1131–1142, Vienna, Austria, October 24–28, 2016. ACM Press. [Cited on page 13.]

[159] Sheldon M. Ross. *A first course in probability*. Pearson Education Limited, 2020. [Cited on pages 19, 20, and 22.]

[160] Richard J. Rossi. *Mathematical statistics: an introduction to likelihood based inference*. Wiley, 2018. [Cited on page 19.]

[161] Jeff Schultz. How much data is created on the internet each day? `https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/`. Accessed: 2020-11-23. [Cited on page 10.]

[162] C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, 1949. [Cited on pages 32, 33, and 34.]

[163] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy*, pages 350–364, Oakland, CA, USA, May 20–23, 2007. IEEE Computer Society Press. [Cited on pages 12, 52, and 59.]

[164] Nigel P. Smart. *Cryptography Made Simple*. Springer Publishing Company, Incorporated, 1st edition, 2015. [Cited on page 31.]

[165] N. Smirnov. Table for Estimating the Goodness of Fit of Empirical Distributions. *The Annals of Mathematical Statistics*, 19(2):279 – 281, 1948. [Cited on page 135.]

[166] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55, Oakland, CA, USA, May 2000. IEEE Computer Society Press. [Cited on pages 11, 12, and 50.]

[167] Xiangfu Song, Changyu Dong, Dandan Yuan, Qiuliang Xu, and Minghao Zhao. Forward private searchable symmetric encryption with optimized I/O efficiency. Cryptology ePrint Archive, Report 2018/497, 2018. `https://eprint.iacr.org/2018/497`. [Cited on page 177.]

[168] Julian Stander and Bernard W. Silverman. Temperature schedules for simulated annealing. *Statistics and Computing*, 4(1):21–32, 1994. [Cited on page 126.]

[169] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *ISOC Network and Distributed System Security Symposium – NDSS 2014*, San Diego, CA, USA, February 23–26, 2014. The Internet Society. [Cited on pages 51 and 177.]

[170] Emil Stefanov and Elaine Shi. Multi-cloud oblivious storage. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 247–258, Berlin, Germany, November 4–8, 2013. ACM Press. [Cited on page 14.]

[171] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 299–310, Berlin, Germany, November 4–8, 2013. ACM Press. [Cited on pages 109, 110, 112, 113, and 146.]

[172] Charles Stein. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 2: Probability Theory*, pages 583–602, Berkeley, Calif., 1972. University of California Press. [Cited on page 76.]

[173] LLC StorageCraft Technology. 7 most infamous cloud security breaches. `https://blog.storagecraft.com/7-infamous-cloud-security-breaches/`. Accessed: 2020-11-23. [Cited on page 10.]

[174] Shifeng Sun, Xingliang Yuan, Joseph K. Liu, Ron Steinfeld, Amin Sakzad, Viet Vo, and Surya Nepal. Practical backward-secure searchable encryption from symmetric puncturable encryption. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 763–780, Toronto, ON, Canada, October 15–19, 2018. ACM Press. [Cited on page 177.]

[175] SWiSSE. *System-wide Security for Symmetric Searchable Encryption*, 2020. `https://github.com/SWiSSSE-crypto/SWiSSSE`. [Cited on pages 181 and 216.]

[176] Benjamin Hong Meng Tan, Hyung Tae Lee, Huaxiong Wang, Shu Qin Ren, and Khin Mi Mi Aung. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. Cryptology ePrint Archive, Report 2019/332, 2019. `https://eprint.iacr.org/2019/332`. [Cited on page 42.]

[177] Isamu Teranishi, Moti Yung, and Tal Malkin. Order-preserving encryption secure beyond one-wayness. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 42–61, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. [Cited on page 59.]

[178] Raphael R. Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ-private information retrieval. *Proceedings on Privacy Enhancing Technologies*, 2016(4):184–201, October 2016. [Cited on pages 16, 138, 186, and 217.]

[179] Jonathan T. Trostle and Andy Parrish. Efficient computationally private information retrieval from anonymity or trapdoor groups. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010: 13th International Conference on Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages 114–128, Boca Raton, FL, USA, October 25–28, 2011. Springer, Heidelberg, Germany. [Cited on page 42.]

[180] Alan Turing. Systems of logic based on ordinals (1938). *The Essential Turing*, 2004. [Cited on page 29.]

[181] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. [Cited on page 13.]

[182] Cédric Van Rompay, Refik Molva, and Melek Önen. Multi-user searchable encryption in the cloud. In Javier Lopez and Chris J. Mitchell, editors, *ISC 2015: 18th International Conference on Information Security*, volume 9290 of *Lecture Notes in Computer Science*, pages 299–316, Trondheim, Norway, September 9–11, 2015. Springer, Heidelberg, Germany. [Cited on page 52.]

[183] Cédric Van Rompay, Refik Molva, and Melek Önen. Fast two-server multi-user searchable encryption with strict access pattern leakage. In David Naccache, Shouhuai Xu, Sihan Qing, Pierangela Samarati, Gregory Blanc, Rongxing Lu, Zonghua Zhang, and Ahmed Meddahi, editors, *ICICS 18: 20th International Conference on Information and Communication Security*, volume 11149 of *Lecture Notes in Computer Science*, pages 393–408, Lille, France, October 29–31, 2018. Springer, Heidelberg, Germany. [Cited on page 52.]

[184] Heribert Vollmer. *Introduction to circuit complexity: a uniform approach*. Springer, 2011. [Cited on page 28.]

[185] Sameer Wagh, Paul Cuff, and Prateek Mittal. Differentially private oblivious RAM. *Proceedings on Privacy Enhancing Technologies*, 2018(4):64–84, October 2018. [Cited on pages 16, 138, 186, and 217.]

[186] Boyang Wang, Yantian Hou, Ming Li, Haitao Wang, and Hui Li. Maple: scalable multi-dimensional range search over encrypted cloud data with tree-based index. In Shiho Moriai, Trent Jaeger, and Kouichi Sakurai, editors, *ASIACCS 14: 9th*

ACM Symposium on Information, Computer and Communications Security, pages 111–122, Kyoto, Japan, June 3–6, 2014. ACM Press. [Cited on page 59.]

[187] Jiafan Wang and Sherman S. M. Chow. Forward and backward-secure range-searchable symmetric encryption. Cryptology ePrint Archive, Report 2019/497, 2019. `https://eprint.iacr.org/2019/497`. [Cited on pages 59, 61, 62, 63, and 64.]

[188] Lingyu Wang, Yingjiu Li, Duminda Wijesekera, and Sushil Jajodia. Precisely answering multi-dimensional range queries without privacy breaches. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS 2003: 8th European Symposium on Research in Computer Security*, volume 2808 of *Lecture Notes in Computer Science*, pages 100–115, Gjøvik, Norway, October 13–15, 2003. Springer, Heidelberg, Germany. [Cited on page 59.]

[189] Xiao Wang, T.-H. Hubert Chan, and Elaine Shi. Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 850–861, Denver, CO, USA, October 12–16, 2015. ACM Press. [Cited on page 146.]

[190] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T.-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 215–226, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press. [Cited on pages 14 and 42.]

[191] Yunling Wang, Jianfeng Wang, Shi-Feng Sun, Joseph K. Liu, Willy Susilo, and Xiaofeng Chen. Towards multi-user searchable encryption supporting Boolean query and fast decryption. In Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li, editors, *ProvSec 2017: 11th International Conference on Provable Security*, volume 10592 of *Lecture Notes in Computer Science*, pages 24–38, Xi'an, China, October 23–25, 2017. Springer, Heidelberg, Germany. [Cited on page 52.]

[192] John Watrous. Quantum computational complexity, 2008. [Cited on page 29.]

[193] WHO. Icd-11 for mortality and morbidity statistics. `https://icd.who.int/browse11/l-m/en`. Accessed: 2020-12-03. [Cited on page 59.]

[194] CMU William W. Cohen, MLD. Enron email dataset. [Cited on pages 111, 117, 131, 132, 163, and 181.]

[195] Peter Williams and Radu Sion. Single round access privacy on outsourced storage. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 293–304, Raleigh, NC, USA, October 16–18, 2012. ACM Press. [Cited on page 14.]

[196] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *ISOC Network and Distributed System Security Symposium – NDSS 2009*, San Diego, CA, USA, February 8–11, 2009. The Internet Society. [Cited on page 106.]

[197] Qiuyu Xiao, Michael K. Reiter, and Yinqian Zhang. Mitigating storage side channels using statistical privacy mechanisms. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 1582–1594, Denver, CO, USA, October 12–16, 2015. ACM Press. [Cited on pages 16, 146, 186, and 217.]

[198] L. Xu, X. Yuan, C. Wang, Q. Wang, and C. Xu. Hardening database padding for searchable encryption. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 2503–2511, 2019. [Cited on page 15.]

[199] Rui Zhang and Hideki Imai. Generic combination of public key encryption with keyword search and public key encryption. In Feng Bao, San Ling, Tatsuaki Okamoto, Huaxiong Wang, and Chaoping Xing, editors, *CANS 07: 6th International Conference on Cryptology and Network Security*, volume 4856 of *Lecture Notes in Computer Science*, pages 159–174, Singapore, December 8–10, 2007. Springer, Heidelberg, Germany. [Cited on page 12.]

[200] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 707–720, Austin, TX, USA, August 10–12, 2016. USENIX Association. [Cited on pages 51, 54, 116, 160, 167, 177, 178, 189, and 216.]

[201] L. Zhao, Q. Liu, H. Huang, and X. Jia. Efficient privacy-preserving query processing on outsourced geographic databases. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2018. [Cited on page 52.]

[202] Cong Zuo, Shi-Feng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. Dynamic searchable symmetric encryption with forward and stronger backward privacy. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *Computer Security – ESORICS 2019*, pages 283–303, Cham, 2019. Springer International Publishing. [Cited on page 12.]

[203] Cong Zuo, Shifeng Sun, Joseph K. Liu, Jun Shao, and Josef Pieprzyk. Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018: 23rd European Symposium on Research in Computer Security, Part II*, volume 11099 of *Lecture Notes in Computer Science*, pages 228–246, Barcelona, Spain, September 3–7, 2018. Springer, Heidelberg, Germany. [Cited on pages 59, 61, 62, 63, and 64.]