Practical yet Provably Secure: Complex Database Query Execution over Encrypted Data

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Florian Hahn
aus Baden

Tag der mündlichen Prüfung: 10.12.2018

1. Referent: Prof. Dr. Jörn Müller-Quade
2. Referent: Prof. Dr. Florian Kerschbaum

Abstract

Encryption schemes are designed to protect data and prevent access by unauthorized parties. With the rise of cloud computing and cloud storage (e.g. in form of databases-as-a-service) encryption schemes are vital for outsourcing sensitive data. However, common symmetric encryption schemes do not support computation over encrypted data rendering them incompatible with the cloud computing scenario. Thus, special encryption schemes are required for these scenarios supporting computation over encrypted data. Fully homomorphic encryption provides a theoretical solution for all possible scenarios while providing semantic security for the encrypted and outsourced data. However, decreasing the flexibility of the encryption schemes to be suitable for a specifically predefined use case and reducing the guaranteed security characteristics can result in better performance, e.g. in lower computational overhead and smaller ciphertext size.

For the use case of databases-as-a-service, CryptDB has been proposed in 2011 by Popa et al. The design of CryptDB is founded on property-preserving encryption. Particularly, such encryption schemes preserve specific properties of the plaintext even after the application of the encryption operation, e.g. the same plaintext always results in the same ciphertext. As one consequence of this approach, index structures provided by the database management system are compatible with such ciphertexts since these index structures are founded on the preserved properties. However, the preserved properties are also exploitable for a potential adversary as they are initially leaked for the whole encrypted data collection. Alternative approaches aim to increase the security of encrypted databases while still supporting query execution. This is achieved by unveiling information only if required. For instance, the information if an encrypted data record matches a specific database query is only unveiled at the point in time the query is executed.

Based on the approach proposed by Curtmola et al. for searchable symmetric encryption we investigate if this approach is applicable for encrypted databases providing varying functionality. Particularly, we aim for provably secure mechanisms that are practical – specifically with respect to the required runtime for query execution over encrypted data. For each novel mechanism presented in this thesis we formally quantify the information that can be extract from the initially encrypted database and the subsequently executed queries. Based on cryptographic assumptions we proof these information as an upper bound. Further, we analyze the amortized runtime of each mechanism and benchmark the runtime based on an implementation.

Kurzfassung

Verschlüsselungsverfahren bieten Schutz für Daten und können den Zugriff auf diese Daten durch unbefugte Dritte verhindern. Mit dem Paradigmenwechsel von lokalen Server und Rechenzentren hin zu Cloud-Computing und dem Angebot der Datenspeicherung (zum Beispiel in Form von Datenbanken) als Dienstleistung, sind Verschlüsselungsverfahren als Schutzmechanismus besonders für sensible Daten essentiell geworden. Jedoch verhindern Standardverschlüsselungsverfahren die Erbringung der vereinbarten Dienste auf verschlüsselten Daten, so dass auf spezialisierte Verschlüsselungsverfahren ausgewichen werden muss. Mit voll-homomorpher Verschlüsselung gibt es eine theoretische Lösung, die dieses Problem für alle Arten der Berechnung löst und gleichzeitig semantische Sicherheit für die Daten liefert. Es ist jedoch möglich, durch den Einsatz weniger universeller Verschlüsselungsverfahren, also Verfahren, die in ihrer Funktionalität eingeschränkt sind, und durch das Abschwächen der Sicherheitseigenschaft, den Berechnungsaufwand auf verschlüsselten Daten und den benötigten Speicherplatz für Chiffretexte stark zu verringern.

Für das Szenario von Datenbanken als Dienstleistung ist mit „CryptDB“ bereits 2011 solch ein Ansatz vorgeschlagen worden. Er basiert auf speziellen Verschlüsselungsverfahren, die gewisse Relationen verschiedener Klartexte selbst nach Anwendung der Verschlüsselungsoperation beibehalten. Der Vorteil dieses Ansatzes ist, dass interne Indexstrukturen des Datenbanksystems auch auf verschlüsselten Einträgen weiterhin funktional sind, da sie gerade auf den beibehaltenen Relationen basieren. Gleichzeitig sind diese Relationen auch aus verschlüsselten Daten noch zu extrahieren, und liefern somit Informationen über die komplette verschlüsselte Datensammlung an potentielle Angreifer. Alternativ existieren Ansätze, die darauf abzielen, die Sicherheit für die verschlüsselte Datenbank zu erhöhen und weiterhin Abfragen darauf zu ermöglichen, beispielsweise indem nur die Information aufgedeckt wird, ob ein Datenbankeintrag eine gegebene Anfrage erfüllt oder nicht. Allerdings benötigen diese Ansätze spezielle Indexstrukturen, um für große Datenmengen für den praktischen Einsatz performant genug zu sein.

In dieser Dissertation wird untersucht, ob solch ein Ansatz beweisbar sichere Methoden für das Auslagern von verschlüsselten Datenbanken liefert, die weiterhin Datenbankoperationen verschlüsselt ermöglicht, und die gleichzeitig für den realen Einsatz praktikabel sind – insbesondere im Hinblick auf benötigte Ausführungszeiten. Genauer wird hierfür einerseits die jeweilige Sicherheitseigenschaft, das heißt die Informationen, die aus der Datenbankanfrage und dem entsprechenden Ergebnis Daten extrahiert werden, formal quantifiziert und anschließend basierend auf kryptographischen Annahmen als obere Grenze bewiesen. Andererseits werden Ausführungszeiten zunächst durch eine theoretische Analyse abgeschätzt und anschließend durch eine praktische Implementierung für einen speziellen Anwendungsfall demonstriert.

Das Ergebnis dieser Arbeit besteht aus neun Kapiteln, die sich folgendermaßen zusammenfassen lassen. Kapitel 1 beginnt mit einer ausführlichen Motivation der Fragestellung, die in dieser Arbeit behandelt wird. In Kapitel 2 werden die formalen Grundlagen, Definitionen und technische Details, auf die im Folgenden immer wieder zurückgegriffen wird, genauer dargelegt. Kapitel 3 gibt eine allgemeine Übersicht über verwandte Arbeiten und bisherige Lösungen, die das Rechnen auf Daten und insbesondere das auf verschlüsselte Datenbanken ermöglichen. Eine detaillierte Übersicht, die sich mit speziellen Ansätzen auseinandersetzt, die nur eine konkrete Datenbankoperation adressieren, werden in den jeweiligen Kapiteln diskutiert. Das formale Rahmenwerk, das die Analyse der einzelnen Ansätze dieser Arbeit sowohl hinsichtlich der Sicherheitseigenschaften, als auch der Laufzeiteigenschaften ermöglicht, wird in Kapitel 4 vorgestellt.

In Kapitel 5 wird eine Lösung für verschlüsselte Datenbanken diskutiert, die Gleichheitsabfragen ermöglicht. Hierbei werden dynamische Datenbanken betrachtet, das bedeutet, der Inhalt der Datenbank ist nicht statisch und es können verschlüsselte Einträge modifiziert, entfernt oder hinzugefügt werden. Im Vergleich zu bisherigen Ansätzen für Gleichheitsabfragen auf dynamischen Datenbanken liefert die hier vorgestellte Lösung semantische Sicherheit für alle hinzugefügten Werte, nach denen noch nicht gesucht worden ist. Gleichzeitig benötigt diese Lösung keinen Zustand, der vom Anwender gespeichert werden muss, um das sichere Entfernen von Einträgen zu ermöglichen. Für das Hinzufügen von Werten ist ein solcher Zustand optional um bessere Laufzeiten zu erzielen. Die Konstruktion, die in diesem Kapitel für Gleichheitsabfragen vorgestellt wird, basiert auf der Idee, dass Ergebnisse von bisherigen Anfragen zwischengespeichert werden, sodass eine erneute Abfrage des gleichen Wertes schneller beantwortet werden kann.

In Kapitel 6 wird eine Lösung für Datenbank-Joins vorgestellt. Hierbei wird das Ziel verfolgt, bestmögliche Sicherheitseigenschaften für alle Datenbankeinträge beizubehalten, die nicht im tatsächlichen Ergebnis der abgefragten Operation auftreten. Durch die Normalisierung von Datenbanken stellen Datenbank-Joins eine wichtige Operation dar, dennoch sind bisher nur wenige Lösungen veröffentlich worden, die sich mit Datenbank-Joins auf verschlüsselten Datenbanken befassen. Alle bisherigen Lösungen legen nach einmaliger Anwendung das Ergebnis des kompletten inneren Datenbank-Joins offen, selbst wenn die komplette Datenbank Abfrage weitere Restriktionen beinhaltet und das tatsächliche Ergebnis damit nur eine kleine Untermenge des kompletten inneren Datenbank-Joins beinhaltet. Im Gegensatz dazu liefert die in Kapitel 6 vorgestellte Konstruktion fein granulare Sicherheitseigenschaften und alle Datenbankeinträge, die mögliche zusätzliche Restriktionen nicht erfüllen, sind auch nach der Join-Operation noch durch semantische Sicherheit geschützt.

In Kapitel 7 wird ein neuer Ansatz für Bereichsanfragen diskutiert. Die hier vorgestellte Lösung liefert semantische Sicherheit für alle Werte, die keine gestellte Anfrage erfüllt haben, und hat gleichzeitig eine amortisierte Laufzeit, die poly-logarithmisch in der Anzahl der Datenbankeinträge ist. Bisherige Lösungen lieferten entweder schwächere Sicherheitseigenschaften (beispielsweise wird die Ordnungsrelation aller verschlüsselten Datenbankeinträge direkt nach der initialen Verschlüsselung offengelegt) oder die Laufzeit einer Anfrage ist linear in der Anzahl der Datenbankeinträge. Der in diesem Kapitel vorstellte Ansatz greift die Idee aus Kapitel 5 erneut auf und erweitert ihn. Genauer wird ein Zwischenspeicher für die Ergebnisse von bereits gestellten Bereichsanfragen erzeugt und basierend auf weiteren Anfragen inkrementell verfeinert. Während alle Anfragen, die bereits durch diesen Zwischenspeicher beantwortet werden können, beschleunigt werden, haben alle Werte, die nicht in diesem Zwischenspeicher enthalten sind, weiterhin semantische Sicherheit.

Kapitel 8 beschäftigt sich mit der Frage, wie Strings (Zeichenketten) verschlüsselt werden können, sodass dennoch das Ergebnis einer verschlüsselten Anfrage nach Substrings (einem zusammenhängenden Teil der Zeichenkette) berechnet werden kann. Der hier vorgestellte Lösungsansatz basiert auf der Beobachtung, dass dieses Problem auf das Problem von verschlüsselten Bereichsanfragen abgebildet werden kann. Dadurch ist das Problem theoretisch durch den Ansatz aus Kapitel 7 lösbar, jedoch wird hier ein Ansatz verfolgt, der die Sicherheitseigenschaften abschwächt, dadurch aber die Laufzeiteigenschaften der vorgestellten Lösung stark verbessern. Dazu wird in diesem Kapitel eine Methode vorgestellt, wie das Problem der verschlüsselten Substring-Anfragen durch spezielle randomisierte Verschlüsselungsverfahren, die die Ordnungsrelation der Klartexte auch auf Chiffretexten beibehalten, gelöst werden kann. Da die Sicherheit solcher Verschlüsselungsverfahren stark von der Struktur der zugrundeliegenden Klartexte abhängt, wird die Sicherheit des

Verfahrens für das Verschlüsseln von Text in englischer Sprache durch das Ausführen bekannter Angriffe evaluiert.

Schließlich wird in Kapitel 9 eine Zusammenfassung der Arbeit und ein Ausblick auf weitere Fragestellungen bezüglich verschlüsselter Datenbanken gegeben.

Danksagung

Zunächst bedanke ich mich bei Jörn Müller-Quade, nicht nur dafür, dass er meine Arbeit betreut hat. Mein Dank gebührt ihm insbesondere auch dafür, dass er mir während meines Studiums bereits Einblicke in die faszinierende Welt der Kryptographie gewährt hat. Nicht zuletzt sein enthusiastischer Vorlesungsstil haben mein Interesse an diesem Gebiet geweckt. Meinen herzlichsten Dank auch an Florian Kerschbaum, der in meine Fähigkeiten vertraut hat und mir dadurch die Möglichkeit bot bei der SAP anzufangen. Florians Schule war nicht immer einfach, aber sie führte zum Erfolg und ich habe unglaublich viel lernen dürfen. Falls ich einmal nicht weiter wusste, so war auf Florians beeindruckendes technisches Verständnis und seine fachlichen sowie politischen Ratschläge stets Verlass.

Ich bedanke mich bei meinen SAP Managern, die ich im Laufe meiner Zeit bei der SAP hatte und die mir stets genug Freiheiten boten um diese Arbeit letztendlich fertigzustellen – Andreas Schaad, Roger Gutbrod, Detlef Plümper und Mathias Kohler. Auch meinen SAP Kollegen, Mitdoktoranden und Freunden bin ich zu Dank verpflichtet. Zunächst danke ich den Büronachbarn aus dem “Kryptographie Büro”: Fuhry, der zu allem etwas Qualifiziertes (?) zu sagen hatte; Anselme, der die Vielzahl an Diskussionen meist ohne Murren ertrug; sowie Andreas, der je nach Thema plötzlich mitdiskutierte. Ich zähle auch das “Anonymisierung Büro” zu meinen Kollegen und Freunden und danke Benjamin, Daniel und Jonas für die vielen gemeinsamen Kaffeepausen und/oder Feierabendbiere.

Meinen Eltern danke ich, dass sie mir durch ihre Erziehung meine Neugierde bewahrt haben und bei mich all meinen Entscheidungen unterstützten.

Abschließend danke ich meiner langjährigen Freundin Laura, dass sie immer für mich da ist und mich auch in den schwersten Zeiten ertragen hat und hinter mir stand, sowie meinem Sohn Phineas, der mich mit seiner Geburt drängte die Arbeit fertigzustellen.

Contents

Acronyms

ABE	attribute-based encryption.
AES	advanced encryption standard.
CP-ABE	ciphertext-policy attribute-based encryption.
DBaaS	databases as a service.
DBMS	database management system.
ECB	electronic codebook mode.
FE	functional encryption.
FHE	fully homomorphic encryption.
FHOPE	frequency-hiding order-preserving encryption.
IND-CCA	indistinguishability under chosen ciphertext attacks.
IND-CPA	indistinguishability under chosen plaintext attacks.
IND-OCPA	indistinguishability under ordered chosen plaintext attacks.
IPE	inner product encryption.
KP-ABE	key-policy attribute-based encryption.
MAC	message authentication code.
OPE	order-preserving encryption.
ORAM	oblivious RAM.
ORE	order-revealing encryption.
PDN	private data networks.
PEKS	public-key encryption with keyword search.
PHE	partially homomorphic encryption.
PIR	private information retrieval.
POPE	partial order-preserving encoding.
PPE	property-preserving encryption.

PPT probabilistic polynomial-time.

PRF pseudorandom function.

PRNG pseudorandom number generator.

PRP pseudorandom permutation.

PSI private set intersection.

RPE range predicate encryption.

SGX Software Guard Extension.

SMC secure multiparty computation.

SQL structured query language.

SSE searchable symmetric encryption.

SWHE somewhat homomorphic encryption.

UDF user defined function.

1 Introduction

In this thesis we present novel constructions enabling complex database queries over encrypted data. All approaches presented in this thesis can be deployed on common hardware while providing security guarantees proved in a formal framework. In Section 1.1 we discuss the motivation of this work and in the following Section 1.2 we summarize the scientific contribution of this dissertation. An overview of the structure of this thesis finalizes this chapter in Section 1.3.

1.1 Motivation

Within recent years, more and more aspects of our daily life have become affected by computer systems. This development has opened novel opportunities in our modern life such as online banking and online shopping, as well as personalized advertisement and improved health services. While the core business of most companies is not directly founded on information technology, they profit from these innovative capabilities in data analytics and knowledge management provided by such technology. As a result, these companies strive to minimize their direct expenses for computer systems and personnel capable in installing and maintaining these systems.

Cloud computing is one approach with increasing popularity that addresses this issue. It is based on the idea of outsourcing the computation to third parties that focus on providing computational resources and the maintenance thereof [58] as provided by, e.g. Google, Amazon and Microsoft. Particularly, cloud computing allows a data owner to outsource her data collection while enabling her to access this data with arbitrary devices anytime over a computer network. Even devices with small computational power and memory can be used to access an enormous data collection. This is possible by delegating computationally expensive operations like searching specific data records to the cloud service provider. Then only a small subset matching the search query is transferred back and processed directly by the client's lightweight device.

Since most services are based on a large amount of data, databases as a service (DBaaS) as suggested by Hacigümüs et al. [75] has become a common application for cloud computing.

While data is of great value providing novel business cases, this value also motivates criminals to attack the systems and steal outsourced data. As a result, cloud service providers and users thereof apply multiple security mechanisms such as network monitors and firewalls, enforcement of security policies, virus scanners and static or dynamic code scans to prevent deployment of vulnerable programs. However, all these security mechanisms do not protect against insider attacks such as, e.g. malicious employees of the service provider with direct physical access to the server or virtual access in form of administrator rights.

Preventing data breaches and providing confidentiality even against such insider attacks can be achieved by encrypting all data on the client side before it is transferred into the cloud. In this scenario it is crucial that the corresponding decryption key is never transferred to the cloud service provider, otherwise insider attackers can extract the decryption key and bypass this protection mechanism. While general encryption schemes such as AES provide data confidentiality for outsourced data, they do not support computation over ciphertexts and hence prevent straightforward deployment for DBaaS scenarios.

Fully homomorphic encryption is a general approach enabling computation over encrypted data. The theoretical capabilities of such encryption schemes are nearly unlimited thus the general idea of fully homomorphic encryption (FHE) has been formulated decades ago [127]. The first instantiation of fully homomorphic encryption has been proposed by Gentry [61] receiving a lot of public attention, hence non-cryptographers consider FHE as a solution suitable for all problems regarding data confidentiality. However, other solutions developed for specific scenarios have gained less attention by non-experts. While these solutions have limits in their supported functionality and might leak additional information compared to FHE they achieve smaller ciphertexts (and hence reduced memory footprint) and faster computation time. That is, these solutions trade functionality and security for better performance.

With CryptDB, a very performant approach for outsourced databases has been proposed by Popa et al. [123] based on property-preserving encryption. Unfortunately, systems founded on property-preserving encryption provide security with debatable consequences. Indeed, Naveed et al. [116] have demonstrated that systems founded on this approach are exploitable with devastating results, e.g. a data breach affecting up to 80% of the outsourced data seemingly protected by property-preserving encryption. As a result, solutions trading performance for security is of great interest for sensitive data outsourced in the DBaaS scenario. One example for an encryption scheme realizing such tradeoff that is specifically crafted supporting exact keyword search over data protected by randomized encryption (instead of deterministic encryption as used in CryptDB) is searchable symmetric encryption initially proposed by Song et al. [137]. In more detail, the data owner can encrypt values with searchable symmetric encryption (SSE) using her secret key and transfers the ciphertexts to the cloud service provider. Using the same secret key the data owner can generate a search token for a keyword and pass it to the cloud service provider. Given this search token to the cloud service provider she can filter for all ciphertexts that match with this search token. Further lines of work on SSE have defined a framework for formal security proofs [49, 84] with the goal to quantify an upper bound of the information leaked during the initial outsourcing process as well as the information extractable from each query execution. Following this framework, we design and implement different schemes for the DBaaS scenario enabling execution of varying query types over encrypted data.

1.2 Contribution of this Work

The main research question investigated in this dissertation can be formulated as follows:

Is it possible to design a practical system that enables queries over an encrypted database while still being provably secure and providing a possibility to quantify the leakage induced by the initial outsourcing step and by each query executed over encrypted data?

This question involves two different aspects to be considered we elaborate in the following.

First, what is the nature of a practical system and what are its properties? In this thesis, we demonstrate practical feasibility by the deployment of such system on common hardware for each construction. Based on a prototype we have implemented we report concrete runtime numbers. In order to give a profound assessment independently of the underlying hardware, we additionally state an amortized runtime analysis for each construction; we assume amortized runtime that is sublinear in the database size to be vital for practical relevance of an encryption scheme enabling queries over encrypted databases even suitable for big data scenarios.

Second, what is a suitable framework to quantify information leaked by an encryption scheme providing query functionality over encrypted data and how can one be sure that this information leakage is not exceeded? In order to answer this question we follow the approach of modern cryptography and state a rigorous proof of security. On the one hand, there exist practical solutions as proposed by Popa et al. [123] that are adopted in real systems, e.g. Ciphercloud[1] and Microsoft[2]. However, these solutions lack a comprehensive security proof and the information leaked by this solution can be exploited to render the protection mechanism insufficient as demonstrated in recent attacks [72, 115]. On the other hand, there exist solutions that provably leak minimal information of the underlying database, that is, only the size of the data stored in the database. While these approaches provide semantic security for the complete database and even the computation result they are not practical as defined previously. That is, the theoretical runtime is lower bounded in the database size since sublinear runtime contradicts the semantic security, e.g. information about the result set size is extractable.

As a result, we are interested in constructions that provide a tradeoff between practicability and security. While the security characteristics are weakened to achieve practical schemes we maintain a formal security definition. We follow the security definitions introduced by Curtmola et al. [49] for encryption enabling exact keyword search. Further, we extend this approach to be suitable for additional functionality besides exact keyword search. According to this security definition we prove security for different query types in this thesis. More particular, the solutions presented in this thesis are practical yet secure and enable the following four database query types:

1. exact keyword search over encrypted data,
2. range queries over encrypted data,
3. secure database joins over encrypted tables,
4. secure substring search.

All these solutions give a positive answer for both of the previously discussed questions simultaneously. Thus, the constructions presented in this thesis can guarantee security properties while they are practically relevant, hence allow outsourcing of sensitive data into untrusted environments addressing real world scenarios.

1.3 Structure of the Dissertation

We present different approaches enabling privacy-preserving query execution of four different query types on encrypted data within this thesis. Each query type is discussed individually in a self-contained chapter, hence the reader can consult each chapter of interest independently.

The general preliminaries in cryptography and database principles laying the foundations for all chapters are given in Chapter 2 together with used notation. A reader knowledgeable in these topics can skip this chapter. In order to keep this general review as compact as possible, advanced cryptographic constructions and database principles are reviewed in the particular chapter if required there. In Chapter 3 we discuss different approaches that allow secure processing of encrypted data for general applications and introduce the idea of more specialized encryption schemes trading this generality for performance gains. However, due to brevity we only review general principles and constructions for privacy-preserving query processing on encrypted data in this overview discussion. For a thorough review of related work suitable for one particular query type we refer to the corresponding chapters dealing with one specific query type more comprehensively. Since we apply similar methodologies for different constructions, we revise our methodology in Chapter 4. After this preliminary discussion, the four major chapters are presented addressing different privacy-preserving query types, namely, equality queries in Chapter 5, database joins in Chapter 6, range

[1] `http://www.ciphercloud.com/`
[2] `https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine`

queries in Chapter 7 and substring queries in Chapter 8. Finally, a conclusion and outlook for future work and further research questions is given in Chapter 9.

Specifically, we follow a unified structure for each of the four major chapters: We start with a section introducing the problem, e.g. in form an example structured query language (SQL) query we strive to be executable over encrypted data, together with an abstract framework suitable for this problem. Specific related work on this particular query type is reviewed in the next section. The third section of each major chapter introduces special cryptographic tools and system designs that are only required in this particular chapter and go beyond the general preliminaries already presented in Chapter 2. Further, the actual implementation of the abstract framework utilizing these special tools and designs are presented in this section. In the following evaluation section a benchmark of each construction is given with respect to security and performance. Additional discussion, possible extensions and tradeoffs for the main construction but without detailed evaluations are discussed in a subsequent section before each major chapter is completed with an individual summary.

The following short abstracts summarizing the major chapters presented in this thesis finalize the introduction of this dissertation.

- In Chapter 5 we present a searchable encryption scheme for *exact keyword matching* enabling updates that leak no more information than the access pattern with asymptotically optimal search time. Constructions for efficient, dynamic searchable encryption schemes previously published suffer from various drawbacks: Either they deteriorate from semantic security to the security of deterministic encryption under updates, or they require to store information on the client side for deleted files and added keywords, or they have very large index sizes. Our construction is based on the novel idea of learning the index from the access pattern itself, hence achieving efficient processing time with small information leakage.

- In Chapter 6 we present a solution for *database joins* that minimizes the information leakage of join values not contained in the actual query result. Although crucial for practical database operations due to database normalization, the join operation of encrypted tables has rarely been addressed by research so far. Previous solutions for database joins over encrypted data leak the complete inner join result, although data queries consist of additional data constraints in common applications, e.g. expressed by additional WHERE clauses in SQL. Our solution provides fine granular security since all join values of data rows not involved in the join computation remain semantically secure.

- In Chapter 7 we present a novel scheme for *range queries* that only leaks the access pattern while supporting amortized poly-logarithmic search time. Privacy-preserving range queries allow encrypting data while still enabling queries on ciphertexts if their corresponding plaintexts fall within a queried range. Previous methods for range queries either leak additional information (like the ordering of the complete data set) or slow down the search process tremendously by scanning the ciphertext in the data collection linearly. The construction in this chapter extends the idea of Chapter 5 of an incrementally constructed search index based on a combination of the access and the search pattern. By doing so, the cloud service provider can use this search index for accelerated query processing while all values that have been never contained in any result set remain semantically secure.

- In Chapter 8 we address the problem of outsourcing sensitive strings while still providing the functionality of *substring searches*. We transform the problem of secure substring searches into range queries hence this construction is compatible with the solution presented in the previous Chapter 7. While security is one important aspect that requires a careful system design, users' acceptance of the solution depends on feasible processing time and integration efforts into existing systems. Thus, we decided to apply frequency-hiding order-preserving encryption allowing efficient query processing on common database systems without further modifications. Since the practical security characteristics of such encryption schemes heavily depend on the structure of the underlying plaintext data we provide an additional practical security evaluation of this construction assuming data protection of natural (English written) text.

2 Notation and Preliminaries

In this section we review notations and definitions used throughout this thesis in Section 2.1. Since we focus on encrypted database processing in this thesis, cryptographic mechanisms that achieve data security are reviewed in Section 2.2, together with data structures that are crucial for efficient query evaluation on databases in Section 2.3. Finally, we give a preliminary general high-level framework for privacy-preserving query execution in Section 2.4 that we will adapt for different types of queries in this thesis.

2.1 Notation

The set of binary strings of length n is denoted as $\{0,1\}^n$, the set of all finite binary strings is denoted as $\{0,1\}^*$. Given a binary string s, we denote $\texttt{len}(s)$ as its bit length. Given two binary strings $u \in \{0,1\}^n, v \in \{0,1\}^m$, the concatenation is written as $u||v \in \{0,1\}^{m+n}$. Further, $x \leftarrow \mathcal{A}$ denotes the output x for a (possible) probabilistic algorithm. The notation $[m,n]$ with $m,n \in \mathbb{N}$ and $m \leq n$ denotes the integer set $\{m,\ldots,n\}$. Sampling from a probability distribution $\mathcal{D}$ is denoted as $x \leftarrow \mathcal{D}$, sampling uniformly random from a set X is denoted as $x \overset{\$}{\leftarrow} X$. We write bold letters $\mathbf{C}$ to refer to a ordered collection of objects, the i^{th} individual object is denoted as $\mathbf{C}[i]$ or $\mathbf{C}_i$. Further, the number of objects contained in $\mathbf{C}$ is denoted as $\texttt{len}(\mathbf{C})$.

Throughout this thesis $\lambda \in \mathbb{N}$ refers to the security parameter. If not stated explicitly, we implicitly assume it is input to all algorithms in an unary representation 1^λ. A function $f : \mathbb{N} \to \mathbb{N}$ is called *negligible* if for every polynomial $p(\cdot)$ there exists a $N \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ with $n > N$ it holds that $f(n) < \frac{1}{p(n)}$. We write $\texttt{negl}$ for the set of all negligible functions and $\texttt{poly}$ for the set of all polynomials.

Algorithms are modeled as Turing Machines $\text{TM}(\cdot)$ with input tape, working tape and output tape. Further, we say algorithm $\mathcal{A}$ is *efficiently computable*, if it can be executed for any input $x \in \{0,1\}^*$ in $p(\texttt{len}(x))$ steps with $p \in \texttt{poly}$. In other words, there exists a polynomial-time Turing Machine $\text{TM}(x)$ modeling $\mathcal{A}$ that halts within $p(|x|)$ steps for any input x written to the input tape.

Extending this approach, efficient *probabilistic* algorithms are modeled as Turing Machines $\text{TM}(\cdot,\cdot)$ equipped with an additional random tape. This random tape is initialized before the Turing Machine is executed and its input is sampled from an input set $r \overset{\$}{\leftarrow} R$. Efficiently computational probabilistic algorithms that can be modeled by such Turing Machines $\text{TM}(\cdot,\cdot)$ are called probabilistic polynomial-time (PPT) algorithms.

There are functions no PPT algorithms can compute efficiently. In fact, this is pivotal for cryptography, e.g. for any computationally secure encryption scheme the decryption function for a given ciphertext should be computational infeasible without the corresponding decryption key (see Section 2.2.1) for more formal details). In order to model powerful adversaries as PPT algorithms $\mathcal{A}$, they are aided with an *oracle* $\mathcal{O}$ that is able to compute auxiliary functions in a single operation – functions the adversary $\mathcal{A}$ cannot compute on its own efficiently. This oracle $\mathcal{O}$ can be queried by $\mathcal{A}$ as a black box, denoted as $\mathcal{A}^{\mathcal{O}(\cdot)}$; given a specific function f the oracle computes, we write $\mathcal{A}^{f(\cdot)}$ to denote algorithm $\mathcal{A}$ with access to such an oracle. For example, given a decryption oracle $\mathsf{Dec}(k,\cdot)$ with hard coded decryption key k enables $\mathcal{A}^{\mathsf{Dec}(k,\cdot)}$ to ask for decrypted values of queried ciphertext but without direct access to the secret decryption key k.

A probability ensemble $\mathcal{X}$ is a set of probability distributions X_i indexed by $i \in \mathbb{N}$, that is, $\mathcal{X} = \{X_i\}_{i\in\mathbb{N}}$. Based on probabilistic algorithms the concept of computational indistinguishability of two probability ensembles can be formulated as follows. We call two probability ensembles $\mathcal{X}, \mathcal{Y}$ *computational indistinguishable* if for all PPT algorithms $\mathcal{D}$

$$|\Pr[\mathcal{D}(x) = 1 : x \leftarrow X_\lambda] - \Pr[\mathcal{D}(y) = 1 : y \leftarrow Y_\lambda]| \leq \text{negl}(\lambda).$$

2.2 Common Cryptographic Preliminaries

In this section we introduce common cryptographic primitives necessary for this thesis and revise their corresponding security definitions. We follow the definitions given by Katz and Lindell in the comprehensive textbook [85]. These constructions are the standard repertoire of cryptographic tools we repeatedly make use of within this thesis. Advanced constructions required for privacy-preserving processing of particular query types are introduced in the corresponding chapters as needed.

2.2.1 Secret-Key Encryption

The classical use case for secret-key encryption is two parties that want to transfer sensitive messages over an untrusted communication channel while preserving confidentiality. Both parties have access to the same key used for encryption by the sender and used for decryption by the receiver. Since we address secure outsourcing in most cases presented in this work, sender and receiver are the same entity and the communication channel goes over an external untrusted storage provider.

Definition 1 (Secret-Key Encryption)**.** *A secret-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *for key space* $\mathcal{K}$*, message space* $\mathcal{M}$ *and ciphertext space* $\mathcal{C}$ *consists of three PPT algorithms.*

$k \leftarrow \mathsf{Gen}(1^\lambda)$ *takes as input the security parameter* 1^λ *and outputs a key* $k \in \mathcal{K}$*.*

$c \leftarrow \mathsf{Enc}(k, m)$ *takes as input the key* k *and a plaintext message* $m \in \mathcal{M}$ *and outputs a corresponding ciphertext* $c \in \mathcal{C}$*. Sometimes we write this as* $\mathsf{Enc}_k(m)$*.*

$m \leftarrow \mathsf{Dec}(k, c)$ *takes as input the key* k *and ciphertext* $c \in \mathcal{C}$ *and outputs the decrypted plaintext message* $m \in \mathcal{M}$*. Sometimes we write this as* $\mathsf{Dec}_k(c)$

For any meaningful secret-key encryption scheme we require correctness of the decryption algorithm, that is, applying the decryption algorithms with the matching key on a ciphertext encrypted under the same key returns the original plaintext message. More formally, for any secret-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, for any $k \leftarrow \mathsf{Gen}(1^\lambda)$ and any message $m \in \mathcal{M}$ it holds that $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$.

The main purpose of encryption is to preserve confidentiality against dishonest parties that try to "break" the encryption scheme. The adversaries' goal of breaking a encryption scheme is quite vague, that is, when is a encryption said to be broken, e.g. just when the adversary can extract the secret key or already when an adversary is able to distinguish between two different messages (e.g. in a fictional military scenario it is sufficient for an adversarial eavesdropper to distinguish the encrypted order of an attack between the encrypted order of an retreat). Obviously, an adversary that can extract the secret key can also distinguish between different messages simply by decrypting both messages; the other way round any (correct) encryption scheme that is secure against such distinguishing attacks must be secure against such key extraction attacks as well. Moreover, the computation power of such potential adversaries may vary tremendously, e.g. one private person with little technical equipment in contrast to a national intelligence agency that has access to supercomputers; the same argument holds for the auxiliary knowledge an adversary has, e.g. does she know the internal structure of the system she attempts to attack or is she oblivious to such details and performs a black box attack. Note that each scheme that is secure against an adversary with fixed power, is also secure against any less powerful adversary.

In order to protect sensitive messages against a broad class of adversaries, the definition of security is given in a formal framework in modern cryptography. There are two widely accepted definitions regarding the security of secret-key encryption schemes, namely *indistinguishability under chosen plaintext attacks (IND-CPA)* and *indistinguishability under chosen ciphertext attacks (IND-CCA)*. Roughly speaking, these security concepts are defined via security experiments conducted by a challenger. In these security experiments, an adversary $\mathcal{A}$ submits two challenge messages m_0, m_1 and the challenger samples one challenge message randomly denoted as m_b and returns the ciphertext $c = \mathsf{Enc}(k, m_b)$ to the adversary. The adversary is said to be successful in this security experiment, if she assigns the correct message to the output ciphertext of the randomly sampled plaintext. The purpose of this security experiment is to model the strongest attack goal in a formal framework. Regarding the auxiliary knowledge of an attacker, the adversary $\mathcal{A}$ gets additional access to an oracle $\mathcal{O}$ the adversary can ask for help. Depending on the specific function the oracle can evaluate, the adversary can perform chosen plaintext attacks, i.e. the oracle only performs the encryption (with hard coded key) or the adversary can perform chosen ciphertext attacks, i.e. the oracle can performs the encryption and the decryption (with hard coded key) except the decryption of the challenge ciphertext. Without this constraint regarding the decryption oracle, any correct encryption scheme would become insecure since the adversary can use the decryption oracle for decrypting the ciphertext challenge and then distinguish between the plaintext challenges. However, there is no constraint for the encryption oracle, that is, any of the challenge messages can be queried to the oracle. As a direct consequence, any secret-key encryption scheme that meets such security definition requires a probabilistic encryption algorithm as highlighted by Goldwasser and Micali [67].

Formally, the IND-CPA experiment for a secret-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ for an adversary $\mathcal{A}$ is defined as shown in Definition 2.

Definition 2 (IND-CPA Experiment)**.** *The IND-CPA experiment* $\mathrm{Exp}_{\Pi,\mathcal{A}}^{\mathrm{IND-CPA}}$ *for a secret-key encryption scheme* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *and for an adversary* $\mathcal{A}$ *is defined as follows.*

1. *A secret key* $k \leftarrow \mathsf{Gen}(1^\lambda)$ *is generated.*
2. *Adversary* $\mathcal{A}^{\mathsf{Enc}(k,\cdot)}$ *gets input* 1^λ *and has access to an encryption oracle* $\mathsf{Enc}(k, \cdot)$ *with hard coded secret key generated in the previous step.*
3. *Two challenge messages* $(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda)^{\mathsf{Enc}(k,\cdot)}$ *are output by the adversary where the adversary wishes to be challenged.*
4. *A bit* $b \xleftarrow{\$} \{0, 1\}$ *is sampled randomly.*
5. *The challenge message* m_b *is chosen according bit* b*. The corresponding ciphertext challenge* $\mathsf{Enc}(k, m_b)$ *is given to adversary* $\mathcal{A}$*.*
6. *Adversary* $\mathcal{A}$ *outputs a guess* b'*.*

The experiment outputs 1 *if* b *equals* b' *and* 0 *otherwise.*

The advantage of an adversary $\mathcal{A}$ succeeding in this IND-CPA experiment is defined as the probability of the experiment outputting 1 minus the probability of correct guessing. A secret-key encryption scheme is called indistinguishable under chosen plaintext attacks, if this advantage is negligible as formalized in Definition 3.

Definition 3 (IND-CPA security for secret-key encryption)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a secret-key encryption scheme and* $\lambda \in \mathbb{N}$ *its security parameter.* Π *is indistinguishable under chosen plaintext attacks if for all PPT adversaries* $\mathcal{A}$ *the advantage of* $\mathcal{A}$ *defined as*

$$\left|\Pr\left[\mathrm{Exp}_{\Pi,\mathcal{A}}^{\mathrm{IND-CPA}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

is negligible in λ*.*

As encryption schemes that are indistinguishable under chosen ciphertext attacks (IND-CCA) are not required for the constructions presented in this thesis we omit the formal definition of IND-CCA security and refer to the comprehensive textbook by Katz and Lindell [85].

2.2.2 Cryptographic Hash Functions and MACs

A hash function $h : \{0,1\}^n \to \{0,1\}^m$ is a deterministic function that compresses its input value, that is, binary string with length n are mapped to binary strings with fixed length m where $n \gg m$. Due to this compression nature, there exist multiple values $a, b \in \{0,1\}^n$ where a *hash collision* occurs, i.e. the hash values for both inputs a and b are the same $h(a) = h(b)$. A well-designed hash function minimizes the probability of such hash collisions by spreading the output over the complete output domain in a "random looking" fashion [47]. In the remainder of this work we often assume a hash function $h : \{0,1\}^* \to \{0,1\}^\lambda$ with arbitrary finite binary strings as input instead of fixed input input length n.

A cryptographic hash function has the additional properties of *pre-image resistance* and *collision resistance*. A hash function is called pre-image resistant if it is one-way: evaluating the hash function for any given input is efficient, however, it is required to be computationally infeasible to invert the hash function. More precisely, given the output $y = h(x)$ of a cryptographic hash function for any input $x \in \{0,1\}^n$ no PPT algorithm $\mathcal{A}$ can find $x' \in \{0,1\}^n$ such that $h(x') = y$ with non-negligible probability. Collision resistance for a hash function h requires that no PPT algorithm $\mathcal{A}$ can find two values x, y such that they result in the same hash value $h(x) = h(y)$ with non-negligible probability. In the remainder of this work we simply write hash functions but refer to cryptographic hash functions if not stated otherwise.

A message authentication code (MAC) is a hash function that is parametrized with an additional secret key k. In order to highlight the difference between the secret key and the payload we denote a MAC as a function $f : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$. Given a tuple $(x, f(k,x))$ for a MAC f it his computationally infeasible to compute a collision, that is, find a message x' such that $f(k,x) = f(k,x')$ even with knowledge of the secret key k. Additionally, we require the MAC to be unforgeable: given set of tuples $\{(x_i, f(k,x_i)\}_{i \in poly(\lambda)}$ chosen by the adversary, the adversary cannot compute a valid MAC $f(k,x')$ for any value x' not contained in the given set. This property is used to protect integrity of a message x but also to ensure the authenticity of the sender to the receiver since only the sender and receiver are assumed to know the secret key k.

For theoretical proofs we either model MACs as keyed pseudorandom functions and hash functions as random oracles if required, hence we review their corresponding definitions in the following as given by Katz and Lindell [85].

Definition 4 (Pseudorandom Function)**.** *Let* $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^m$ *be an efficiently computational keyed function. We say* F *is a* pseudorandom function (PRF) *if for all PPT distinguishers* $\mathcal{D}$*, there exists a negligible function negl*$(\cdot) \in$ `negl` *such that*

$$\left|\Pr\left[\mathcal{D}^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{D}^{f(\cdot)}(1^\lambda)\right]\right| = negl(\lambda)$$

where $k \xleftarrow{\$} \{0,1\}^\lambda$ is secret key sampled uniformly at random and $f : \{0,1\}^* \to \{0,1\}^*$ is a function chosen randomly from the set of all functions mapping bitstrings with length n to bitstrings with length m. If F is bijective and the inverse function F^{-1} can be computed efficiently given the secret key k we call F a pseudorandom permutation (PRP).

Note that success probability of the distinguisher is expressed in dependence of the sampled key k. However, the distinguisher cannot access the actual key but only query an oracle that has this key hard coded, otherwise she could compute F by herself (with known key k) and compare his simulated output for a fixed string with the actual output of the oracle. This would enable the distinguisher to win the game with non-negligible probability.

In contrast, a hash function h as described previously, has no additional secret key as input but is a *deterministic* function. In a real-world implementation, the hash function is realized by standardized algorithm such as SHA-256[1]. In this thesis, we follow the popular approach vastly used for efficient implementations [17, 112, 117] and model hash functions as random oracles for theoretical security analyses – a model founded by Bellare and Rogaway [16]. From a theoretical point of view this construction is debatable as highlighted by Canetti et al. [36], however, others such as, e.g. Koblitz and Menezes [93] argue that a proof in the random oracle model is better than no formal proof at all.

Informally speaking, in the random oracle model one replaces a hash function $h : \{0,1\}^n \to \{0,1\}^\lambda$ with a random oracle $\mathcal{RO} : \{0,1\}^n \to \{0,1\}^\lambda$ that works as follows: Each participant of the protocol can query the random oracle with arbitrary input $x \in \{0,1\}^n$ and gets a uniformly sampled random answer. For each input x, the output $y = \mathcal{RO}(x)$ is sampled independently and uniformly from $\{0,1\}^\lambda$; this output y is then fixed for repeated queries of the same value x for all parties. This consistency models the standardized and deterministic nature of a hash function h for all participants. One can imagine two different ways for the random oracle to work internally: either each value $y = \mathcal{RO}(x)$ is sampled randomly on-the-fly for value x queried the first time and then stored in a big table in order to stay consistent, or the complete content of this big table is stored from the beginning on (at least for all values x queried during the protocol execution). The big advantage gained by this methodology is that the experiment environment (e.g. the challenger) can chose values as required as long as the chosen values are sampled according the correct uniform distribution. This possibility of choosing the values as required is called "programmability" and the corresponding oracle is called programmable random oracle. Note that it is easy to construct a pseudorandom function (PRF) $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^\lambda$ given a random oracle $\mathcal{RO} : \{0,1\}^* \to \{0,1\}^\lambda$, e.g., by mapping $F_k(x) = \mathcal{RO}(k||x)$[2].

2.3 Databases and Data Structures

A database is a collection of data entries that is stored and organized in a way providing functionality to query this data efficiently, e.g. filtering for specific attributes or computing data aggregation. In this thesis we assume a database consists of multiple database tables where each table has a specific number of columns and an arbitrary number of rows. A column is identifiable by a unique attribute name and each column stores specific attributes of a fixed data type, e.g. integers, strings, timestamps or links to binary files. Each row is uniquely identifiable by a subset of attribute values stored in the corresponding columns; this set of attributes is called the *primary key*. In this thesis we simplify the assumption of primary keys consisting of exactly one attribute value; this can be achieved by adding an artificial unique value (e.g. an ascending

[1] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

[2] Be aware that this construction is not secure for real hash functions based on the Merkle-Damgård construction due to length extension attacks. In fact this is a prime example for the subtlety one must keep in mind when using the random oracle model.

counter) for each primary key (possibly consisting of multiple values). We refer to these simplified primary keys as *identifier* of a data entry. More formally we define databases and their individual tables as follows.

Definition 5 (Database Table)**.** *A database table T is defined by a scheme, that is, an ordered set of attributes $\{A_1, \ldots, A_n\}$. Table T contains tuples over the domain $ID \times V_1 \times \cdots \times V_n$ called rows. Each row or data entry $(id, v_1, \ldots, v_n)$ has a unique identifier id and values $v_i \in V_i$, where V_i is the value domain of attribute A_i. We write $|T|$ do denote the size of the table, that is, the number of different tuples and hence the number of different identifiers.*

Definition 6 (Database)**.** *A database $\mathbf{D} = \{T_1, \cdots T_n\}$ is a finite set of tables as introduced in Definition 5.*

The query interface for a database is offered by a database management system (DBMS) such as, e.g. MySQL, PostgreSQL or SAP HANA. As a result, a DBMS offering flexibility in the type of queries that can be processed consists of a combination of different data structures for varying purposes. In this section we revise data structures for plaintext data that are then utilized and modified in this thesis to be compatible with encrypted data and different types of queries on this data.

One possible query type for a database table is to retrieve the complete row indexed by one single identifier queried by the database user. A data structure that supports such queries is called (regular) forward index; it indexes each data entry under its unique identifier. One way to implement such forward indexes with constant lookup time is a *hash table* or hash map for a hash function h with the identifier as input, and the location where the corresponding data entry can be found as output, e.g. a deterministic mapping of the unique identifier to a memory address the *payload* is located. The identifier used as input for the hash function is called the key of the hash table.

A more natural query type for a database table is to filter for all data entries that match a specified attribute value without knowledge of the row identifiers stored in the table. While a forward index can be used to access single data entries by their identifier in constant time, such filtering queries require a linear scan of all table rows given only a forward index. Hence, additional auxiliary indexes are crucial for a database supporting such attribute filtering queries in sublinear search time. One data structure that supports such queries is called inverted index; it indexes all data entries under their corresponding attribute value. Utilizing hash tables keyed with these attributes, the filtering operation for such indexed attributes can be performed in constant time.

EmpID	**Name**	**Dept**	**Salary**
1	Harry	Finance	4000
2	Sally	Management	7000
3	Harry	Sales	5000
4	George	Sales	3500

Table 2.1: Full Database Table

Name	**EmpIDs**
Harry	1, 3
Sally	2
George	4

Table 2.2: Inverse Index

Although query execution for exact pattern matching on a database resulting in all data entries matching one specified attribute value seems quite easy, it requires engineering effort to implement this query execution with sublinear search time. In order to generally describe the functionality offered by a (encrypted or unencrypted) DBMS we write q to describe an arbitrary query compatible to this DBMS. Given a particular database $\mathbf{D}$ containing actual values we denote $\mathbf{D}[q]$ as the *result set* after a successful execution of database query q on this database $\mathbf{D}$. In this work we assume query result $\mathbf{D}[q]$ consists of row identifiers that match query q but we omit complete database entries in our notations for readability reasons. For example, assuming database $\mathbf{D}$ as specified in Table 2.1 and a query q filtering for all employees with salary between 3200 and 4500 the result set $\mathbf{D}[q]$ consists of the identifiers 1 and 4. These identifiers can then be used to reconstruction the complete data entries (1, Harry, Finance, 4000) and (4, George, Sales, 3500).

Note that a first step towards an encrypted database can be achieved in a straightforward way for the payload, i.e. all attributes that are never used in any query. That is, these values can be encrypted using a common secure encryption scheme such as, e.g., the advanced encryption standard (AES) specified by NIST[3]. For example, assuming database $\mathbf{D}$ as specified in Table 2.1 and the only query type supported by the given DBMS are range queries over the salary attribute, then all values for the attribute “Salary” are stored in plaintext enabling range queries while values for attributes “Name” and “Dept” can be encrypted with any common encryption scheme. While this approach is a first step it is no sufficient solution: Even thought the names are encrypted in this database containing all employees of our imaginary company, a potential attacker might be able to reconstruct some of these entries. Given auxiliary knowledge, e.g., the name of the chief execution officer of this imaginary company, then a potential attacker concludes that the encrypted name in the row containing the maximum salary value (stored in plaintext) is the CEO’s name with high chances. Thus, we focus on encryption schemes supporting query execution on encrypted data and omit the payload encryption in our considerations in the remainder of this thesis.

2.4 Privacy Preserving Query Execution

In order to be performant for real-world applications – especially in the context of big data – the approach of search indexes has been transferred to methods providing privacy preserving query protocols. While the historical development of such constructions is discussed more comprehensively in Section 3.3 we introduce a general high-level framework for privacy preserving query execution in the following. We assume the complete database $\mathbf{D}$ to be outsourced is initially stored on the trusted environment on the client side and a privacy-preserving search index supporting a specific type of queries is constructed there before outsourcing. This search index is then protected by cryptographic means and can be given to an untrusted party without additional trust assumptions. More specifically, we assume the untrusted party (e.g. the service provider offering DBaaS) in this work to be an *honest-but-curious* (or semi-honest) adversary. That is, while the server follows the specified protocol, it keeps track of all exchanged messages and tries to extract additional information from this transcript. This attacker model is particularly reasonable for a rational economically motivated service provider: although the service provider follows the specified protocol she is interested in additional information that could be monetized. In contrast, a malicious adversary is a much stronger model where the service provider is not following the specified protocol but potentially mishandles executions, deletes data completely or suppresses parts of the result set. However, such kind of attacks would become suspicious with a great likelihood and results in a damaged reputation contradicting the economical interests of the service provider.

Formally, the general definition of framework for privacy-preserving query execution in the honest-but-curious model is stated as follows.

Definition 7 (General Framework for Privacy-Preserving Query Execution)**.** *A protocol* $\Pi = \big(\mathsf{Gen}, \mathsf{Enc}, \mathsf{QueryToken}, \mathsf{Query}\big)$ *for general privacy-preserving query execution consists of the following (possible probabilistic) algorithms:*

$K \leftarrow \mathsf{Gen}(1^\lambda)$ *is executed on the client. On input of security parameter* 1^λ *this algorithm outputs a master key* K.

[3] http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

$\mathbf{C} \leftarrow \mathsf{Enc}(K, \mathbf{D})$ *is executed on the client. On input of master key K and an initial database $\mathbf{D}$ this algorithm outputs the corresponding encrypted version $\mathbf{C}$.*

$\tau_q \leftarrow \mathsf{QueryToken}(K, q)$ *is executed on the client. On input of master key K and a query q compatible to $\mathbf{D}$ this algorithm outputs an query token τ_q compatible with $\mathbf{C}$, the encrypted version of $\mathbf{D}$.*

$\mathbf{D}[q] \leftarrow \mathsf{Query}(\tau_q, \mathbf{C})$ *is executed on the server. On input of query token τ_q and an encrypted database $\mathbf{C}$ this algorithm outputs the result set $\mathbf{D}[q]$.*

The procedure for outsourcing a database to an untrusted environment is initiated by the client calling $\mathsf{Gen}(1^\lambda)$ creating a master key mk. Obviously, this master key is never transferred outside the client's trusted environment. In the next step the client encrypts the database $\mathbf{D}$ to be outsourced by calling $\mathsf{Enc}(K, \mathbf{D})$ resulting in an encrypted database $\mathbf{C}$ that allows processing of one specific query type. This encrypted database $\mathbf{C}$ is then outsourced to the untrusted environment. Each database query q (of compatible type) is transformed by the client using the master key K by calling $\mathsf{QueryToken}(K, q)$ resulting in a query token τ_q. The query token τ_q is transferred to the untrusted environment where the encrypted database $\mathbf{C}$ is stored and used there to retrieve the query result $\mathbf{D}[q]$ calling $\mathsf{Query}(\tau_q, \mathbf{C})$.

Note that this framework addresses *static databases* encrypted in the initial step and not alterable afterwards. That is, adding or deleting or updating data entries after the initial outsourcing step is not supported in this case. This framework can be extended to address *dynamic databases* by providing supplementary token algorithms for add and delete tokens (requiring the master key K for creation) that allow database alterations after the initial encryption step. Updates for a table row are then supported by deleting the obsolete database entry from the encrypted database and adding the updated database entry the index after the deletion.

3 Related Work

In this chapter we review related work enabling database queries over encrypted data. In Section 3.1 we discuss foundational work on databases outsourcing and complete database systems enabling query execution over encrypted data. In the following Section 3.2 we discuss property-preserving encryption widely used in solutions for encrypted databases. Attacks on property-preserving encryption are reviewed in Section 3.2.3 as these attacks give evidence that such encryption schemes provide weak security characteristics and are insufficient to provide data confidentiality. Next, in Section 3.3 we present searchable encryption with provable upper bounds for the leaked information extractable from each executed query. We elaborate on this formal security definition widely used for searchable encryption in the following Chapter 4. In Section 3.4 we outline homomorphic encryption and functional encryption. Due to their generality, these encryption schemes can be utilized to address encrypted databases theoretically. However, these general solutions have poor performance compared to solutions specifically developed for encrypted databases or even one explicit database query type. In Section 3.5 we briefly discuss related work not completely matching our requirements for encrypted databases(cf. Section 2.4). We sketch approaches providing data confidentiality but with additional server requirements in the first two subsection and approaches with deviating security goals in the remaining two subsections. Particularly, constructions based on special trusted hardware and multiple servers are outlined in Section 3.5.1 and in Section 3.5.2. Finally, we discuss solutions protecting the query and the corresponding result in Section 3.5.3 and approaches for data anonymization in Section 3.5.4.

We emphasize that we give a supplement discussion about related work suitable for specific query types in the corresponding chapters but consider this chapter as a general overview of approaches related to protection mechanisms for query execution over outsourced data.

3.1 Encrypted Databases

The databases as a service (DBaaS) paradigm has been proposed by Hacigümüs et al. [75] with the goal to eliminate expensive software and hardware, hiring professionals and transfer these tasks to a service provider for small and medium companies (especially companies whose main business is not based on information technology). In their fundamental work they identified data privacy as one essential issue to be addressed by the database community. In the same work they proposed to implement an encryption and decryption function on the server as user defined function (UDF). Each time the client wishes to add data to the database the cryptographic key is transferred to the server in order to encrypt the payload and store the ciphertext in the database. The analogue procedure is triggered each time the client wished to query data, that is, the cryptographic key is transferred to the server in order to decrypt the stored data and execute the query over the retrieved plaintext. Obviously, this solution offers no protection against an untrusted database service provider since she can modify the UDF such that the cryptographic key is copied and stored, however, it provides security against snapshot attackers, e.g. external adversaries that may get access to disk files. In summary, in this work they present a solution that is secure against outsider attackers but can be attacked by insider attackers.

In the same year, Hacigümüs et al. presented a preliminary solution for the identified data privacy and security challenges without the intermediate decryption step, thus even secure against insider attackers [74].

Their construction is founded on the deployment of a coarse index that does not contain the individual data but maps data in different partitions. These partitions preserve specific properties of the plaintext data, e.g. identical values are always mapped into the same partition; however, the actual plaintext value cannot be extracted from these partitions. Such a coarse index is utilized by the untrusted database server to perform coarse query processing based on the partitions and the still encrypted result set containing false positives is transferred back to the client. Using the decryption key, this result set can be decrypted on the client side and the false positives can be corrected by the client. Varying approaches are proposed optimizing the query execution order with the target to minimize false positives and hence reduce the computation overhead for the client.

The techniques for query processing on encrypted data have been advanced by Popa et al. [123]. Compared to the construction by Hacigümüs et al., they found a solution without false positive results based on property-preserving encryption (PPE). We refer to the next Section 3.2 for technical details of property-preserving encryption schemes and discuss the general ideas of CryptDB in the remainder of this section. As the name suggests, applying this kind of encryption scheme preserves specific properties, more particular, relations – such as equality or the order relation – of plaintext values remain valid on the ciphertext output by a property-preserving encryption algorithm. Due to the preservation of these properties even on ciphertexts the deployment overhead of such encryption schemes on common database systems is low. Firstly, internal indexing techniques based on such preserved properties can directly be applied on encrypted values as on plaintext data. Secondly, queries stated by the client are directly executable on encrypted data or can be transformed by the application of the appropriate encryption scheme of the query payload. That is, grouping and sorting can be done on encrypted data and queries are supported by the encryption scheme that preserves the relation specifically queried, e.g. filter for all values that match a stated string can be implemented by searching for the corresponding ciphertext of the stated string. Obviously, the more functionality is supported by the ciphertexts output by a PPE scheme the more information is leaked in form of the preserved properties. Popa et al. propose adjustable encryption to address this dilemma, here the properties are hidden as long as they are not required to answer a query. Technically, adjustable encryption means the application of multiple encryption schemes one after the other: beginning with the weakest (but most flexible in terms of functionality) encryption scheme, then encrypting the ciphertext output by the first encryption scheme with a stronger encryption scheme providing more security but less functionality and finally hiding all properties by the application of a semantically secure encryption scheme. This adjustable encryption is applied for each database column individually and the resulting "ciphertext onion" for each cell is then stored in the database. In case of a query that is not supported by the currently stored ciphertext, one "onion layer" is peeled off by the server, that is, the client sends the decryption key to the server enabling the server to decrypt the current ciphertext unveiling the underlying ciphertext with more properties preserved. The decryption process can be repeated until the query is supported or the minimal protection level is reached according to a given policy. Additional functionality for query processing on the server side and increased performance for the ciphertext decryption on the client side can be supported by storing multiple ciphertexts of the same plaintext supporting different functionalities in parallel in the database. Obviously, the security level for the plaintext is as low as the security level provided by the weakest encryption scheme.

Companies with varying sizes have implemented the approach of CryptDB: not only start-up such as, e.g. Ciphercloud[1] or Vaultive[2], but also big players such as, e.g. Microsoft[3] offer commercial solutions for

[1] `http://www.ciphercloud.com/`
[2] `http://www.vaultive.com/`
[3] `https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine`

encrypted databases inspired by CryptDB. Further, Popa et al. demonstrated the possibility to build web applications on top of encrypted databases implementing the approaches of CryptDB [124].

The practical evaluation results demonstrate that performance overhead of property-preserving encryption is negligible [71, 123]. The reason for this minimal performance overhead is that indexing techniques provided by the database engine can result in huge search time speed-up and are directly applicable for property-preserving encryption. That is, data structures such as hash tables can be applied to deterministic encryption providing fast processing time for exact keyword search and even advanced data structures such as search trees are compatible with order-preserving encryption providing fast processing time for range queries.

3.2 Property-Preserving Encryption

The general framework of a property-preserving encryption scheme has been stated by Pandey and Rouselakis [119] as given the following definition.

Definition 8 (Property-Preserving Encryption)**.** *A property-preserving encryption scheme* $\Pi = \big(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\big)$ *for plaintext space* $\mathcal{M}$ *and a binary property* $P : \mathcal{M} \times \mathcal{M} \rightarrow \{0, 1\}$ *consists of the following four (possibly probabilistic) polynomial-time algorithms:*

$K \leftarrow \mathsf{Gen}(1^\lambda)$ *is a probabilistic algorithm that takes as input the security parameter* λ *and outputs a secret key* K.

$c \leftarrow \mathsf{Enc}(K, m)$ *is a possibly probabilistic algorithm that takes as the secret key* K *and a plaintext* $m \in \mathcal{M}$ *and outputs a ciphertext* c.

$m \leftarrow \mathsf{Dec}(K, c)$ *is a deterministic algorithm that takes as input the secret key* K *and a ciphertext* c *and outputs a plaintext* m.

We require that the property is preserved after the encryption, that is, for all messages $m_1, m_2 \in \mathcal{M}$ *and all secret keys* K *output by* Gen *it holds that* $P\left(m_1, m_2\right) = P\left(\mathsf{Enc}(K, m_1), \mathsf{Enc}(K, m_2)\right)$.

The practical advantage of property-preserving encryption is natural, namely, it is easy to integrate such encryption schemes into existing databases. However, the security consequences are debatable and heavily depends on the underlying plaintext distribution to be stored in such encrypted database as we will see in the following Section 3.2.3.

3.2.1 Deterministic Encryption

Obviously, for deterministic encryption and adversary can win the IND-CPA experiment as stated in Definition 2. Specifically, each time the same plaintext m is input to the encryption algorithm the same ciphertext c is output and further the adversary has access to an encryption oracle with no restrictions. Thus, the adversary can submit both challenge messages to the encryption oracle even before the challenge submission and learn the corresponding (deterministic) ciphertexts making it trivial to win the experiment. This determinism is even worse for the asymmetric encryption setting, where no encryption oracle is required since everyone (including the adversary) is assumed to know the public encryption key. Bellare et al. investigated deterministic encryption in the asymmetric encryption setting and the formal security notions thereof [14]. The resulting security notation is indeed based on the idea to limit the access to the public encryption key. More specifically, the security experiment contains two stages between separate adversaries. The first adversary does not gain access to the encryption key and is asked to choose a sequence of challenge messages together with some side information thereof. The resulting challenge sequence is then submitted to a second adversary that has access to the encryption key, with the task to extract and output the side information. Boldyreva et al. proposed the first asymmetric encryption scheme without a random oracle [23] and Bellare et al. extended the security definition with variants thereof [15]. The construction of symmetric encryptions schemes that output deterministic ciphertexts can be achieved quite natural and are appropriate for the cases of encrypted databases. In more details, blockciphers are deterministic by definition and can be randomized by specific modes of operation based on additional randomized initialization vectors. Hence, either choosing the deterministic electronic codebook mode (ECB) mode or fixing the initialization vector results in a deterministic symmetric encryption scheme as also proposed for CryptDB [123].

3.2.2 Order-Preserving Encryption

While exact keyword search and grouping is compatible with deterministic encryption, sorting and range queries can be executed over ciphertexts that are output by order-preserving encryption (OPE) as introduced by Agrawal et al. [5]. Compared to the construction proposed by Hacigümüs the following construction by Agrawal et al. yields no false results. In the work by Agrawal et al., the security properties of their construction is only analyzed for a fixed plaintext distribution; however, the attacker's advantage heavily depends on the plaintext distribution and the work lacks a theoretical rigorous security analysis. Boldyreva et al. [21] have introduced two thorough security definitions for OPE, that is, indistinguishability under ordered chosen-plaintext attack (IND-OCPA) and the weaker definition of pseudorandom order-preserving function against chosen-ciphertext (POPF-CCA) attack. They proved that IND-OCPA security can only be achieved by an algorithm with ciphertext spaces that are exponentially large (in the plaintext space) or requiring additional client state. Further, they have succeeded in constructing a scheme that achieves POPF-CCA security. This construction is the first OPE scheme that only requires a secret key to be stored on the client side. In their subsequent work [22], Boldyreva et al. have stated that a random order-preserving function leaks the high-order bits of the plaintext.

Popa et al. [122] proposed the first construction that achieves IND-OCPA but requires a state with storage size linear in the number of distinct plaintexts. This state is stored on a dedicated OPE server and this state is then retrieved in an interactive multi-round protocol. Further, this state is mutable hence the encryption of novel values might trigger an updating process. Kerschbaum and Schröpfer improved the construction by Popa et al. eliminating the necessity of a dedicated OPE server but also improve the construction such that the probability of an event requiring the updating process is negligible [91].

The general idea of OPE is that the ordering relation on plaintexts is preserved for ciphertexts, e.g. given x, y with $x \leq y$ then $\mathsf{Enc}^{\mathsf{OPE}}(x) \leq \mathsf{Enc}^{\mathsf{OPE}}(y)$. Specifically, the identical function $\leq$ can be evaluated on plaintexts and on ciphertexts always yielding the same result. A generalization of OPE is given with order-revealing encryption (ORE); again the ordering information can be extracted from two ciphertexts output by ORE but a publicly known function f has to be evaluated with the ciphertexts as input [27, 44, 97]. As a result, the integration overhead of such order-revealing encryption schemes into existing database system increases, e.g. the $\leq$ operator must be rewritten. Nevertheless, index structures for ORE encrypted data can be generated by the database system after the initial upload since the special function f can be evaluated publicly by any party. As a consequence, the information leakage of ORE is the same as the information leakage of OPE and any party – including an adversary – can extract the order of all outsourced ciphertext[4]. Thus, in the remainder of the chapter whenever we discuss the information leakage of property-preserving encryption we implicitly address property-revealing encryption as well.

[4] This information leakage renders OPE and ORE encryption insecure in the asymmetric key setting. Particularly, assuming a public key algorithm outputting a valid order-revealing ciphertext, an adversary can decrypt any value with computation effort that is logarithmic in the plaintext domain size by running a binary search

3.2.3 Attacks on Property-Preserving Encryption

As discussed in the previous sections it is intended by design of PPE that information about the plaintext values can be extracted from the ciphertexts. The practical consequences of this information leakage, however, are not clear and heavily depend on the side knowledge of an adversary and the distribution of the plaintext values. For deterministic encryption this weakness has been known and exploited for centuries[5]. For example, a mono-alphabetic substitution cipher applied on each character encrypting natural language is attackable with simple frequency analysis assuming the language and its character distribution as auxiliary information known by an adversary. Even more information than just the frequency of each ciphertext is extractable for ciphertexts output by deterministic order-preserving encryption, hence the question regarding the practical security provided by such encryption scheme is quite natural.

With the growing popularity of PPE based on the publication of CryptDB, this question has been examined by the academic community. A first practical implementation demonstrating successful attacks has been presented by Naveed et al. [115]. They proposed two kinds of attacks on order-preserving encryption, namely the sorting attack for targets that contain the most part of the message space in encrypted form, and the cumulative attack that combines ordering information with frequency information. The sorting attack is particularly powerful against values drawn from a small plaintext domain. Using combinations of sorting attacks and cumulative attacks Naveed et al. were able to empirically recover more than 80% of the patient records for 95% of encrypted databases containing electronic medical records encrypted with deterministic OPE.

In order to mitigate these kind of attacks, Kerschbaum proposed a stateful but frequency-hiding order preserving encryption scheme that does not only fulfill the IND-OCPA security but the strictly stronger security notion indistinguishability under frequency-analyzing ordered chosen plaintext attacks [89]. Further, Kerschbaum proved negligible probability for the rebalancing operation of the client's state (also inducing a re-encryption) assuming a database with a huge number of data queries but moderate number of data inserts. Partial order-preserving encryption (POPE) is an alternative scheme fulfilling this strong security notion utilizing a trusted comparison oracle and has been published recently by Roche et al. [128]. In contrast to Kerschbaum's scheme, the assumption for the underlying database operations is orthogonal for POPE, that is, Roche et al. assume a huge number of data inserts but moderate number of data queries.

Another series of practical attacks against (not only deterministic) order-preserving encrypted values has been published recently by Grubbs et al. [72]. Particularly, the bucket attack is the first attempt of exploiting only the ordering characteristics without the need of frequency information, hence it is applicable on randomized order-preserving encryption. In Section 8.4.2 we elaborate on the bucketing attack and mount it against our instantiation of frequency-hiding order preserving encryption used for encrypted substring queries.

Lacharité et al. proposed another line of attacks that do not require auxiliary data [96]. They assume a dense dataset (i.e. each value taken from the domain is encrypted at least once) and uniform queries leaking the access pattern. Further improvements are presented given additional rank leakage for each range query as induced by all (frequency hiding) order-preserving encryption schemes. Based on this information leakage Lacharité et al. have stated algorithms that are able reconstruct all plaintexts after $N \log N + O(N)$ uniformly distributed queries where N is the domain size.

[5] Queen Mary of Scots was sentenced to death in 1587 based on deterministically encrypted letters that have been cracked.

3.3 Searchable Encryption

All successful attacks on property-preserving encryption exploit the fact that the complete ciphertext collection leaks the relation of each ciphertext to all other ciphertexts contained in the collection. Especially, this information is leaked directly after the initial encryption (and outsourcing) step and enables the server to construct advanced data structures to accelerate the processing time of queries that might be stated in the future.

Hiding the relation between multiple ciphertexts after the initial encryption step by semantically secure encryption but unveiling information only if queried has been proposed in the preliminary work by Song et al. [137]. Specifically, with their work on SSE they investigated queries for exact keyword search. SSE enables the client with access to the master key to create a search token for a keyword to be queried. Given this search token to the party holding the collection of ciphertexts output by SSE, each ciphertext can be compared with the search token unveiling if it matches the keyword or if it does not. More formally, a SSE scheme can be described as stated in the following definition.

Definition 9 (Searchable Symmetric Encryption Scheme as proposed by Song [137])**.** *A secure searchable symmetric encryption scheme* SSE *is a tuple of four (possibly probabilistic) polynomial-time algorithms:*

$K \leftarrow \mathsf{SSE\text{-}Setup}(1^\lambda)$ *is a probabilistic algorithm that takes as input a security parameter* λ *and outputs a master key* K*.*

$c_m \leftarrow \mathsf{SSE\text{-}Enc}(K, m)$ *is a probabilistic algorithm that takes as input a master key* K *and a plaintext* m*. It outputs a randomized ciphertext* c_m*.*

$t_w \leftarrow \mathsf{SSE\text{-}Token}(K, w)$ *is a deterministic algorithm that takes as input a master key* K *and a keyword* w*. It outputs a (deterministic) search token* t_w*.*

$\{0, 1\} \leftarrow \mathsf{SSE\text{-}Match}(c_m, t_w)$ *is a deterministic algorithm that takes as input a ciphertext* c_m *and a search token* t_w*. It returns* 1 *if* $w = m$ *with* $c_m \leftarrow \mathsf{SSE\text{-}Enc}(K, m)$ *and* $t_w \leftarrow \mathsf{SSE\text{-}Token}(K, w)$ *generated with the same master key* K*. Otherwise, this algorithm returns* 0*.*

Motivated by encrypted emails providing search functionality, a scheme for public-key encryption with keyword search (PEKS) has been presented by Boneh et al. [24]. Particularly, any party knowing the public key can encrypt values, while only the holder of the master key can generate search tokens.

Note that both in the symmetric and the asymmetric setting each ciphertext must be tested individually, hence this approach results in linear search time scanning a encrypted ciphertext collection. As discussed in Section 2.3 we assume linear runtime to be too inefficient for big data scenarios. Thus, indexing techniques for encrypted data resulting in huge search time speed-up are vital for such scenarios. Goh has published the first scheme using indexing techniques for searchable encryption based on obfuscated Bloom filters for each indexed document [64]. With constant lookup time per Bloom filter the search time is reduced from linear in the number of indexed keywords to a search time that is linear in the number of indexed documents. Due to the nature of Bloom filters, however, this approach might raise false positives that must be corrected on the client side. Further, Goh stated the first formal security definition for secure search indexes known as semantic security against adaptive chosen keyword attacks (IND-CKA).

Chang and Mitzenmacher have proposed a scheme not yielding false positives founded on the idea of one prebuilt dictionary per document [41]. Each dictionary is encoded as bit-vectors and blinded before it is outsourced. The search token generated by the client enables the server to lift the blinded bit corresponding to the queried keyword. In addition, they proposed a stronger security definition compared to Goh's with the goal to protect the document size and provide security for the search words.

In the fundamental work by Curtmola et al. [49], they have proposed the first construction with optimal search time based on inverted indexes per keyword. Further, they highlighted a flaw in the security definition stated by Chang and Mitzenmacher and gave an alternative simulation-based definition for security. They proved that their simulation-based definition is at least as strong as a security definition based on semantic security with the result simulation-based security is still widely used for SSE. Refer to Section 4.1.1 for a thorough discussion of this definition.

Recall that all these enhanced constructions for keyword search over encrypted documents consider a large document collection during the encryption step. This complete overview of all values to be encrypted provides the possibility to construct an encrypted search index during the initial encryption hence achieving faster execution time (compared to a linear scan of all encrypted words). Chase and Kamara summarized such approaches in a generalization known as structured encryption with controlled disclosure [42]. As an application for this generalization they stated an encryption scheme for graphs providing neighbor queries on encrypted graphs.

The design of algorithms for structured encryption with controlled disclosure becomes even more challenging for dynamic data collections. Particularly, in the case of static datasets the content of the complete data collection can be analyzed during the initial encryption step and is immutable after this analysis. In the case of dynamic datasets, however, novel content might be added requiring an adjustment of the encrypted index structure raising novel challenges, e.g. adding a new keyword to a prebuilt inverted index might easily leak the fact that a new keyword has been added. Kamara et al. proposed the first dynamic SSE scheme based on inverted indexes with provable security in the simulation-based framework [84]. Improvements for dynamic SSE for exact keyword search have been proposed, such as, e.g. an approach enabling parallelization of the search process [83] or a construction suitable for file hosting systems such as DropBox [116].

Extensions for searchable symmetric encryption providing additional functionality have also been published, such as, e.g. complex queries containing conjunctive [69] and disjunctive [29] keyword combinations. Further results achieve significant efficiency gains for these complex queries [38, 39] and approaches for fuzzy search have been proposed [55]. For a comprehensive overview of all the different solutions for searchable encryption we refer to the survey published by Bösch et al. [30].

Although searchable encryption has a clear security advantage compared to deterministic encryption, Islam et al. demonstrate an attack based on the leaked access patterns [82]. Their attack relies on the knowledge of the distribution of specific keywords and works for a great number of ciphertexts given the corresponding search tokens. Another approach has been presented by Zhang et al. [150] based on file-injection attacks. Note that this assumption is always valid for public-key encryption with keyword search, but we deploy searchable *symmetric* encryption and assume the client to be trustworthy. Further, in both scenarios the adversary's goal is to learn the underlying keywords for a given search tokens. This extracted knowledge in combination with the access pattern leaks some information of the content of the encrypted documents, however, all non-matching ciphertexts remain semantically secure.

3.4 General Computation on Encrypted Data

Tools for computing on encrypted data have been studied by a great number of different lines of work resulting in varying solutions. In the following subsections we given an overview on function encryption and (fully) homomorphic encryption that might be suitable for encrypted databases and sketch their differences.

3.4.1 Functional Encryption

Functional encryption (FE) is an asymmetric encryption scheme with the possibility of function evaluation over ciphertexts output by the encryption algorithm [28, 129]. Particularly, anyone knowing the public key pk can encrypt value x and retrieve the corresponding ciphertext $c \leftarrow \mathsf{Enc}^{\mathrm{FE}}(pk, x)$. The holder of the corresponding secret master key can generate a function key k_f for a specific function f. Any party with access to this function key can evaluate function f over ciphertext c, i.e. $\mathsf{Dec}^{\mathrm{FE}}\left(k_f, \mathsf{Enc}^{\mathrm{FE}}\left(pk, x\right)\right) = f(x)$. Note that the function key is restricted to the function f it has been generated for and most FE schemes are restricted to functions with special characteristics. Functional encryption schemes are distinguished between schemes that are function-hiding, i.e. the function f cannot be extracted given a corresponding function key k_f, and schemes that are not function-hiding. Thus, PEKS is one special case of function-hiding FE where the function key is generated for a specified keyword to be searched for. Other functions often addressed by FE are tests for orthogonality of encrypted vectors and inner-products of encrypted vectors [6, 98], functions considering the identity [25, 129], and functions considering supplement attributes [80, 100]. Recently, theoretical approaches have been proposed enabling function key generation for arbitrary functions encoded as binary circuits [60, 66].

3.4.2 Homomorphic Encryption

Although the idea of homomorphic encryption has been formulated four decades ago [127], the first instantiation of a FHE encryption scheme has been proposed by Gentry ten years ago [61]. In theory, FHE enables the encryption of data while preserving the possibility to evaluate any function over encrypted data. In contrast to functional encryption, fully homomorphic encryption is not limited to a specific function key that is generated by the holder of the secret key, but any party can evaluate any function. Especially for non-cryptographers this flexibility has become folklore and FHE is considered as the one solution suitable for use cases requiring computation over encrypted data. The function to be evaluated is typically represented by a boolean circuit with depth that is polynomial in the input length; the individual gates are realized by multiplication or addition over a finite field, hence the encrypted input must be encoded as elements in the same finite field. Following Gentry's scheme [61], a lot of alternative schemes have been proposed amongst others [32, 33, 135, 142]. Nevertheless, FHE is too slow for practical adoption enabling the evaluation of arbitrary functions over encrypted data. One example for the required overhead induced by FHE is given by the evaluation of an AES circuit on encrypted data [62, 63], where the AES-encryption operation took 245 seconds and required 3GB of memory.

Recall that FHE provides full flexibility in the supported function because the number of multiplications and additions are only limited to be polynomial in the input length. Decreasing this flexibility with a stricter limitation in the number of, e.g. supported multiplication, yields a construction that is known as somewhat homomorphic encryption (SWHE). One prominent proposal for SWHE has been given by Boneh, Goh and Nissim [26] allowing exactly one multiplication over encrypted data and enabling the evaluation of boolean circuits in 2-DNF[6]. Most approaches for FHE are founded on SWHE in combination with a bootstrapping step. From a high-level perspective, one can imagine this bootstrapping step as an re-encryption step on encrypted data before all multiplications are consumed resulting in a refreshed ciphertext supporting a larger number of multiplications.

Further decrease of the flexibility results in partially homomorphic encryption (PHE), where only one specific operation can be evaluated on encrypted data. There exist different approaches for varying opera-

[6] DNF stands for disjunctive normal form disjunction, a logical formula that consists of disjunctions of conjunctive clauses. In case of a 2-DNF, each conjunctive clause contains at most 2 literals.

tions, such as, e.g. addition as proposed by Paillier [118] or multiplication as proposed by El Gamal [54], or bitwise XOR as proposed by Goldwasser and Micali [67].

Note that all homomorphic encryption schemes provide semantic security, particularly, even the result of the function evaluated is semantically secure. This result privacy is another difference between fully homomorphic encryption and functional encryption. As implication, even though FHE might be flexible enough to implement encrypted databases in theory, the practical consequences of this semantic security render sublinear runtime impossible. That is, the information leakage of functional encryption and specifically searchable encryption in form of the access patterns can be utilized for runtime improvements that cannot be achieved by FHE due to its strict security properties.

3.5 Additional Approaches for Outsourced Databases

In this section we describe approaches for outsourcing sensitive data that do not fully fit into our model as outlined in Section 2.4. In more details, we briefly discuss solutions for data confidentiality that require special trusted hardware in Section 3.5.1 and multiple servers in Section 3.5.2. In Section 3.5.3 we review solutions that strive to protect the query and the corresponding result. Finally, we give an overview for data anonymization in Section 3.5.4

3.5.1 Hardware-Based Approaches

The idea of hardware-based approaches for securing computation especially in hostile environments has been discussed for over 20 years [136]. Intel's recent release of Software Guard Extension (SGX) [48, 110] has amplified the popularity of hardware based protection mechanisms in the security community [9, 35, 132]. The implementation of cryptographic primitives aided by SGX has been proposed recently, e.g. functional encryption with SGX [57] or oblivious RAM with SGX [131]. The integration of SGX into nearly each processor available on the market and the small computational overhead compared to solutions solely based on cryptographic assumptions promise solutions suitable for a variety of use cases. Especially the small computational overhead enables SGX to address big data scenarios, thus it has been proposed to implement specific queries such as ranges over data protected by SGX [34] and even complete database systems protected by trusted hardware (not explicitly designed for SGX) have been proposed [7, 11]. However, the current version of Intel's SGX only supports 128 MB memory (which is shared between the program code and the payload), hence current solutions for big data utilizing SGX rely on expensive context switches. Further, recent attack vectors on hardware level such as, e.g. Rowhammer [146] or Meltdown [104] or Specte [94], demonstrate the difficulty of hardware design especially with respect to its security properties.

3.5.2 Solutions with Multiple Servers

Alternative construction for privacy-preserving databases have been proposed by Aggarwal et al. based on a distributed architecture deployed on two servers [3]. In their security model each service provider operating one dedicated server might be honest-but-curious but they do not collaborate. Aggarwal et al. discuss different approaches how such fragmentation could be implemented depending on the information to be protected. On the one hand, they propose information-theoretical security as achieved by the one-time pad encryption and distribute the random key and the resulting ciphertext on different servers. On the other hand, they propose a separation of different columns protection the confidentiality of the relation between specific columns.

Evolving the idea of Aggarwal et al., Ciriani et al. [45] proposed a fragmentation of databases on multiple servers, where parts of the fragmentation are stored in plaintext and the remainder of the complete database is stored in encrypted form on each server. Varying fragments for different queries, that is, varying column combinations are kept in plaintext on different servers; however, for all servers the corresponding decryption key remains on the client and thus the complete plaintext database can only be retrieved by the client. In contrast to the security model by Aggarwal et al. even in the case of a malicious collaboration of servers, the intersection of columns that are only stored encrypted cannot be retrieved. Depending on the query to be executed, a different fragment might be chosen to evaluate parts of the query on the plaintext columns whereas the remaining query parts are then executed on the client side over plaintext data retrieved there.

MimoSecco combines the approach of data fragmentation and secure hardware [2]. In more details, Achenbach et al. propose a database scheme that separates the data and the search index. Each record is completely encrypted before it is stored on the data server; further, an inverted index is built on plaintext data, however, the list of actual occurrences of this plaintext value is encrypted hiding the actual relation. A query is then first executed over the inverted index and based on the corresponding result the actual result is retrieved. This is achieved using the secure hardware decrypting the result returned by the index server and querying the matching payload from the data server.

Another approach using multiple servers for encrypted databases has been proposed by Bater et al. [12] for private data networks (PDN). In this scenario, each member of the PDN holds a private database that is not unveiled to any other peer of the PDN nor the query writer. The query writer submits queries to be executed on each private database and is interested in a merged result. Bater et al. proposed a solution based on secure multiparty computation (SMC) with varying optimization for each query type.

In summary, all approaches founded on varying number of servers increase the operational costs depending on the number of required servers.

3.5.3 Hiding the Access Pattern

As discussed in Section 3.4.1, FE leaks the result of the evaluated function by design. The same is true for searchable encryption, where the result set of all files (or rows in the case of databases) matching the queried search term. Private information retrieval (PIR) [95] and oblivious RAM (ORAM) [65] are two cryptographic tools that are designed to hide the access pattern. PIR hides the access pattern for each database access, however, PIR does not require the confidentiality of the content stored in the database. That is, the data might be assumed to be public but the access pattern of a client accessing this data is protected. Theoretically, PIR can be realized by homomorphic encryption evaluated over the complete database, selecting the data record queried by the user; in fact, Boneh, Goh and Nissim proposed the implementation of PIR as one application for their SWHE scheme [26].

In contrast to PIR, the construction of oblivious RAM implements a read-write memory and takes a sequence of multiple data accesses into consideration, providing the option to amortize the communication cost. From a high-level perspective, data records are organized in blocks and every block accessed is shuffled afterwards, that is, it is deleted at one position and rewritten at another position eliminating the correlation of the accessed blocks. Some constructions assume a very simple server that only supports data read and write functionality, there the shuffling step is amortized over a sequence of data access operations using a stash that remains on the client side [138] achieving poly-logarithmic communication overhead. Other ORAM constructions assume a more powerful server allowing computation on encrypted data and they achieve constant communication overhead [126]. Recent ideas haven been published combining ORAM and PIR [1, 109].

However, Naveed [114] pointed out that the combination of ORAM and SSE in order to eliminate the access pattern leakage is no practical option.

3.5.4 Data Anonymization

Compared to cryptographic solutions, data anonymization is an orthogonal tool providing protection of individual sensitive records contained in a database to be outsourced. The idea to disclose an anonymized version of a database containing sensitive information to third parties that can analyze the data by themselves has been used widely in practice. Commonly, personal identifiers are either removed or replaced with randomly looking data such as, e.g. a hashed value of the sensitive information. However, such anonymization technique is not sufficient as demonstrated by varying attacks [113, 140] and thus data anonymization resulted in its own academic research branch.

Generalization has been proposed as one approach achieving data anonymization, firstly defined by Samarati and Sweeney [130] in their work on k-anonymity. The basic idea of k-anonymity is to generalize sensitive attributes such that these attributes can be collected in buckets until different attribute buckets contain at least k records. Further improvements have been formulated with l-diversity [107] and t-closeness [101] addressing different attacks of the previous generalization approach.

Differential Privacy is an alternative to generalization technique that has been proposed by Dwork [52]. From a high-level perspective, this approach is based on the idea to randomly add Laplacian noise to each individual data record while the noise is canceled out for an aggregation of multiple records. The exponential mechanisms strives to address discrete attribute domains [111] and (ϵ, δ)-differential privacy has been proposed weakening the privacy guarantees to achieve more utility [53].

However, all anonymization approaches for databases trade privacy for accuracy of the query result (e.g. in case of differential privacy this is known as utility), that is, the result set is not complete or it might contain false positives or yield results that are not accurate. Although work on differential privacy is currently very popular, we omit further discussions since we focus on solutions yielding the correct result set after query execution in this thesis.

4 Methodology

In this chapter we describe the methodology we follow for each construction in order to evaluate our protocols enabling privacy-preserving query processing. We address four different types of queries in the following chapters, namely, filtering queries based on exact matching patterns in Chapter 5, database join queries in Chapter 6, filtering queries based on range matching in Chapter 7 and substring filters in Chapter 8. The approach for each type of query is evaluated with respect to two different dimensions, namely security on the one hand, and performance on the other hand. Depending on the type of query supported by the considered protocol and the specific design goals we apply different approaches for the evaluation. A description of these different approaches and the motivation thereof are discussed in this chapter. A preliminary version of this chapter has been published in the proceedings of the first SAP Security Search Seminar 2018.

4.1 Security Assessment Methodology

We follow different paths to assess the security properties of the solutions depending on the functionality of the query transformed into a privacy-preserving counterpart. More particular, for each construction providing basic functionalities such as exact pattern matching or range queries where we claim advanced security properties we prove this claim in a formal framework discussed in the following Section 4.1.1. In contrast, the implementation of secure substring search in Chapter 8, utilizes such fundamental queries but focus on faster processing time. The acceleration of the processing time exploits the special structure of natural (e.g. English written) text also exploitable by a potential attacker, hence we assess the security of this construction in practical ways, that is, the best known attacks applicable to this construction are run against our protocol. In the following section, we motivate and describe the standard framework for formal security proofs for searchable symmetric encryption (SSE) as firstly stated in details by Curtmola et al. [49] in 2006 and further analyzed by Chase and Kamara later [42].

4.1.1 Framework for Formal Security Proofs for SSE

Compared to general symmetric encryption schemes as reviewed in Section 2.2.1 where the security goal is to hide all information (except the length) of the underlying plaintext, the goal of searchable symmetric encryption is somewhat contradictory to this security definition. Specifically, one design goal of schemes support privacy-preserving query processing as sketched in Section 2.4 and specifically searchable symmetric encryption as elaborated in Section 3.3 is to unveil the result set matching the queried search token making some leakage of the underlying plaintext not only inevitable but worthwhile. As a direct consequence, even the weakest widely accepted security definition for general symmetric encryption schemes, that is, indistinguishability under chose plaintext attacks (IND-CPA security as given in Definition 3) cannot directly be fulfilled for any SSE scheme where an attacker can generate search tokens offered by a corresponding token oracle.

If one wants to follow the security definition of indistinguishability, a possibility solving this contradiction is to restrict the oracle queries in a way that any adversary cannot use the oracle answers to distinguish the challenges (consisting of document collections as discussed in Section 2.4) submitted by the attacker.

This idea has first been published by Goh [64] under the term "Semantic Security Against Adaptive Chosen Keyword Attack (IND-CKA)". An alternative security framework has been published by Chang and Mitzenmacher [41] for their interactive protocol they proposed for exact pattern matching on encrypted documents. They have followed the fundamental notion of simulation based proofs frequently used for secure multiparty computation [147, 148], zero knowledge proofs [68] and semantic security [67] as revised by Lindell in his comprehensive tutorial recently [103]. Curtmola et al. [49] have analyzed the relation of both security definitions given by Goh and by Chang and Mitzenmacher. Further, their work has resulted in a de facto standard security framework for analyzing the security properties offered by searchable symmetric encryption schemes and has been used in multiple constructions. We follow their approach in this thesis, hence we discuss and motivate simulated-based proofs in the context of searchable encryption in more details in the following.

From a high-level perspective, the idea of simulation based proofs is as follows: Given a protocol Π consisting of PPT algorithms $(A_1, \dots, A_n)$ with sensitive input, one defines an experiment between a challenger $\mathcal{C}$ and a distinguisher $\mathcal{D}$, often referred to as adversary in the literature. Therefore we use the terms distinguisher and adversary interchangeably in this discussion. In the initialization step the challenger $\mathcal{C}$ samples uniformly at random between two possible versions of the protocol Π the distinguisher can interact with for the rest of the experiment: one version is called the *real protocol* denoted as $\mathbf{Real}^{\Pi}$ and the other version is called the *ideal protocol* denoted as $\mathbf{Ideal}^{\Pi}$. In the real protocol, the challenger follows directly the specific implementation given for Π. That is, for any input x that is output by $\mathcal{D}$ for algorithms $A_i \in \Pi$, the challenger executes $y \leftarrow A_i(x)$ and forwards the corresponding result y to $\mathcal{D}$. In contrast, in the ideal protocol the challenger can only access a simulator $\mathcal{S} = (\mathcal{S}_1, \dots, \mathcal{S}_n)$ instead of the actual protocol implementation $(A_1, \dots, A_n)$. This simulator $\mathcal{S}$ has no access to the actual input x output by $\mathcal{D}$ but only to "some information" about x. This information about x is modeled by a leakage function $\mathcal{L}_i(x)$ input to simulator $\mathcal{S}_i$. $\mathcal{S}_i\left(\mathcal{L}_i\left(x\right)\right)$ simulates A_i given the restricted information $\mathcal{L}_i\left(x\right)$ and yields a simulated output $\widetilde{y}$. The overall idea is depicted in Figure 4.1. The security proof consists of a sound construction of

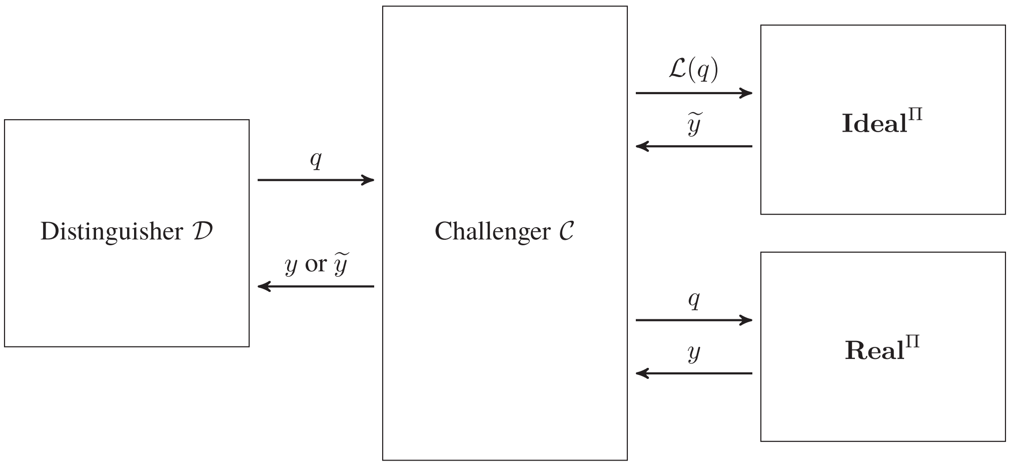


Figure 4.1: Concept of simulation-based security proofs. Note that the choice between $\mathbf{Ideal}^{\Pi}$ and $\mathbf{Real}^{\Pi}$ is made during the initialization and cannot be changed for the rest of this experiment execution.

simulator $\mathcal{S}$ for carefully quantified leakage functions $\mathcal{L}_1, \dots, \mathcal{L}_n$ such that no distinguisher $\mathcal{D}$ given either the real output y or the simulated output $\widetilde{y}$ can decide which version of the protocol has been used by the challenger (except with negligible probability).

Specifically for the simulation based proofs for protocols for privacy-preserving query processing $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{QueryToken}, \mathsf{Query})$ as given in Definition 7 in Section 2.4, the leakage functions must individually be defined for each algorithm that requires the master key K as input. The simulation of the encryption algorithm Enc has $\mathcal{L}_1(\mathbf{D})$ as input, and the simulation for token generation $\mathsf{QueryToken}$ has $\mathcal{L}_2(q)$ as input. Note that the information quantified by $\mathcal{L}_1$ and $\mathcal{L}_2$ is an *upper bound* of the encrypted database and the query token respectively. The line of arguments is as follows: The challenger has either access to the real algorithm with real input x based on cryptographic primitives as reviewed in Section 2.2, e.g., pseudorandom functions, while the constructed simulator $\mathcal{S}_i$ has only restricted access modeled by $\mathcal{L}_i(x)$ that uses a randomly sampled function. Assuming a distinguisher $\mathcal{D}$ that can distinguish between the real output y and the simulated output $\widetilde{y}$ with non-negligible probability, we use this distinguisher to attack the underlying cryptographic primitive.

This approach – known as reduction argument – is a common tool in theoretical computer science and is often used for security proofs of cryptographic algorithms and protocols. The construction of computationally secure algorithms or protocols is based on a problem that is assumed to be computationally hard. That is, no PPT algorithm can solve such a problem with more than negligible probability. This problem is then utilized to design a novel computationally secure algorithm, e.g., ciphertexts output by an encryption algorithms are instances of a computationally hard problem. Security is then defined relative to this problem and proved by a contradiction argument: given an attacker who can successfully (with more than negligible probability) attack the cryptographic algorithm can be used to construct a computationally efficient solver for the problem that is assumed to be computationally hard. Given a problem instance of this problem this instance must be transformed into a ciphertext of the encryption scheme that is “attackable” and the output of the attacker must then be transformed into a solution of the original problem instance. The concept of security proofs by reduction arguments is depicted in Figure 4.2

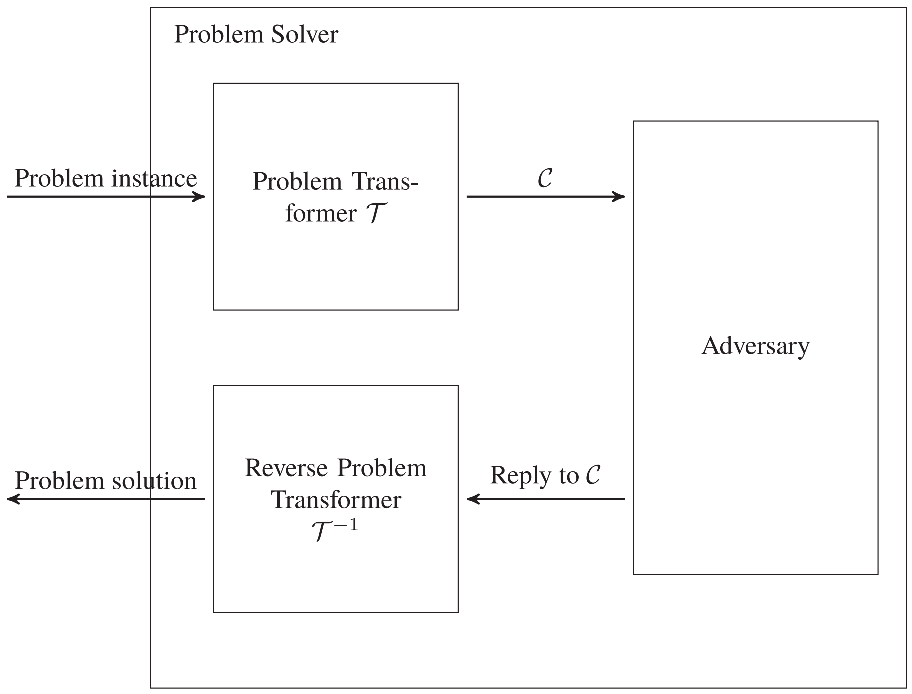


Figure 4.2: Concept of cryptographic security proofs by reduction argument.

In the case of protocols for privacy-preserving query processing, the difficulty of such simulator construction is to generate the encrypted database index $\widetilde{\mathbf{C}}$ in a way that it is *consistent* with all simulated search tokens $\widetilde{\tau}$. There exist two different kind of adversaries in the literature, namely the selective (or non-adaptive) adversary and the adaptive adversary. The selective adversary commits in the beginning of the security experiment to the complete batch of queries to be simulated as formalized in Definition 10.

Definition 10 (Selective Leakage Security)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{QueryToken}, \mathsf{Query})$ *be a privacy-preserving query processing protocol. Consider the following experiments with stateful distinguisher* $\mathcal{D}$*, stateful simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *and leakage functions* $(\mathcal{L}_1, \mathcal{L}_2)$*.*

$\mathbf{Real}^{\Pi}$: *The challenger runs* $\mathsf{Gen}(1^\lambda)$ *to get a master key* K*. Next,* $\mathcal{D}$ *outputs a dataset* $\mathbf{D}$ *and a query batch* $\mathbf{q}$ *consisting of a polynomial number of database queries. The challenger returns a ciphertext* $\mathbf{C} \leftarrow \mathsf{Enc}(K, \mathbf{D})$ *and a batch of query tokens* $\tau_{\mathbf{q}}$ *consisting of tokens* $\tau_i \leftarrow \mathsf{QueryToken}(K, q_i)$ *for each query* $q_i \in \mathbf{q}$*. The tuple* $(\mathbf{C}, \tau_{\mathbf{q}})$ *is returned to* $\mathcal{D}$*. Finally,* $\mathcal{D}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}^{\Pi}$: *The simulator sets up its internal environment. Next,* $\mathcal{D}$ *outputs a dataset* $\mathbf{D}$ *and a query batch* $\mathbf{q}$ *consisting of a polynomial number of database queries. The leakage* $\mathcal{L} = (\mathcal{L}_1(\mathbf{D}), \mathcal{L}_2(\mathbf{q}))$ *is given to the simulator* $\mathcal{S}$*. This simulator computes a simulated ciphertext* $\widetilde{\mathbf{C}} \leftarrow \mathcal{S}_1(\mathcal{L})$ *and a simulated batch of query tokens* $\widetilde{\tau_{\mathbf{q}}} \leftarrow \mathcal{S}_2(\mathcal{L})$ *consisting of simulated tokens* $\widetilde{\tau_{q_i}}$ *corresponding to query* $q_i \in \mathbf{q}$*. The tuple* $\left(\widetilde{\mathbf{C}}, \widetilde{\tau_{\mathbf{q}}}\right)$ *is returned to* $\mathcal{D}$*. Finally,* $\mathcal{D}$ *returns a bit* b *that is output by the experiment.*

In contrast, the adaptive adversary gets each intermediate result and can adjust all future queries as formalized in Definition 11. This ability of adjusting the queries increases the difficulty in creating a consistent simulator hence is harder to achieve.

Definition 11 (Adaptive Leakage Security)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{QueryToken}, \mathsf{Query})$ *be a privacy-preserving query processing protocol. Consider the following experiments with stateful distinguisher* $\mathcal{D}$*, stateful simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ *and leakage functions* $(\mathcal{L}_1, \mathcal{L}_2)$*.*

$\mathbf{Real}^{\Pi}$: *The challenger runs* $\mathsf{Gen}(1^\lambda)$ *to get a master key* K*. Next,* $\mathcal{D}$ *outputs a dataset* $\mathbf{D}$ *and receives* $\mathbf{C} \leftarrow \mathsf{Enc}(K, \mathbf{D})$*. The distinguisher* $\mathcal{D}$ *makes a polynomial number of adaptive queries and for each query* q_i $\mathcal{D}$ *receives* $\tau_{q_i} \leftarrow \mathsf{QueryToken}(K, q_i)$*. Finally,* $\mathcal{D}$ *returns a bit* b *that is output by the experiment.*

$\mathbf{Ideal}^{\Pi}$: *The simulator sets up its internal environment. Next,* $\mathcal{D}$ *outputs a dataset* $\mathbf{D}$*, where the leakage* $\mathcal{L}_1(\mathbf{D})$ *is given to the simulator and the simulated ciphertext* $\widetilde{\mathbf{C}} \leftarrow \mathcal{S}_1(\mathcal{L}_1(\mathbf{D}))$ *is returned to* $\mathcal{D}$*. The distinguisher* $\mathcal{D}$ *makes a polynomial number of adaptive queries. For each query* q_i *the simulator is given the leakage* $\mathcal{L}_2(q_i)$ *and return the appropriate query token* $\widetilde{\tau_{q_i}} \leftarrow \mathcal{S}_2(\mathcal{L}_2(q_i))$*. Finally,* $\mathcal{D}$ *returns a bit* b *that is output by the experiment.*

We say the protocol Π for privacy-preserving query processing is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against selective resp. adaptive chosen-keyword attacks if for all probabilistic polynomial-time algorithms $\mathcal{D}$ there exists a probabilistic polynomial-time simulator $\mathcal{S}$ so that advantage of $\mathcal{D}$ defined as

$$\left|\Pr\left[\mathbf{Real}^{\Pi} = 1\right] - \Pr\left[\mathbf{Ideal}^{\Pi} = 1\right]\right|$$

is negligible in λ.

This definition can be adjusted for SSE schemes supporting dynamic databases with additional algorithms providing functionality of add token and delete token generation. In this case, the add tokens and delete tokens have to be generated by the simulator with corresponding additional leakage functions that must be consistent as well.

4.1.2 Practical Security Evaluation

Although the theoretical framework discussed in the previous section offers a tool to quantify an upper bound of information leakage for specific protocols the practical consequences of this leakage is not clear.

Depending on the sensitivity of the actual data to be outsourced, the data owner might be willing to accept the induced information leakage or even increase the leakage in order to trade security for better performance or additional functionality for encrypted data. Further, the practical implications heavily depend on the type and entropy of data to be outsourced.

The constructions for exact pattern matching presented in Chapter 5, for secure joins presented in Chapter 6 and secure range queries presented in Chapter 7 achieve novel trade offs between security and performance for fundamental functionalities that have already been supported by property-preserving encryption. Since we claim increased security for these constructions compared to property-preserving encryption we prove it in the simulation-based proof framework. In contrast, the construction for privacy-preserving substring search presented in Chapter 8 is based on the functionality of secure range queries. Assuming a natural text of a spoken language (e.g. English) to be outsourced, the entropy of such data is well known, therefore we examine the practical security consequences for this construction using property-preserving encryption for additional search performance. That is, we have applied our construction and try to extract as much information as possible applying the best known attacks on property-preserving encryption. We refer to Chapter 8 for a thorough discussion and a comprehensive description of the actual attack.

4.2 Performance Assessment Methodology

The second dimension we evaluate our protocols is performance, hence we give a brief description of the methodology for performance assessment in this section. While we focus on the running time of a protocol this methodology could be extended to other key performance indicators such as memory consumption or required network bandwidth. We assume running time of the protocol as the most important parameter for acceptance of the proposed solution by the end user. Therefore we measure the time period between creation of the search token until receiving the corresponding query result as the overall running time of one protocol run corresponding to the time the end users submits the query until the result retrieval. Further, we investigate this overall running time in more details and give micro benchmarks for critical operations, e.g. query token generation, in order to identify bottle necks or distinguish between processing time on the client's device and the server environment. Specific details and particular considerations depend on each construction individually and are discussed in the corresponding chapter.

4.2.1 Theoretical Runtime Analyses

In order to get an "awareness" about the magnitude of runtime performance, we analyze the costs of an algorithm using the big O notation. Big O notation gives an upper bound for the asymptotic performance of an algorithm, i.e. when the input reaches or exceeds a specific value and allows to classify algorithms with respect to their runtime as already done in Section 2.1 for PPT algorithms. Formally, big O notation is stated as follows [47]:

Definition 12 (Big O Notation)**.** *Let $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$. We say $f(x)$ is in $O(g(x))$ if there exists a constant $c > 0$ and a threshold $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \in \mathbb{N}$ with $n > n_0$. We write $f(x) = O(g(x))$ for a function $f(x)$ in $O(g(x))$.*

The big O notation is transitive by definition: given three functions $f(x)$, $g(x)$ and $h(x)$ with $f(x)$ is in $O(g(x))$ and $g(x)$ is in $O(h(x))$ it holds that $f(x)$ is in $O(h(x))$. Further, the most important classes for algorithms are enumerated in the following in ascending order with faster growing runtime: constant runtime denoted as $O(1)$, logarithmic runtime denoted as $O(\log n)$, linear runtime denoted as $O(n)$, polynomial runtime denoted as $O(n^c)$ for a constant[1] $c \in \mathbb{N}$ and exponential runtime denoted as $O(c^n)$ for a constant $c \in \mathbb{N}$. Any search index for fast query processing should decrease search complexity in way such that it is smaller than linear in the database table size. Note that these classes express the worst case runtime for an algorithm. While no input for the algorithm results in a runtime larger than this bound, there might be a great amount of inputs that result in strictly lower runtime.

In this thesis we design systems that are queried repeatedly, therefore it is natural to ask for the runtime of a *sequence of queries*. One could simply multiply the worst case runtime expressed in big O notation with the number of queries. Given a protocol with only a low number of possible inputs with such worst case runtime, however, this assessment would result in a too pessimistic (i.e. too high) runtime estimation, thus we follow the approach of amortized analysis as introduced by Tarjan [141]. While amortized analysis does not state explicit performance numbers for one specific query it gives a better understanding of the average performance properties. We refer to the textbook by Cormen et al. for additional details [47].

4.2.2 Practical Runtime Evaluation

Big O notation and amortized analysis are vital tools for estimation of a program's runtime as function of the input size. Nevertheless, practical evaluation on real input data is still vital for testing the practical feasibility of an approach and allows comprehensive comparison of different approaches for specific use cases. Especially the impact of large constants are omitted in big O notation and neither are considered in amortized runtime analysis as discussed in the previous section. As a consequence we have implemented each construction in this thesis and evaluated it in a real-world environment, that is, we model a real database, encrypt it using the proposed construction and ask for realistic privacy-preserving query sequences. This practical evaluation is performed on a testbed corresponding "state-of-the-art" environment and is specified in full details in each chapter.

[1] With $c = 1$ we get linear runtime.

5 Exact Keyword Matching

In this chapter we present protocols for exact keyword queries on encrypted data. Such functionality enables the database client to delegate exact keyword filtering to the untrusted database server. The DBMS on this server is then able to filter for all encrypted values that match the keyword specified by the client. In the following Section 5.1 we give an introduction, discuss the problem in more details and give a general interface that offers the functionality for exact pattern matching on encrypted data with the option to add and delete data entries securely. An overview of related work addressing this type of query is presented in Section 5.2. In Section 5.3 we describe the abstract idea of our solution from a high-level perspective and present a concrete construction. Based on this concrete construction a detailed evaluation is given in Section 5.4 regarding both formal security and performance. Performance evaluation is done in two ways, that is, theoretical runtime numbers are given together with practical benchmark results. A variation of this construction is given in Section 5.5 trading communication overhead for additional client storage. Finally, Section 5.6 provides a summary of this chapter. The content of this chapter has been published in joint work with Florian Kerschbaum at CCS 2014:

- Hahn, Florian ; Kerschbaum, Florian: Searchable Encryption with Secure and Efficient Updates. In: *Proceedings of the Conference on Computer and Communications Security*, 2014 (CCS)

5.1 Introduction

One functionality often required for database systems is the possibility of filtering data that matches exactly a string or equals a value specified by the user during runtime. For example, assume a database containing employees and their individual salary as sketched in Table 5.1 the client wishes to outsource in encrypted form. Later, the client wishes to query the salary for one specific employee defined by the exactly stated

EmpID	**Name**	**Salary**
1	Harry	4000
2	Sally	7000
3	Harry	5000
4	George	3500

Table 5.1: Example of plain database table *Emp*

name, e.g. the corresponding SQL query retrieving Sally’s salary might be stated as given in Listing 5.1

```
SELECT * FROM Emp WHERE Name = ‘‘Sally’’
```

Listing 5.1: Example SQL query for exact keyword matching

yielding the result given in Table 5.2 on the client side. Previous solutions adapted in real-world prototypes for encrypted databases realize this functionality based on deterministic encryption [123]. The query transformation is then simply the deterministic encryption of the filtering attribute, that is, the constant to be

EmpID	Name	Salary
2	Sally	7000

Table 5.2: Plaintext result after query execution for table *Emp*

searched for is replaced by its deterministic encryption, e.g. $\mathsf{Enc}^{\mathrm{Det}}$(Sally). However, as discussed previously in Section 3.2 and its Subsection 3.2.3 the initial deterministically encrypted database is vulnerable to quite simple attacks such as frequency analysis. The constructions presented in this chapter mitigate this attack vector while providing the functionality of exact pattern matching. Particularly, the equality relation between outsourced values is obfuscated by initial randomized encryption, and the search for a specified value is enabled using an additional search token created by the client and released only if required, i.e. actually queried by the client.

We aim for our practical searchable symmetric encryption (SSE) scheme to be efficient, dynamic and secure. By efficiency we mean sublinear search time and this is achieved using an (inverted) index the server constructs incrementally based on released information. A dynamic searchable encryption scheme provides the client the ability to modify the database by adding new data or deleting indexed data. These privacy preserving index updates raise major problems either performance wise or security wise (cf. Section 5.2 for a detailed discussion of related work).

We present a dynamic searchable encryption scheme with secure and efficient updates. Even under updates, our schemes leaks no more than what can be inferred from the search tokens. Our index size is linear in the number of keywords, hence asymptotically optimal. While the time for the first initial search of a specific keyword is linear in the number of indexed keywords, we show that it amortizes over multiple searches and is hence practical.

5.1.1 Framework

We describe the general interface addressing the scenario introduced previously. Following the general approach of symmetric searchable encryption we discuss our scheme based on files to be outsourced instead of individual database records. We sketch the ideas how the application of encrypted files can be transferred to the application of encrypted databases after introducing the notation.

We assume each file f having a unique file identifier $ID(f)$, each file consists of words that is $f = (w_1, \dots, w_{\mathtt{len}(f)})$ with $w_i \in \{0,1\}^*$. For a fileset $\mathbf{f}$ we denote $\mathtt{len}(\mathbf{f})$ as the number of individual files in $\mathbf{f}$. Given a keyword w we write $\mathbf{f}_w$ as the subset of all files $\mathbf{f}$ containing w. In addition, the set of all file identifiers pointing to files that contain this keyword w is denoted by $\mathbf{I}_w$. More formally it is defined as $\mathbf{I}_w = \{ID(f_i) \mid f_i \in \mathbf{f}_w\}$.

In order to apply this approach on searchable database encryption, one replaces each file with one row in a table, where the keyword of a file is the specific value for an attribute stored in one specific database column; the complete set of all indexed files then corresponds to the complete database table. The result $\mathbf{f}_w$ for searching a keyword w in fileset $\mathbf{f}$ transferred to the application of encrypted databases then corresponds to the result set $\mathbf{D}[q]$ for a database query q with specified search term w.

Compared to other dynamic searchable encryption schemes, the scheme presented in this work does not offer a dedicated operation for initial outsourcing a set of files but starts with an empty search index γ. By executing the Add algorithm, the service provider's encrypted search index γ is updated by file specific add tokens α_f for each file f to be added. This add token α_f is generated by the client calling $\mathsf{AddToken}$. In order to perform a search query for keyword w, the client generates a deterministic search token τ_w calling $\mathsf{SearchToken}$. This search token τ_w is handed to the service provider initiating the actual search operation on the server calling Search with input of the search index γ and τ_w. For simplicity of the exposition we assume that all generated search tokens are given to the service provider, i.e. if a search token has been created by the client it is also transferred to service provider and the service provider gains knowledge of that deterministic search token. Finally, in order to delete a file f the client creates a delete token δ_f for file f.

Definition 13 (Securely Updating Index-Based Searchable Encryption Scheme)**.** *A securely updating index-based searchable encryption scheme is a tuple of seven (possibly probabilistic) polynomial-time algorithms* $\mathrm{SUISE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SearchToken}, \mathsf{Search}, \mathsf{AddToken}, \mathsf{Add}, \mathsf{DeleteToken}, \mathsf{Delete})$ *such that:*

$(K, \gamma, \sigma) \leftarrow \mathsf{Gen}(1^\lambda)$ *is a probabilistic algorithm that takes as input a security parameter λ and outputs a master key K, a (still empty) search index γ and a (still empty) search history σ.*

$(\sigma', \tau_w) \leftarrow \mathsf{SearchToken}(K, w, \sigma)$ *is a (possibly probabilistic) algorithm that takes as input a master key K, a keyword w and search history σ. It outputs an updated search history σ' and a search token τ_w.*

$(\boldsymbol{I}_w, \gamma') \leftarrow \mathsf{Search}(\tau_w, \gamma)$ *is a deterministic algorithm that takes as input a search token τ_w and a search index γ. It outputs a sequence of identifiers $\boldsymbol{I}_w$ and an updated search index γ'.*

$\alpha_f \leftarrow \mathsf{AddToken}(K, f, \sigma)$ *is a (possibly probabilistic) algorithm that takes as input a master key K, a file f and a search history σ. It outputs an add token α_f.*

$\gamma' \leftarrow \mathsf{Add}(\alpha_f, \gamma)$ *is a deterministic algorithm that takes as input an add token α_f and a search index γ . It outputs an updated search index γ'.*

$\delta_f \leftarrow \mathsf{DeleteToken}(K, ID(f))$ *is a (possibly probabilistic) algorithm that takes as input a master key K and an identifier $ID(f)$ of the file that shall be removed. It outputs a delete token δ_f.*

$\gamma' \leftarrow \mathsf{Delete}(\delta_f, \gamma)$ *is a deterministic algorithm that takes as input a delete token δ_f for the file that shall be removed and a search index γ. It outputs an updated search index γ'.*

5.2 Related Work

An overview of general related work on searchable symmetric encryption is given in Section 3.3 and encrypted databases in Section 3.1. In this section we review related work particularly applicable to the functionality of exact pattern matching on encrypted and dynamic data.

Song et al. introduced searchable encryption for equality checks in their work [137] with no search index. As a result, each encrypted value must be checked separately for a given search token, requiring linear search time in the number of files times the number of keywords per file. At the same time, dynamic datasets are easy to support for this scheme as no search index must be updated.

A number of works investigated efficiency increases for symmetric searchable encryption for equality search. Goh first proposed the use of indexes [64] using Bloom filters [20] per indexed file reducing the search to be constant for each encrypted file. These Bloom filters for individual files are additionally salted with their public file identifiers resulting in different search indexes even for two files containing the same words but with different identifiers. This index leads to a sublinear search time in the overall number of indexed words and a linear search time in size of the file collection, however, false positives might be induced due to the application of Bloom filters. Chang and Mitzenmacher followed Goh's approach and created one search index per document but thwarted false positive results. Particularly, their approach is based on a prebuilt dictionary for each indexed file: assuming m as the universe size of potential keywords, such dictionary for a specific file is represented by an m-bit array, where bit i is set to 1 if word w_i appears in the file. Before outsourcing, these dictionaries are blinded with pseudorandom functions, while searching for keyword w_i is then performed by lifting this blinding for bit i for each file individual dictionary. Curtmola et al. have been the first to use inverted indexes [49] achieving asymptotically optimal search time. These inverted indexes are implemented using an oblivious linked list. The created search token unveils the entry point into this linked listed together with the possibility traversing all relevant entries in the linked list providing the complete result set. While constructing this index can be performed on the plaintext file collection, modifying it without leakage is difficult.

The first *dynamic* searchable symmetric encryption scheme with asymptotically optimal search time has been proposed by Kamara et al. [84]. In their work, Kamara et al. distinguish between initially indexed files and files added after the initial outsourcing step. While the initially indexed files leak the same information as they would do using Curtmola's approach [49], the security of all files added after the initial outsourcing step degenerates to deterministic encryption. Later, Kamara and Papamanthou addressed this information leakage with a tree-based multi-map data structure, however, the index is of size of the number of documents times the number of keyword [83]. An alternative approach for dynamic searchable symmetric encryption has been published by Cash et al. [38]. They create a separate search dictionary indexing all files added after the initial indexing step. This dictionary is complemented with keyword specific labels for all keywords contained in the new file to be added. These labels are computed for each individual keyword together with a keyword specific counter, i.e. how often a file with this specific keyword has already been added. While considering this keyword specific counter into label generation hides the keyword frequency for all files added after the initial indexing step, the client is required to know all these keyword specific counters. Further, deleted files are stored at an extra revocation list, hence the index size *increases* with file deletion. An alternative dynamic SSE scheme based on a construction named Blind Storage can be deployed on servers that do not provide additional functionality besides uploading and downloading files [116]. Hence, Blind Storage is suitable for file hosting services, like e.g. DropBox or Microsoft's OneDrive. However, this restricted functionality is not given in our use case of encrypted databases where we have the capability to modify the DBMS.

Stefanov et al. [139] have proposed the first scheme with a security property they call forward privacy for a dynamic SSE scheme. By forward privacy they mean the following: if a keyword w has been searched for and later a new file is added also containing w, the server does not learn that this already queried keyword w is contained in that file. They achieve forward privacy by an index of document-keyword pairs stored in a hierarchical structure applying techniques known from the constructions of oblivious RAM (cf. Section 3.5.3). This hierarchical index is then rebuilt periodically by the client to keep its logarithmic structure sound. A later result published by Bost [31] also achieved forward privacy. In contrast to Stefanov's approach, this construction solely relies on pseudorandom functions but does not rely on constructions inspired by ORAM.

The dynamic searchable symmetric encryption scheme proposed in this work has asymptotically optimal search time, asymptotically optimal storage cost and no additional leakage on updates.

5.3 Implementation

The main idea of our construction is based on the observation that common searchable symmetric encryption schemes yield – with a few exceptions like $\Sigma o\varphi o\varsigma$ [31] – deterministic search tokens (also referred to as the search pattern). Combining these deterministic search tokens with the unveiled result set (also referred to as access pattern) after a successful search execution allows us to iteratively construct a result cache for search executions on the server side. While the first search for a specific keyword requires runtime that is linear in the number of indexed files, all subsequent search operations for this same keyword is then optimal. Further, the result cache construction is solely based on the search pattern and the access pattern. Keeping this cache consistent even when adding or deleting files without additional leakage requires a careful protocol design described in the following.

For the implementation we use several data structures such as lists and (chained) hash tables described in the following. A hash table T stores values v associated with keys k, written as $T[k] = v$. We write $v \in T$ if there is a key k so that $T[k] = v$. For our implementation it is crucial that it is feasible to access a value v with corresponding key k stored in a hash table in constant time. If the values stored in the hash table are lists, we call it a chained hash table.

Assume a pseudorandom number generator (PRNG) G that outputs random numbers with bit length λ. Given a pseudorandom functions $F : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$, and a random oracle $H : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ as introduced in Section 2.2 we construct dynamic SSE scheme SUISE = (Gen, SearchToken, Search, AddToken, Add, DeleteToken, Delete) as follows:

$(K, \gamma, \sigma) \leftarrow \mathsf{Gen}(1^\lambda)$**:** Sample a λ-bit strings $K \xleftarrow{\$} \{0,1\}^\lambda$. In addition, create two empty chained hash tables γ_f, γ_w and an empty set σ. Output (K, γ, σ), with $\gamma = (\gamma_f, \gamma_w)$.

$(\tau_w, \sigma') \leftarrow \mathsf{SearchToken}(K, w, \sigma)$**:** Calculate $F_K(w) = \tau_w$, set $\sigma' = \sigma \cup \{\tau_w\}$ and output (τ_w, σ').

$(\mathbf{I}_w, \gamma') \leftarrow \mathsf{Search}(\tau_w, \gamma)$**:** Parse search index $\gamma = (\gamma_w, \gamma_f)$ and check if there is an entry for τ_w in γ_w.

- If yes, then set $\mathbf{I}_w = \gamma_w[\tau_w]$ and $\gamma'_w = \gamma_w$.
- Otherwise create an empty list $\mathbf{I}_w$ and do for every $\overline{c} \in \gamma_f$:
  1. for every $c_i \in \overline{c}$ that is $i \in [1, \mathtt{len}(\overline{c})]$, set $c_i = l_i || r_i$ and check if $H_{\tau_w}(r_i) = l_i$. If yes then insert $ID(f)$ that corresponds to $\overline{c}$ into $\mathbf{I}_w$.

  Update γ'_w by creating an entry $\gamma_w[\tau_w] = \mathbf{I}_w$

Output $\mathbf{I}_w$ and (an updated version of) $\gamma' = (\gamma'_w, \gamma_f)$.

$\alpha_f \leftarrow \mathsf{AddToken}(K, f, \sigma)$**:** For file f that consists of a sequence of words create a list $\overline{f}$ of *unique* words $f \supseteq \overline{f} = (w_1, \ldots, w_{\mathtt{len}(\overline{f})})$. Generate a sequence of pseudorandom values $s_1, \ldots s_{\mathtt{len}(\overline{f})}$ with PRNG G and create an empty list $\mathbf{x}$. For every word $w_i \in \overline{f}$ do the following:

1. compute the corresponding search token $\tau_{w_i} = F_K(w_i)$
2. if this search token was used for a previous search, i.e. if $\tau_{w_i} \in \sigma$, add τ_{w_i} to $\mathbf{x}$.
3. set $c_i = H_{\tau_{w_i}}(s_i) || s_i$

Now sort $\overline{c} = (c_1, \ldots, c_{\mathtt{len}(\overline{f})})$ in lexicographic order, set $\alpha_f = (ID(f), \overline{c}, \mathbf{x})$ and output α_f.

$\gamma' \leftarrow \mathsf{Add}(\alpha_f, \gamma)$**:** Parse $\alpha_f = (ID(f), \overline{c}, \mathbf{x})$, $\gamma = (\gamma_w, \gamma_f)$ and set $\gamma_f[ID(f)] = \overline{c}$. In addition, for every $x_i \in \mathbf{x}$ add $ID(f)$ to $\gamma_w[x_i]$. Output the updated version $\gamma' = (\gamma_w, \gamma_f)$.

$\delta_f \leftarrow \mathsf{DeleteToken}(K, f)$**:** Output file identifier $ID(f)$ as delete token.

$\gamma' \leftarrow \mathsf{Delete}(\delta_f, \gamma)$**:** Parse $\gamma = (\gamma_w, \gamma_f)$ and $\delta_f = ID(f)$, check for every list $\mathbf{e}$ saved in γ_w if $ID(f) \in \mathbf{e}$ and remove $ID(f)$ in this case from $\mathbf{e}$. Remove $\gamma_f[ID(f)]$ from γ_f. Output an updated search index $\gamma' = (\gamma_w, \gamma_f)$.

5.4 Evaluation

In this section we evaluate the theoretical security properties in a formal framework. Further, we examine the performance both from a theoretical point of view but also with a practical benchmark based on an implementation of our protocol.

5.4.1 Formal Security Proof

We prove security of this construction following the simulation based security proofs discussed in Section 4.1.1. First, we adapt the general definition given previously in a way fitting to our specific construction. The leakage given to simulator $\mathcal{S}$ is required for simulating the add token generation and the search token generation. Since the delete token is neither based on sensitive data but only on the identifier of the file to be deleted, nor is it based on the master key, we omit the leakage for this token. We prove security against an adaptive adversary as defined in the following.

Definition 14. *Let* SUISE $= (\mathsf{Gen}, \mathsf{Enc}, \mathsf{SearchToken}, \mathsf{Search}, \mathsf{AddToken}, \mathsf{Add}, \mathsf{DeleteToken}, \mathsf{Delete})$ *be a securely updating index-based searchable encryption scheme and λ the security parameter. Consider the following experiments with stateful adversary $\mathcal{A}$, stateful simulator $\mathcal{S}$ and stateful leakage functions $\mathcal{L}_1, \mathcal{L}_2$.*

$\mathbf{Real}^{\mathrm{SUISE}}(\lambda)$: *the challenger runs $\mathsf{Gen}(1^\lambda)$ to get the tuple (K, γ, σ). The adversary $\mathcal{A}$ makes a polynomial number of adaptive queries $q \in \{w, f\}$ and for each query q the challenger generates either a search token $\tau_w \leftarrow \mathsf{SearchToken}(K, w, \sigma)$ or an add token $\alpha_f \leftarrow \mathsf{AddToken}(K, f, \sigma)$. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.*

$\mathbf{Ideal}^{\mathrm{SUISE}}(\lambda)$: *the simulator sets up its internal environment. The adversary $\mathcal{A}$ makes a polynomial number of adaptive queries $q \in \{w, f\}$ and for each query q the simulator is given the appropriate leakage, i.e. either given $\mathcal{L}_1(\boldsymbol{f}, w)$ or $\mathcal{L}_2(\boldsymbol{f}, f)$. $\mathcal{S}$ returns the appropriate simulated token $\widetilde{\tau_w}$, $\widetilde{\alpha_f}$. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.*

We say SUISE *is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive dynamic chosen-keyword attacks if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a non-uniform probabilistic polynomial-time simulator $\mathcal{S}$ so that advantage of $\mathcal{A}$ defined as*

$$\left|\Pr\left[\mathbf{Real}^{\mathrm{SUISE}}(\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}^{\mathrm{SUISE}}(\lambda) = 1\right]\right|$$

is negligible in λ.

As discussed in the previous Chapter 4, some operations leak particular information to the service provider. In detail, we use the two leakage functions $\mathcal{L}_1, \mathcal{L}_2$ defined as follows:

$$\begin{aligned}\mathcal{L}_1(\mathbf{f}, w) =& (\mathtt{acc}_t(w), ID(w))\\ \mathcal{L}_2(\mathbf{f}, f) =& (ID(f), \mathtt{len}\left(\overline{f}\right), \sigma_t(\overline{f}))\end{aligned}$$

where $\mathtt{acc}_t(w)$ is the access pattern at time t defined as set $\{ID(f_i) \mid w \in f_i \text{ and } f_i \in \mathbf{f}\}$, $\overline{f}$ is the set of unique words in file f, and $\sigma_t(\overline{f})$ is the set of IDs of all searched words until time t that also appear in $\overline{f}$.

Search tokens are deterministic, that is, from a client's perspective a search token for the same word w requested several times is always the same; from the (potential malicious) server's perspective given the same search token several times , the same word w is always queried. In other words, an adversary is able to link generated (repeated) search tokens with queries, although the attacker is not able to extract the plain query. This knowledge is denoted by $ID(w)$ in the defined leakage functions. Note that this possibility of re-identification of already queried keywords corresponds to the security property provided by deterministic encryption. Now we are ready to proof the following theorem:

Theorem 1. *If the used function F is a pseudorandom function and function G is a pseudorandom generator and H is a programmable random oracle, then* SUISE *as described in Section 5.3 is $(\mathcal{L}_1, \mathcal{L}_2)$-secure against adaptive dynamic chosen-keyword attacks in the random oracle model.*

Proof. We describe a polynomial time simulator $\mathcal{S}$ for which the advantage of any PPT adversary $\mathcal{A}$ to distinguish between the output of $\mathbf{Real}^{\text{SUISE}}$ and $\mathbf{Ideal}^{\text{SUISE}}$ is negligible. Our simulator adaptively simulates a search index $\widetilde{\gamma}$ with the additional information given by the leakage functions.

Setting up the environment: $\mathcal{S}$ creates

- An empty list $\widetilde{\sigma}$ as simulated search history,
- an empty simulated search index $\widetilde{\gamma} = (\widetilde{\gamma_w}, \widetilde{\gamma_f})$ consisting of two empty hash tables
- an empty dictionary ρ to keep track of queries to the random oracle.
- A chained hash table P is used to keep track of tuples $(j, \widetilde{\tau})$ consisting of previously simulated search positions and simulated search tokens for each individual file. In more details, an entry of P consists of a linked list, we denote P_{f_i} as the linked list for file f_i, that is stored at hash table entry $P[ID(f_i)]$.
- A empty hash table T is created to keep track of the assignment of simulated search tokens $\widetilde{\tau_w}$ for word w with $ID(w)$ and
- A empty hash table A is created to keep track of simulated add tokens $\widetilde{\alpha_f}$ for already added files with $ID(f)$.

Simulating search tokens $\widetilde{\tau}$: Given leakage

$$\mathcal{L}_1(\mathbf{f}, w) = (\mathtt{acc}_t(w), ID(w)),$$

the simulator checks if $ID(w)$ is in T, i.e. if a search token for this word has been queried before.

- If this is the case, $\mathcal{S}$ outputs $T[ID(w)]$.
- Otherwise, a random bit string $\widetilde{\tau} \xleftarrow{\$} \{0,1\}^\lambda$ is sampled and stored at $T[ID(w)]$ and added to $\widetilde{\sigma}$. For every $ID(f_i) \in \mathtt{acc}_t(w)$ the simulator sets J_{f_i} as the set of already used positions; more formally set $J_{f_i} = \{j_l \mid (j_l, \tau_l) \in P_{f_i} \text{ with } 0 \leq l \leq |P_{f_i}|\}$. Then the simulator chooses an unused random position $j_i \xleftarrow{\$} [1, |\widetilde{\gamma_f}[ID(f_i)]|] \setminus J_{f_i}$ and adds the tuple $(j_i, \widetilde{\tau})$ to the list P_{f_i} of used positions for file f_i. Finally, $\mathcal{S}$ outputs $\widetilde{\tau}$.

Simulating add tokens $\widetilde{\alpha}$: Given leakage

$$\mathcal{L}_2(\mathbf{f}, f) = \left(ID(f), \mathtt{len}\left(\overline{f}\right), \sigma_t(\overline{f})\right),$$

$\mathcal{S}$ checks if there is an entry at $A[ID(f)]$, i.e. if an add token for file f with $ID(f)$ has been simulated before.

- If an add token $\widetilde{\alpha_f}$ for file f with $ID(f)$ has been requested before, the simulator outputs $A[ID(f)]$.

- Else, this file has not been added before and the simulator chooses for every $i \in [1, \texttt{len}\left(\overline{f}\right)]$ a random bit string $\widetilde{s_i} \xleftarrow{\$} \{0,1\}^{2\lambda}$, and sorts this generated set $(\widetilde{s_1}, \ldots, \widetilde{s}_{\texttt{len}(\overline{f})})$ in lexicographic order to get $\widetilde{s}$ and stores this at $\widetilde{\gamma_f}[ID(f)]$. In addition, an empty list $\widetilde{\mathbf{x}}$ is created and for every $ID(w) \in \sigma_t(\overline{f})$ the token $\widetilde{\tau_w} = T[ID(w)]$ is looked up, added to $\widetilde{\mathbf{x}}$ and $ID(f)$ is added to $\widetilde{\gamma_w}[\widetilde{\tau_w}]$. $\mathcal{S}$ creates a temporary set J and for all $l \in [1, \texttt{len}(\widetilde{\mathbf{x}})]$ a random fake position $j_l \xleftarrow{\$} [1, |\widetilde{\gamma_f}[ID(f_i)]|] \setminus J$ is sampled and added to J. This position j_l is marked as used in the list of used search positions by adding the tuple $(j_l, \widetilde{\tau_l})$ to P_f in the chained hash table P, where $\widetilde{\tau_l} = \widetilde{\mathbf{x}}[l]$. Finally, $\mathcal{S}$ outputs $\widetilde{\alpha} = (ID(f), \widetilde{s}, \widetilde{\mathbf{x}})$ and stores this simulated token at $A[ID(f)]$.

Answering random oracle queries: Given query (k, r), the simulator checks if this query was submitted before, i.e. if there is an entry $l = \rho[k||r]$.

- If this is the case, set $l = \rho[k||r]$.
- Otherwise, $\mathcal{S}$ checks if key k is linked with some $ID(w)$, i.e. if there is an entry $k = T[ID(w)]$ for some $ID(w)$. If there is no used search token, a random bit string $l \xleftarrow{\$} \{0,1\}^\lambda$ is sampled and $\rho[k||r] = l$ is set to stay consistent for future queries.

  Else, k is linked with some $ID(w)$, for every $ID(f_i) \in \widetilde{\gamma_w}[k]$ the simulator looks up the tuple $(j, k') \in P_{f_i}$ where $k' = k$. Then the value $\widetilde{s_j} \in \{0,1\}^{2\lambda}$ is set as the j-th entry of $\widetilde{s} = \widetilde{\gamma_f}[ID(f_i)]$ and divided in two λ-bit strings $l'||r' = \widetilde{s_j}$.

  - $\mathcal{S}$ checks if $r' = r$, sets $l = l'$ in this case and stores l at $\rho[k||r]$.
  - If there was no fitting r' for any $ID(f_i) \in \widetilde{\gamma_w}[k]$, a random $l \xleftarrow{\$} \{0,1\}^\lambda$ is sampled and stored at $\rho[k||r]$ to stay consistent for future queries.

Finally, l is returned.

The indistinguishability of a simulated search token $\widetilde{\tau}$ and a real search token τ follows from the pseudorandomness of F. Also, the indistinguishability of a simulated search history $\widetilde{\sigma}$ and a real search history σ follows from the pseudorandomness of F. The indistinguishability of a simulated add token $\widetilde{\alpha}$, especially of $\widetilde{s}, \widetilde{\mathbf{x}}$, and a real add token α follows from the pseudorandomness of G and F. Since we choose the output of our simulated random oracle H either totally random or out of a predefined domain, that itself is generated in a random way, our random oracle is indistinguishable from a pseudo-random function. □

Our leakage definition for the add operation still includes the identifier of the file $ID(f)$) and the number of unique keywords in that file $len(\overline{f})$. One can hide both by adding a level of indirection. First encrypt each file f, resulting in the identifier $ID(f)$. Then for each $w_i \in \overline{f}$ encrypt file $f' = \{ID(f)\}$ resulting in unique identifier $ID(f')$. One can now create the add token for $ID(f')$ and w_i. A simulator for the add token operation is simple to derive from our simulator: $ID(f')$ – which is unique in the system – replaces $ID(f)$ and is leaked instead, but $len(\overline{f'}) = 1$ and can hence be omitted.

5.4.2 Amortized Runtime Analysis

In our algorithm the first search for a keyword requires a linear scan, but subsequent searches are constant in time[1] due to the chained hash table. Hence, the initial overhead amortizes and we reach asymptotically optimal search time for long-running systems. In the following we analyze the required number of search operations until we reach this optimum.

[1] This is a theoretical simplification. From a practical point of view there might be side effects slightly increasing the access time with increased number of stored elements, e.g. due to memory paging.

Let n be the number of unique keywords stored in the ciphertexts; let m be the total number of stored keywords in the ciphertext. Once we have created an index entry for a keyword, our search complexity is m/n: We have a constant lookup in the hash table and return m/n ciphertexts on average. An initial search can take up to m search (lookup) operations and there can be at most n of those. Hence, the initial effort is (upper) bounded by mn. We are interested in the number N of searches such that the amortized cost becomes optimal. Since we need to return at least m/n entries, this is the lower bound optimum. The cost is asymptotically optimal, if there exists a constant c, such that the cost is at most the optimum times c. The amortized cost is the cost for initial searches (mn) divided by the number of searches plus the cost for one subsequent search:

$$\frac{mn}{N} + \frac{m}{n} \leq c\frac{m}{n}.$$

We conclude from this formula that we need at least $N \geq n^2$ searches until our cost is asymptotically optimal. The constant $c = 2$ is low and we need at most 0.5 cryptographic hash operation on the server on average. Read (search) requests dominate many systems like databases, such that this number can be quickly reached in practice.

5.4.3 Practical Benchmark

The following experiments have been implemented in Java 7. Either operations performed by the server, or operations performed by the client, have been executed on an Intel Xeon 1230v3 CPU 3.30GHz with 8GB RAM running Windows 8. To minimize I/O access time all files used in our simulations are loaded into main memory before starting measurements.

For the implementation of our keyed random function F and our random oracle H we use the implementation of HMAC-SHA-1 that is contained in the default Java library. Also contained in the default Java library, we use SHA1PRNG as pseudorandom number generator G that outputs a pseudorandom number with length of 160 bits.

On the client's side

One main argument for outsourcing data is the use of slow and weak hardware on client's side. To show our SUISE scheme feasible for this scenario, we simulated the creation of add tokens and search tokens for $250,000$ random words per run. We repeated each creation run 100 times and present the average of these 100 runs for creating one token. We simulated both versions for storing the search history that is storing it on the client or storing it on the server.

The cost for creating search tokens depends on the cost of generating one HMAC-SHA-1 mainly. Creating an add token without checking the search history (so let the server check for indexed words used in previous search queries) needs two HMAC operations and one random number. If the client has to check the search history, the cost for add token generation also depends on the size of search history. For our simulation we filled the search history with 0, $100,000$ and $1,000,000$ unique words. By using Java's HashSet as data structure implementing the search history, the lookup time can be minimized. Remember that the search history contains *unique* search words used before and stays quite small (see Figure 5.2) in relation to the number of search queries.

On the server's side

All our simulations ran single threaded, but can easily be executed in parallel. By dividing the search index γ_f into subsets and search these subsets on different cores it is possible to speed up searches for search

operation	$\|\sigma\|$	time $[\mu s]$
SearchToken	-	1.14
AddToken	0	4.77
AddToken	10^5	5.06
AddToken	10^6	5.47

Figure 5.1: Average duration for creating one token for one word.

tokens that were not searched before. In our test scenario all operations ran on one machine so we were able to ignore latency through network transfers that may occur in practical application.

We omit benchmarks for Add and Delete since the runtime of these tasks depend on the chosen methods for creating indexes and updating these indexes. In addition, one can interpret these operations as storing, accessing, adding and deleting plaintext in an efficient way, because no cryptographic primitives are used there. Either cryptographic primitives are used on the client before and benchmarked there (e.g. creating add tokens) or are not needed at all.

So, the runtime of Search is discussed in the following. For our experiments we added 50 ebooks downloaded from Project Gutenberg[2]. Before adding these files, all words were transformed to lower case and punctuation was removed. Our complete fileset $\mathbf{f}$ consisted of $3,654,417$ words separated by whitespaces after this transformation. Removing words that appear multiple times in one file resulted in a filest containing $337,724$ words so our index γ_f had size $337,724$. Altogether $95,465$ words were indexed, i.e. $95,465$ different search tokens would result in at least one positive match. To simulate realistic search queries we use a list of word frequencies[3] which represent real world search queries but omitted the first 100 entries that mainly contained pronouns and prepositions. This word frequency list contains about $400,000$ words and our search words are chosen randomly weighted according to their frequency.

In order to benchmark search operations Search at the service provider we generate 5000 random search tokens using the probability distribution explained above. The mean search time for these 5000 search tokens results in one measurement point. A complete benchmark run consists of $75,000$ search queries, i.e. 15 values are measured per run. In total, we repeated these benchmark runs 10 times and plotted the average and the error bars provide the standard derivation of these 10 runs. The average time for an initial, linear search was 414.38 ms. The average time for a second, constant time search was 0.01 ms.

Figure 5.2 shows the size of the search history over the time of the experiment. It also depicts the decreasing number of newly generated search tokens that were not contained in the search history before. Denoted by the white part, every bar represents the size of search history σ before the 5000 search queries. The gray part[4] represents the amount of queried search tokens that were not known before these 5000 search queries. So, combining the white and the gray part shows the size of search history σ after executing these 5000 queries. Figure 5.2 demonstrates the decreasing amount of newly generated search tokens with increasing amount of total search queries. On the one hand, due to this effect the majority of our randomized add tokens remain randomized. At the end of the experiment less than 16% of the unique keywords are in the inverted index γ_w and hence encrypted deterministically. On the other hand, this results in decreasing search time, because already searched tokens can be looked up in this inverted index.

This effect is presented in Figure 5.3 in more detail. It shows the mean search time in ms for blocks of 5000 searches over the time of the experiment. One bar at position i in Figure 5.2 can be linked with the

[2] `http://www.gutenberg.org/`
[3] Taken from `http://invokeit.wordpress.com/frequency-word-lists/`
[4] The distribution of these still unknown search tokens follows Zipf's Law [151].

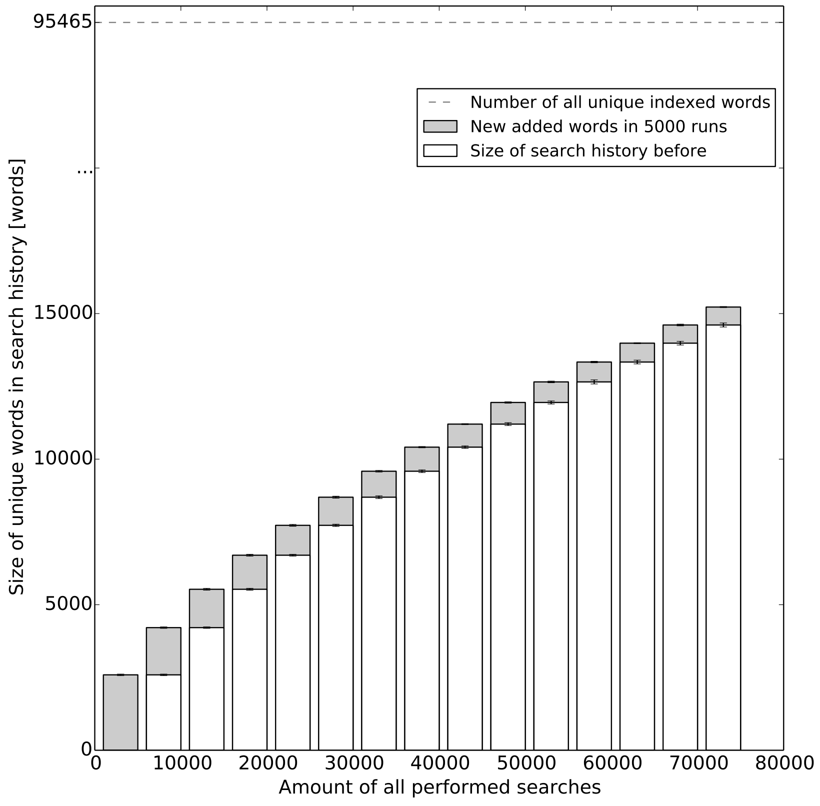


Figure 5.2: Unique search tokens queried; gray represents new search tokens not asked before.

point in Figure 5.3 at position i on x-axes. At the beginning the service provider did not know any search tokens so that the search history σ is empty. Given a search token in this situation, the service provider had to check the index of every file, i.e. every list representing one file in γ_f had to be checked value by value. With an increasing size of the search history an increasing number of search tokens were indexed in our reverse index γ_w and hence the service provider was able to answer these queries faster. Our results also show that the optimal search time is reached much faster than in n^2 searches, since we only perform 70.000 searches for more than 95.000 unique keywords.

5.5 Tradeoff between Communication Overhead and Client Storage

We maintain a history of previously used search tokens at the client and use it during the add operation. The client creates the corresponding search tokens immediately as the deterministic identifier of the keyword. Hence, the service provider can include it in the index. Note that the search token is not necessarily part of the add token, rather it could be randomized. In order to check whether a randomized add token corresponds to a search token, the service provider would need to check all previous search tokens. Only then, it could convert the randomized add token to the deterministic search token. To the contrary, the client can compute both – search and add token of the inserted keyword – and simply look up the search token in the history. Hence, the cost of one insertion is $O(\texttt{len}\,(()\,\overline{f}))$ using a history at the client and $O(\texttt{len}\,(()\,\overline{f})\texttt{len}\,(SRCH\_HIS_t(\mathtt{f})))$ using a history at the service provider. Our solution using no client storage (except the key) modifies the Add and AddToken operations as follows:

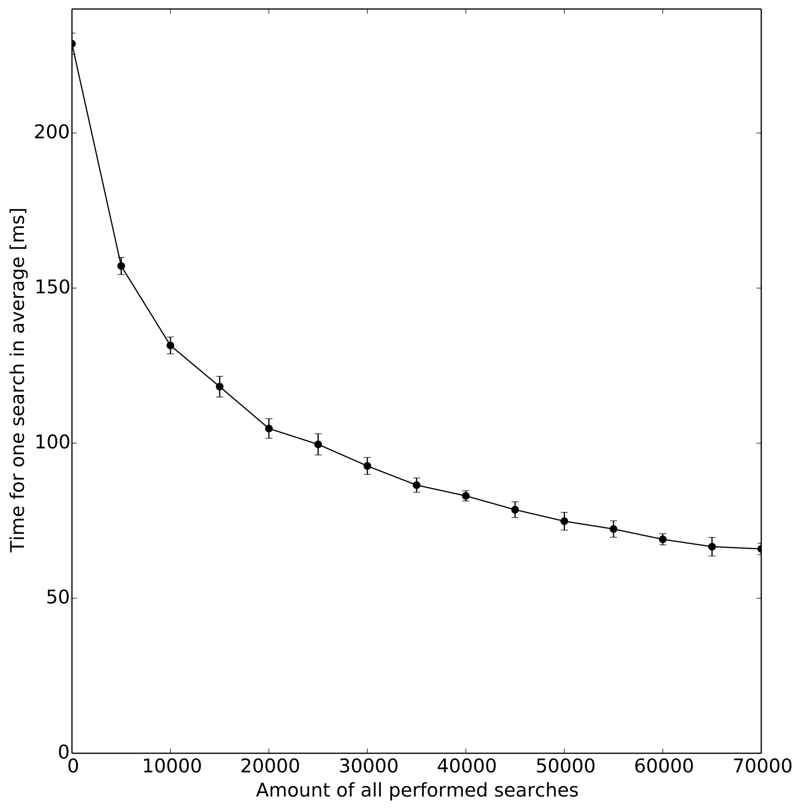


Figure 5.3: Mean query time for one random generated search token of 5000.

- $\alpha_f \leftarrow \mathsf{AddToken}(K, f, \sigma)$: For file f that consists of a sequence of words create a list $\overline{f}$ of *unique* words $f \supseteq \overline{f} = (w_1, \ldots, w_{\mathtt{len}(\overline{f})})$. Generate a sequence of pseudorandom values $s_1, \ldots s_{\mathtt{len}(\overline{f})}$ with PRNG G. For every word $w_i \in \overline{f}$ set $c_i = H_{\tau_{w_i}}(s_i)||s_i$ with $\tau_{w_i} = F_K(w_i)$. Now sort $\overline{c} = (c_1, \ldots, c_{\mathtt{len}(\overline{f})})$ in lexicographic order and set $\alpha_f = (ID(f), \overline{c})$. Output α_f.
- $\gamma' \leftarrow \mathsf{Add}(\alpha_f, \gamma)$: Parse $\alpha_f = (ID(f), \overline{c})$, $\gamma = (\gamma_w, \gamma_f)$ and set $\gamma_f[ID(f)] = \overline{c}$. In addition, for each $\tau_{w_i} \in \gamma_w$ and each $c_j \in \overline{c}$ set $c_j = l_j||r_j$ and check if $H_{\tau_{w_i}}(r_j) = l_j$. If yes, add $ID(f)$ to $\gamma_w[w_i]$. Output the updated version of the search index $\gamma' = (\gamma_w, \gamma_f)$.

The history at the client is the same as the index words $ID(w)$ in the inverted index γ_w at the service provider. Hence, the client can always restore its history by downloading these from the server. Moreover, let the number of unique keywords be n, then the size of the history will never exceed $O(n)$ independent of the number of searches performed.

5.6 Summary

We have demonstrated a new technique for efficient, dynamic searchable encryption. Our idea is to learn and incrementally construct the index from the search token. We have theoretically shown that this leads to the optimal search time over a sufficiently long period. We have experimentally shown that this search time is low in absolute numbers and hence highly practical. Furthermore, it is reached much faster in practice than the theoretical upper bound.

Our scheme can be implemented without client storage and only requires to store two cryptographic hash values per index entry. Additions and deletes can be performed securely. We maintain semantic security even during updates, i.e. we leak no additional information. We give detailed leakage functions in our simulation-based security proof.In our experiments using real-world search terms, 84% of all keywords were never searched for. These keywords remain semantically secure encrypted and hence profit from the additional security under updates provided by our scheme.

We believe our construction to be valuable from two perspectives. First, it provides a novel design alternative for constructing dynamic searchable encryption scheme. Second, it provides a favorable tradeoff compared to deterministic encryption, since it is almost as efficient – the time for the second search of a keyword is the same – but significantly more secure. Hence, it provides a viable alternative for practical adoption.

6 Secure Joins

In this chapter we present protocols for secure joins but with fine granular leakage enabling the client to delegate database joins of encrypted tables. More particular, we do not only consider full joins of two tables but include additional attribute constraints, e.g. in form of where clauses, in our considerations. These additional constraints together with the actual join operations allow us to achieve fine granular information leakage. In the following Section 6.1 we give an introduction, discuss the problem in more details with one example and give a formal definition. Further, we present a general interface that offers the functionality of database joins on encrypted tables. An overview of related work addressing the problem of database joins on encrypted tables is presented in Section 6.2. In Section 6.3 we describe a naive solution for this problem and highlight shortcomings of this attempt. Further, we describe our abstract ideas addressing these shortcomings and present a concrete implementation. Based on this concrete implementation a detailed evaluation is given in Section 6.4 regarding both formal security and performance. Although join operations are technically no search operation[1] we follow the approach of searchable symmetric encryption for the theoretical security analysis and define a leakage function proving security based on this leakage. Additional tradeoffs between the security of values used as predicates in the where clause and practical execution time are discussed in Section 6.5. Finally, Section 6.6 provides a summary of this chapter. The content of this chapter is based on the master's thesis by Nicolas Loza who has been supervised by the author of this dissertation. Further, an academic paper containing the results is to be published.

- Loza, Nicolas: *Implementing Secure Join Operations over Encrypted Databases with Low Information Leakage*, Karlsruher Institute für Technologie (KIT), Master's Thesis, 2017
- Hahn, Florian ; Loza, Nicolas ; Kerschbaum, Florian: Encrypted Database Joins with Fine Granular Security, ???? (In Preparation).

6.1 Introduction

In order to decrease data redundancy and increase data consistency for data stored in relational databases, the process of database normalization introduced by Codd [46] is applied during the database design phase. In this thesis we address joins for tables in third normal form, that is, all tables contain only columns that are non-transitively dependent on the primary key. This is achieved by splitting the table into two tables, where previously depended data is stored in its own separate table.

The dependency is then modeled as primary key in the one table, and foreign key in the second table and can be reconstructed using the join operation in the data query, e.g. a SQL `SELECT` statement in combination with the `JOIN` keyword. For example, Table 6.1 shows a table before such transformation, whereas the resulting tables after the transformation are shown in Table 6.2 with primary key "DName" and in Table 6.3 with foreign key "Dept". One instance for an equi-join query in combination of additional filtering conditions we aim to support on encrypted databases with fine granular information leakage is expressed in SQL as given in the following Listing 6.1.

[1] Following the notation of relational algebra, the join operation is a binary operation, whereas the search operation or selection is a unary operation

EmpID	**Name**	**Salary**	**Dept**	**Manager**
1	Harry	4000	Finance	George
2	Sally	7000	Sales	Harriet
3	Thomas	5000	Finance	George
4	Oscar	3500	Marketing	Harriett
5	Jim	5500	Marketing	Harriett

Table 6.1: Employee with Managers in one table.

DeptID	**DName**	**Manager**
1	Finance	George
2	Sales	Harriet
3	Marketing	Harriet

Table 6.2: Dept with primary key "DName".

EmpID	**Name**	**Salary**	**Dept**
1	Harry	4000	Finance
2	Sally	7000	Sales
3	Thomas	5000	Finance
4	Oscar	3500	Marketing
5	Jim	5500	Marketing

Table 6.3: Employee with foreign key "Dept".

```
SELECT * FROM Emp JOIN Dept ON Dept = DName WHERE Manager = 'Harriet'
```

Listing 6.1: Example SQL join query

The corresponding result table for that join query is given in Table 6.4. Although the result set might be encrypted, the relation between the encrypted rows in different tables is inevitably leaked to the service provider due to the efficient processing delegation.

Name	**Salary**	**DName**	**Manager**
Sally	7000	Sales	Harriet
Oscar	3500	Marketing	Harriet
Jim	5500	Marketing	Harriett

Table 6.4: Result of query stated in 6.1 on Table 6.3 and Table 6.2.

More formally, given two tables $\mathbf{T_0}, \mathbf{T_1}$, the result of the *inner join* operation on two join columns, one from $\mathbf{T_0}$ and one from $\mathbf{T_1}$ is the set of all combinations of rows from $\mathbf{T_0}, \mathbf{T_1}$ containing equal values in their join columns. In this chapter we focus on inner joins but use the more general term join interchangeably. Let us assume table $\mathbf{T_0}$ has schema $(PK_{\mathbf{T_0}}, A_1, \dots, A_l)$ with primary key $PK_{\mathbf{T_0}}$ and attributes $A_1, \dots, A_l$, this table consists of $|\mathbf{T_0}|$ records $(\text{pk}^1_{\mathbf{T_0}}, a^1_1, \dots, a^1_l), \dots, (\text{pk}^{|\mathbf{T_0}|}_{\mathbf{T_0}}, a^{|\mathbf{T_0}|}_1, \dots, a^{|\mathbf{T_0}|}_l)$; table $\mathbf{T_1}$ has schema $(FK_{\mathbf{T_0}}, B_1, \dots, B_m)$ with foreign key $FK_{\mathbf{T_0}}$ establishing the relationship to table $\mathbf{T_0}$ and attributes $B_1 \dots, B_m$, this table consists of $|\mathbf{T_1}|$ records $(\text{fk}^1_{\mathbf{T_0}}, b^1_1, \dots, b^1_m), \dots, (\text{fk}^{|\mathbf{T_1}|}_{\mathbf{T_0}}, b^{|\mathbf{T_1}|}_1, \dots, b^{|\mathbf{T_1}|}_m)$. In the following we use the row number as row ID, e.g. the third row in Table 6.2 has value "Marketing" as primary key. The join of *join attributes* $PK_{\mathbf{T_0}}$ and $FK_{\mathbf{T_0}}$ is an operation with table $\mathbf{T_0}$ and table $\mathbf{T_1}$ as input and denoted as $\mathbf{T_0} \bowtie \mathbf{T_1}$. The result of $\mathbf{T_0} \bowtie \mathbf{T_1}$ has schema $(PK_{\mathbf{T_0}}, A_1, \dots, A_l, B_1, \dots, B_m)$ and consists of all records $(\text{pk}^i_{\mathbf{T_0}}, a^i_1, \dots, a^i_l, b^j_1, \dots, b^j_m)$ with equal *join values* (either primary or foreign key) $\text{pk}^i_{\mathbf{T_0}} = \text{fk}^j_{\mathbf{T_0}}$ for all $i \in [1, |\mathbf{T_0}|], j \in [1, |\mathbf{T_1}|]$. Note that the primary keys $\text{pk}^i_{\mathbf{T_0}}$ in table $\mathbf{T_0}$ need to be unique, but each primary key maps to possible multiple foreign keys $\text{fk}^j_{\mathbf{T_0}}$; in the following we denote the result of this map as $M_{\mathbf{T_0} \to \mathbf{T_1}}$. In many applications the end user is not interested in the complete inner join over both tables but only a small subset thereof based on additional filtering-predicates chosen from attributes $\{A_1, \ldots, A_l\}$ and $\{B_1, \ldots, B_m\}$.

Current practically implemented solutions for joins on encrypted databases are based on deterministic encryption as proposed by CryptDB [123], i.e. the same plaintext is encrypted to the same ciphertext such that $\mathsf{Enc}_k^{\text{Det}}(x) = \mathsf{Enc}_k^{\text{Det}}(y) \Leftrightarrow x = y$. This preserved property enables join computation on such deterministically encrypted foreign and primary keys. The straightforward application of deterministic encryption supporting database joins, i.e. encrypting join values in different tables under the same key, induces leakage that can be extracted even before the join operation is performed. This leakage can be prevented with advanced techniques such as re-encryption before the actual join operation: Before the actual join operations, join values contained in different tables are encrypted using different keys. For example, values for the "Dept" attribute are deterministically encrypted under key k_1 and values for the "DName" are encrypted under key k_2, hence $\mathsf{Enc}_{k_1}^{\text{Det}}(\text{Finance}) \neq \mathsf{Enc}_{k_2}^{\text{Det}}(\text{Finance})$ contained in different tables. In case of a queried join operation, one column is re-encrypted given a corresponding token generated by the client using both keys but without unveiling the underlying plaintexts to the untrusted server. After the re-encryption process, the encrypted values stored in different columns to be joined are encrypted under the same key enabling join calculation on deterministically encrypted data by a simple equality comparison of ciphertexts. Note that different ciphertexts in the same table unveil equality relations directly after the initial outsourcing step, hence leaking the self-join of an encrypted table. This leakage is addressed by the application of adjustable encryption in CryptDB, i.e. encapsulating the deterministic ciphertexts with an additional randomized encryption that is removed if required.

All previous schemes providing secure join functionality have a property we call *all-or-nothing* security for inner joins: Before the join query the inner join might be hidden due to additional blinding, however, after the join operation of two different tables the inner join of these tables is leaked completely. That is, even though the result set the client is interested in might be a small subset of this inner join – it might even be an empty set due to additional filtering predicates – the server learns the complete inner join. In our example query specified in Listing 6.1, although the tuples (Harry, 4000, Finance) and (Thomas, 5000, Finance) from Table 6.3 "Employee" are encrypted and not part of the actual query result, the server learns that these tuples contain the same value and that they contain the same join-value as tuple (Finance, George) from Table 6.2 "Dept".

In this chapter we strive to minimize this additional leakage for encrypted database joins not directly derivable from the encrypted result set the client has queried. As a result, we achieve a *fine granular* security policy on the granularity level of the actual query answer maintaining semantic security for all datasets that have never been part of any query result.

6.1.1 Framework

We assume the values to be joined on, named join-values ν are known before hand, i.e. the client knows which column of $\mathbf{T_0}$ contains the primary keys and which column contains the foreign keys $\mathbf{T_1}$. All additional columns can be used by the client to state additional filtering clauses for the join operation. We denote ι as the complete join query comprising the specified filtering predicates also denoted as row attributes in the following.

The general framework of our construction works as follows: In a first step, the client creates a master key K used for data encryption and later for query token generation. The tables $\mathbf{T_0}$ and $\mathbf{T_1}$ are encrypted using K, denoting the encrypted tables as $\mathbf{C_0}$ and $\mathbf{C_1}$. The encryption of a complete table is broken down to multiple calls of EncRow encrypting individual rows. Since we may support different encryption algorithms for primary keys and foreign keys, their corresponding key role is indicated by b for each row encryption. After the outsourcing step, the client wishes to execute join queries on encrypted tables; we assume these join queries ι are supplemented with additional filter constraints on the attribute predicates. The client calls $\mathsf{GenToken}$ with the master key K and the join query ι resulting in a join token τ_ι for this specific query. This join token τ_ι is then transferred to the server enabling her to compute the join result on encrypted tables $\mathbf{C_0}$ and $\mathbf{C_1}$ calling Join.

Formally, the framework of our secure join scheme with support of additional the filtering predicates as discussed previously provides the following algorithms:

Definition 15 (Secure Join Scheme)**.** *Let $\mathbf{T_0}$ and $\mathbf{T_1}$ be the tables to be encrypted and joined later on. A scheme* $\mathsf{SecJoin}$ *supporting* secure joins with fine granular leakage *implements the following algorithms:*

$K \leftarrow \mathsf{Setup}(1^\lambda)$ *is a probabilistic algorithm that takes as input a security parameter λ. It outputs the master key K.*

$c \leftarrow \mathsf{EncRow}(K, \mathsf{b}, \nu, \mathbf{s})$ *is a probabilistic algorithm that takes as input the master key K, the indicator* b *indicating the type of join-value ν (i.e. ν is a foreign key if* $\mathsf{b} = 1$*, primary key otherwise) and the corresponding row attributes* $\mathbf{s}$*. It outputs an encrypted join value c that is compatible with table $\mathbf{T_b}$ and can be joined with table $\mathbf{T_{1-b}}$.*

$\mathbf{C_b} \leftarrow \mathsf{EncTab}(K, \mathbf{T_b})$ *is a deterministic algorithm that takes as input the master key K and a table $\mathbf{T_b}$. It runs* EncRow *for every row in $\mathbf{T_b}$ and the collection of all resulting encrypted join values is returned in form of an encrypted table $\mathbf{C_b}$.*

$\tau_\iota \leftarrow \mathsf{GenToken}(K, \iota)$ *is a (possibly probabilistic) algorithm that takes the master key K and a* join query *ι consisting of additional conditions on the attribute predicates for the tables $\mathbf{T_0}, \mathbf{T_1}$, e.g. specified via a where clause in SQL. It returns a join token τ_ι for the corresponding query.*

$M_{\mathbf{T_0} \rightarrow \mathbf{T_1}} \leftarrow \mathsf{Join}(\mathbf{C_0}, \mathbf{C_1}, \tau_\iota)$ *is a deterministic algorithm that takes as input the two encrypted tables $\mathbf{C_0}$ and $\mathbf{C_1}$, together with join token τ_ι. The result is a map $M_{\mathbf{T_0} \rightarrow \mathbf{T_1}}$, which maps row IDs in $\mathbf{T_0}$ to their sets of matching row IDs in $\mathbf{T_1}$.*

Note that this scheme can be extended to support joins over multiple tables with the same foreign key column to be joined on, i.e. for all tables $\mathbf{T}_j$ that contain the foreign key call EncRow with indicator $\mathsf{b} = 1$.

6.2 Related Work

Popa et al. [125] provide the functionality of secure joins by the utilization of deterministic encryption that offers the possibility of re-encryption [19]. That is, each column is encrypted deterministically with a different key, however, given a re-encryption token this key can be changed *without* an intermediate decryption on the server side. In the case of a join query, the client creates such re-encryption token allowing to adjust the encryption key of one column matching the encryption key of the second table, hence enabling equi-joins by simple ciphertext comparison. For such adjustable joins on multiple tables Kerschbaum et al. [90] present strategies for optimizing the performance with respect to the required re-encryption overhead. However, the security of deterministically encrypted data strongly depends on the underlying plaintext distribution and may be exploitable by an attacker as Naveed et al. demonstrated recently [115] and reviewed in more details in Section 3.2.3.

The idea of using secure coprocessors for computing joins on outsourced data has been introduced by Agrawal et al. [4]. Because of the low amount of memory available in these devices, the security of such methods relies on reading and writing from and to the processor in such a way, that the I/O access pattern leaks the smallest amount of information as possible. Li et al. [102] later showed that the security assumption from Agrawal et al. lead to unnecessary information leakage, and have replaced it with a new definition, based on which they proposed three new algorithms for computing general joins on arbitrary predicates. Despite these efforts, neither work [4, 102] provides a well-defined leakage formulation for their proposed algorithms. Arasu and Kaushik [8] give an approach for oblivious query processing that hides the access pattern of the join query and only unveils the join's result size. They call query processing oblivious iff. no attacker can distinguish the memory access sequence of two different databases and presented an algorithm fulfilling this security definition. However, all solutions based on secure hardware require additional trust assumptions regarding the hardware vendor.

An alternative approach solely based on cryptographic assumptions is based on SMC, specifically private set intersection (PSI) as introduced by Fagin et. al [56] and formally studied by Freedman et al. [59]. The first efficient solutions have been proposed by De Cristofaro et al. [50]. Since then many improved protocols have been published including ones theoretically most efficient [121] and practically deployed [149]. Outsourced PSI [87, 88] allows clients to upload their data to a server and then perform PSI which is closer to our system setting. However, all PSI protocols assume that both sets contain only unique join values. This constraint renders all these solutions not applicable to secure joins due to possibly multiple occurrence of the same foreign key in one of the tables to be joined.

The first solution without this constraint has been proposed by Carbunar et al. [37] based on obfuscated bloom filters constructed for the join values. For enabling the join operation, this obfuscation is removed, degrading the security level for values stored in columns to be joined to that of deterministic encryption. Further, due to the nature of bloom filters, post-processing for removing false positives is required on the client side. In contrast, our solution does not raise false positives, hence client side computation is independent of the result set size. Pang et al. [120] propose a secret key encryption scheme and Wang et al. [145] propose a public key encryption scheme both rely on pairing based cryptography that allow equi-joins, encrypted by the client and by an untrusted third party, respectively. However, all schemes with support of non-unique join values published so far offer all-or-nothing security, in the sense that once the join has been performed the inner join is unveiled completely.

6.3 Implementation

Our construction achieving fine granular leakage is based on two general ideas: First, searchable symmetric encryption as proposed by Song et al. [137] provides the same functionality as deterministic encryption; hence one can replace the deterministically encrypted values with either search tokens or search ciphertexts. Second, it is sufficient to unveil the join result for all rows that match the additional filtering clause. Instead of joining the complete tables followed by a filtering on that join-result, we follow the orthogonal approach and run a privacy-preserving filtering on the complete table and perform the join operation on that filtered result set. Particularly, we demonstrate the second idea with the following straw-man implementation for the framework stated in Definition 15. This implementation increases the security compared to all current solutions [37, 120, 123, 145] for databases with *only one* additional attribute column. The drawbacks of this straw-man construction are then highlight in the end of this section and we discuss solutions in the remainder of this chapter.

6.3.1 Straw-Man Solution

The goal the of straw-man implementation presented in this section is to improve security with respect to the previously discussed all-or-nothing security, i.e. this construction provides security with finer granularity. In this preliminary construction we assume only one additional attribute column for each table; $\mathbf{T_0}$ has schema (PK_A, A_1) and $\mathbf{T_1}$ has Schema (FK_A, B_1) with join-attributes PK_A, FK_A respectively, and a filter clause ι with one filtering predicate a for attribute A_1 and one filtering predicate b for B_1. For the sake of simplicity we follow the idea by Popa et al. [123] and utilize deterministic encryption for protecting the encrypted values supporting joins. However, in contrast to CryptDB, we reduce the leakage of a secure join scheme $\mathsf{SecJoin}$ founded on the following observation: it is sufficient to unveil the join result for all rows that match the additional filtering clause ι, i.e. having value a for column A_1 in $\mathbf{T_0}$ and value b for column B_1 in $\mathbf{T_1}$. Thus, instead of joining the complete tables $\mathbf{T_0}$ and $\mathbf{T_1}$ followed by a filtering on that join-result, we follow the orthogonal approach and run a privacy-preserving filtering on the complete tables $\mathbf{T_0}$ and $\mathbf{T_1}$, and perform the equi-join operation on that filtered result set. This is enforced by additional encapsulation of all join-values using a semantically secure encryption scheme $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ keyed with a secret key k_a and k_b derived from the attribute predicates. Further, we assume the decryption algorithm Dec indicates successful decryption. That is, decryption $\mathsf{Dec}(k', \mathsf{Enc}(k, m))$ is called successful if and only if $k' = k$; this can be implemented by, e.g. concatenating the hash value $h(m)$ to the encryption: $\mathsf{Enc}(k, m||h(m))$ and checking this relation in the decryption algorithm. Given a deterministic encryption scheme $\Pi^{\mathsf{Det}} = (\mathsf{Gen}^{\mathsf{Det}}, \mathsf{Enc}^{\mathsf{Det}}, \mathsf{Dec}^{\mathsf{Det}})$, a semantically secure encryption scheme $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ with key space $\mathcal{K}$ and a key derivation function $\mathsf{KDF} : \{0,1\}^\lambda \times \{0,1\} \times \{0,1\}^* \to \mathcal{K}$, we can implement a secure join scheme supporting one attribute per table according to the framework given in Definition 15 as follows:

$K \leftarrow \mathsf{Setup}(1^\lambda)$**:** Output $K \leftarrow \mathsf{Gen}^{\mathsf{Det}}(1^\lambda)$.

$c \leftarrow \mathsf{EncRow}(K, \mathsf{b}, \nu, s)$**:** Sample row key $\mathsf{SK}_s \leftarrow \mathsf{KDF}(K, \mathsf{b}, s)$ and encrypt the deterministic encryption $\mathsf{Enc}^{\mathsf{Det}}(K, \nu)$ of the join value ν using row key SK_s as $c \leftarrow \mathsf{Enc}(\mathsf{SK}_s, \mathsf{Enc}^{\mathsf{Det}}(K, \nu))$. Output the double encrypted ciphertext c.

$\mathbf{C}_\mathsf{b} \leftarrow \mathsf{EncTab}(K, \mathbf{T}_\mathsf{b})$**:** For every row in $\mathbf{T}_\mathsf{b}$ call EncRow and return the corresponding encrypted table $\mathbf{C}_\mathsf{b} = \{\mathsf{EncRow}(K, \mathsf{b}, \nu^j, s^j)\}_{j \in [1, \mathtt{len}(\mathbf{T}_\mathsf{b})]}$.

$\tau_\iota \leftarrow \mathsf{GenToken}(K, \iota)$**:** In this simplified construction $\iota = (a, b)$ consists of exactly two attributes, a for attribute A_1 and b for attribute B_1. Derive the corresponding encapsulation keys $\mathsf{SK}_0 \leftarrow \mathsf{KDF}(K, 0, a)$, $\mathsf{SK}_1 \leftarrow \mathsf{KDF}(K, 1, b)$ and return join token $\tau_\iota = (\mathsf{SK}_0, \mathsf{SK}_1)$ for the given query.

$M_{\mathbf{T_0} \leftarrow \mathbf{T_1}} \leftarrow \mathsf{Join}(\mathbf{C_0}, \mathbf{C_1}, \tau_\iota)$**:** Parse token $\tau_\iota = (\mathsf{SK}_0, \mathsf{SK}_1)$. For every encrypted join value $e^j \in C_\mathsf{b}$ use SK_b for decrypting it, resulting in $\mathsf{Dec}(\mathsf{SK}_\mathsf{b}, e^j) = \mathsf{Enc}^{\mathsf{Det}}(K, \nu)$. For each encrypted primary key $\mathsf{ep}^i = \mathsf{Enc}^{\mathsf{Det}}(K, \nu)$ in $\mathbf{C_0}$ that is decrypted successfully, create map entry $M[i]_{\mathbf{T_0} \to \mathbf{T_1}} \leftarrow \{j : \mathsf{ef}^j = \mathsf{Enc}^{\mathsf{Det}}(K, \nu) \in \mathbf{C_1}$ with $\mathsf{ef}^j = \mathsf{ep}^i\}$. Finally, output the complete mapping $M_{\mathbf{T_0} \to \mathbf{T_1}}$.

In the following, we discuss three drawbacks of this straw-man construction and give formal comprehensive solutions for these drawbacks in the next section.

First, filtering is linear in the table size, that is, all rows must be decrypted and checked for a successful decryption. We emphasize that we assume solely the join values as sensitive data, hence we strive to minimize the leakage that can be extracted in regards to the secure join operations and the underlying join-values. In contrast, we do not consider the encryption of additional row attributes in EncRow explicitly but only decode their values to provide fine granular security for join values. Depending on the use case, one might outsource the additional attribute columns in plaintext, enabling the construction of indexing data structures on plaintext that allow to retrieve efficiently all rows whose correct decryption keys are contained in τ_ι. If the row attributes are considered to be sensitive, the application of a dynamic and efficient searchable symmetric encryption scheme as discussed in the previous Chapter 5 is a viable alternative and the leakage is analyzed in Section 6.5. In this chapter we do not address the information leakage for the row attributes and the additional leakage induced by such inverted indexing techniques. This leakage has been studied in previous work on dynamic and efficient searchable symmetric encryption. Particularly the work published by Cash et al. [39] is of great interest for our application, since it provides the functionality to query search terms in boolean formulas. Hence, we either assume plaintext row attributes or the secure application of dynamic and efficient searchable symmetric encryption schemes. Both cases result in more efficient pre-filtering step compared to a linear scan of the complete tables. In the following discussion we solely focus on protecting the join-values (i.e. both primary and foreign key) but do not consider the security of additional attribute predicates, nor do we address the algorithms for potential pre-filtering.

Second, the application of deterministic encryption for securing the join values has additional leakage that cannot directly be derived from the join result. That is, all values matching the where clause in table $\mathbf{T_1}$ leak that they have the same foreign key fk even if they are not part of the equi-join result, e.g. because no matching primary key from table $\mathbf{T_0}$ fulfills the where clause. As also observed by Pang et al. [120], this enables the server to extract the result of a self-join, although not queried explicitly by the client.

Third, generalization of this straw-man construction to multiple row attributes for each table increases the required memory to be exponentially in the number of attributes. In more details, given table $\mathbf{T_0}$ with schema $(PK, A_1, \dots A_n)$ and the possibility to filter for all n attributes, the protected join value must be blinded with all possible combinations of the n attributes resulting in 2^n different keys SK_i^j for $j \in [1, 2^n]$ and the resulting blinded encrypted join values for all 2^n different keys must be stored for each row in $\mathbf{T_0}$. The analogue argument is true for table $\mathbf{T_1}$.

While the first drawback is addressed extensively by previous work, e.g. extensions for searchable encryption for exact pattern matching as discussed in the previous chapter, we focus on the latter two for the remainder of this chapter.

6.3.2 Required Tools

We identified different cryptographic tools that address these problems and enable us to reduce the information leakage induced by join operation on a finer granularity. Putting these tools together we present a comprehensive description of our implementation fitting the framework for secure joins as specified in Definition 15. We quantify an upper bound for the information leakage induced by our implementation based on the security properties offered by these tools and prove this upper bound in the formal framework from Section 4.1.1.

Recall that we assume two different tables $\mathbf{T_0}, \mathbf{T_1}$ where the join values of $\mathbf{T_0}$ are primary keys, hence they are unique, while the join values of $\mathbf{T_1}$ are foreign keys and might occur several times. As a result, the application of deterministic encryption Π^{Det} for equality checks on encrypted data as proposed in Subsection 6.3.1 has no consequences on the security level of encrypted values contained in join column of $\mathbf{T_0}$, however, weakens security for encrypted values contained in join column of $\mathbf{T_1}$. In order to minimize this security penalty while still providing the functionality of matching encrypted values for equality, we replace the deterministic encryption scheme with a searchable searchable symmetric encryption (SSE) as introduced by Song et al. [137] and summarized in Definition 9. Recall that the ciphertext c_w output by SSE-Enc for keyword w is randomized even for the same input (i.e. the same key K and plaintext w), while the token t_v output by SSE-Token is deterministic for search word v. With access to both, namely a ciphertext c_w and a token t_v, it is possible to check for equality using algorithm SSE-Match, i.e. if $w = v$ is true. We refer to both, the SSE-ciphertext c_w and the SSE-token t_w for word w as *SSE-values* in the following. In order to model the inner join functionality for $\mathbf{T_0}$ and $\mathbf{T_1}$ using SSE, one encrypts all unique join-values of $\mathbf{T_0}$ calling SSE-Token and all (probably non-unique) join-values of $\mathbf{T_1}$ calling SSE-Enc. Here, the correct choice is crucial for the security, since the application of SSE-Enc on the non-unique values hides the frequency due to its randomized output characteristics making self-joins on $\mathbf{T_1}$ impossible.

We emphasize that, although $\mathbf{T_1}$ on its own is semantically secure after applying SSE-Enc, all values that occur in $\mathbf{T_0}$ as well have additional leakage due to the comparison functionality provided by SSE-Match. An honest-but-curious adversary with access to all SSE-values can reconstruct mapping $M_{\mathbf{T_0} \to \mathbf{T_1}}$, that is, the adversary can define sets

$$R_i = \{j \in [1, |\mathbf{T_1}|] \mid \mathsf{SSE\text{-}Match}\left(\mathsf{SSE\text{-}Enc}\left(K, fk^j\right), \mathsf{SSE\text{-}Token}\left(K, pk^i\right)\right) = 1 \text{ for } i \in [1, |\mathbf{T_0}|]\}$$

grouping randomized ciphertexts (in this case their IDs) for the same underlying plaintext value. As a consequence, blinding the SSE-values (both SSE-ciphertexts and SSE-tokens) remains a vital protection to provide fine granular security properties for the outsourced databases with support of secure joins.

We address the straw-man solution's exponential memory blowup by utilizing the concept of attribute-based encryption (ABE). ABE is an expansion of public key cryptography that allows the encryption and decryption of messages based on attributes assigned to the ciphertext during encryption time. Originally presented by Sahai et al. [129], it focused on ascribing the ciphertext with a policy described as predicate logic $f(\cdot)$. This predicate logic is then required to be satisfied by the user's credentials for a successful decryption. Later, Goyal et al. [70] defined this as *ciphertext-policy attribute-based encryption (CP-ABE)*, while also defining its complementary: *key-policy attribute-based encryption (KP-ABE)*. In the latter, attributes are used to annotate ciphertexts, and formulas over these attributes are assigned to keys generated by the user. These formulas must then be satisfied by the attributes in the ciphertext for a successful decryption. For the purposes of our construction, we will henceforth use KP-ABE, and whenever we use the term ABE we implicitly refer to KP-ABE.

In order to define the algorithms necessary for ABE, we first need to define the term access structure.

Definition 16 (Access Structure according to [80])**.** *Let $P = \{P_1, ..., P_n\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^P$ is monotone if $\forall B, C : B \in \mathbb{A} \land B \subseteq C \Rightarrow C \in \mathbb{A}$. An* access structure *(respectively, monotone access structure) is a collection (resp., monotone collection) $\mathbb{A}$ of non-empty subsets of P, i.e. $\mathbb{A} \subseteq 2^P \backslash \{\emptyset\}$. The sets in $\mathbb{A}$ are called the* authorized sets*, and the sets not in $\mathbb{A}$ are called the* unauthorized sets.

Based on this definition, we can now specify the algorithms implemented by any KP-ABE scheme we will use in the following as black box.

Definition 17 (KP-ABE Scheme according to [80])**.** *Given a message space $\mathcal{M}$ and access structure space $\mathcal{G}$, we define a key-policy attribute-based encryption (KP-ABE) scheme* ABE *as a tuple of the following (possibly probabilistic) polynomial-time algorithms:*

$(PK, \mathrm{MK}) \leftarrow \mathsf{ABE\text{-}Setup}(1^\lambda, U)$ *is a probabilistic algorithm that takes as input a security parameter λ and an universe description U defining the set of allowed attributes in the system. It outputs the public parameters PK and the secret key* MK.

$CT \leftarrow \mathsf{ABE\text{-}Enc}(PK, M, \mathbf{S})$ *is a probabilistic algorithm that takes as input the public parameters PK, a message M and a set of attributes* $\mathbf{S}$*, where each attribute $s_i \in \mathbf{S}$ is drawn from universe U. A randomized ciphertext CT associated with the attribute set is output.*

$SK_{\mathbb{A}} \leftarrow \mathsf{ABE\text{-}Key}(\mathrm{MK}, \mathbb{A})$ *is a probabilistic algorithm that takes as input the master secret key* MK *and an access structure* $\mathbb{A}$*, and outputs a (randomized) private key* $SK_{\mathbb{A}}$ *associated with the attributes described by* $\mathbb{A}$*.*

$M \leftarrow \mathsf{ABE\text{-}Dec}(SK_{\mathbb{A}}, CT)$ *is a deterministic algorithm that takes as input a private key* $SK_{\mathbb{A}}$ *associated with access structure* $\mathbb{A}$ *and a ciphertext* CT *associated with attribute set* $\mathbf{S}$ *and outputs the message* M *encrypted in* CT *iff.* $\mathbf{S}$ *satisfies* $\mathbb{A}$*.*

Security for KP-ABE schemes is defined as follows.

Definition 18 (Selective security against chosen plaintext attacks for KP-ABE from [80])**.** *Let* $\mathsf{ABE} = (\mathsf{ABE\text{-}Setup}, \mathsf{ABE\text{-}Enc}, \mathsf{ABE\text{-}Key}, \mathsf{ABE\text{-}Dec})$ *be a KP-ABE scheme for message space* $\mathcal{M}$ *and access structure space* $\mathcal{G}$*. Consider the following experiment* $\mathrm{Exp}^{\mathrm{SelCPA}}_{\mathsf{ABE},\mathcal{A}}(\lambda, U)$ *between a challenger and an adversary* $\mathcal{A}$*, security parameter* λ *and attribute universe* U*.*

Init: $\mathcal{A}$ *gives a set of attributes* $\mathbf{S}^*$ *used for encrypting the message she wishes to be challenged later on.*

Setup: *The challenger runs the* ABE-Setup *algorithm and gives the public parameters* PK *to* $\mathcal{A}$*.*

Query Phase 1: *The challenger initializes an empty table* T *and an integer counter* $j = 0$*. The adversary can repeatedly issue queries, where each query is one of two types:*

1. *Create query: The adversary submits an access structure* $\mathbb{A}$*. The challenger sets* $j = j + 1$*. It runs the key generation algorithm* $SK_{\mathbb{A}} \leftarrow \mathsf{ABE\text{-}Key}(\mathrm{MK}, \mathbb{A})$ *and stores in* $T[j]$ *the entry* $(\mathbb{A}, SK_{\mathbb{A}})$*. This can be repeatedly queried with the same input.*
2. *Corrupt query: The adversary inputs a number* i*. If there exists an entry* $T[i]$*, then the challenger obtains tuple* $(\mathbb{A}, SK_{\mathbb{A}})$ *stored at position* i*. Here,* $\mathcal{A}$ *cannot corrupt an access structure* $\mathbb{A}$ *which is satisfied by* $\mathbf{S}^*$*. If no entry* i *exists or it violates the restriction then the challenger returns* $\perp$*. Otherwise, the challenger returns the private key* $SK_{\mathbb{A}}$ *to* $\mathcal{A}$*.*

Challenge: $\mathcal{A}$ *submits two equal length messages* M_0 *and* M_1 *from the message space* $\mathcal{M}$*.*

The challenger flips a random coin $b \overset{\$}{\leftarrow} \{0, 1\}$ *and encrypts* M_b *under* $\mathbf{S}^*$*. The resulting challenge ciphertext* $CT^* \leftarrow \mathsf{ABE\text{-}Enc}(PK, M_b, \mathbf{S}^*)$ *is given to* $\mathcal{A}$*.*

Query Phase 2: *The adversary can issue the same queries as in Phase 1.*

Guess: $\mathcal{A}$ *outputs a guess* b'*.*

The experiment outputs 1 *if* b *equals* b' *and* 0 *otherwise.*

We say that ABE *is selective-secure against chosen-plaintext attacks for attribute universe* U *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *running this security experiment, it holds that*

$$\left| \Pr\left[\mathrm{Exp}^{\mathrm{SelCPA}}_{\mathsf{ABE},\mathcal{A}}(\lambda, U) = 1 \right] - \frac{1}{2} \right|$$

is negligible in λ*.*

Note that general KP-ABE schemes have no claims with respect to the security of the attribute set $\mathbf{S}$ used in ABE-Enc, hence an attacker can potentially extract information about the used attribute set from a given ciphertext generated under this set.

Further, according to the standard definition of KP-ABE, a finite attribute universe U is used as domain of attributes specified in set $\mathbf{S}$. However, there are lines of work that propose implementations of KP-ABE with arbitrarily large attribute universes, like the one presented by Hohenberger and Waters [80]. For future references we therefore will omit the usage of U. The access structures used to generate ABE-keys can be constructed from any boolean formula, as shown by Lewko and Waters [99].

In our application, we will blind the (SSE encrypted) join value with ABE under the attribute values specified in the additional predicates and we use ABE-keys to describe the restrictions of a join query in form of additional filtering attributes, e.g. the `WHERE` clause in a SQL query. This constructions supports arbitrary restrictions described as boolean formulas, since ABE supports them as well.

We emphasize that, following the approach of Kiayias et al. [92], this flexibility in the policy formulation can be utilized to allow range filtering with only logarithmic (in the value domain size) attribute blowup, e.g. a column stores values $v_i \in D$ that should be compatible with range queries in the where clause in a SQL query. Further details are given in Appendix A.3 as this construction is crucial for encrypted range queries. For the sake of a brief and coherent security proof, however, our construction in the remainder of this work will focus only on *conjunctions*, i.e. formulas where all the specified restrictions must be fulfilled. Thus, from now on we write $\mathsf{ABE\text{-}Key}(\mathrm{MK}, \{s_1, \dots, s_l\})$, referring to the access structure describing the conjunction of all values $s_1, \dots, s_l$.

Depending on the construction of KP-ABE, the sizes of the ciphertexts produced by $\mathsf{ABE\text{-}Enc}$ and of the keys produced by $\mathsf{ABE\text{-}Key}$ can vary. While some solutions might result in constant ciphertext size [10], we will focus on constructions optimized for efficient evaluation operation for our purposes. One example for such construction is given by Hohenberger and Waters [80] which requires only two pairings per decryption and none for encryption or key generation, and produces ciphertexts and keys with sizes that are linear in the number of attributes used for their generation. We refer to Appendix A.2 for a formal description of this scheme.

6.3.3 Protocol

In summary, the straw-man construction is based on blinding deterministic encryption of the join values. The keys used for this blinding are derived from their corresponding attribute predicates. This construction has two drawbacks. First, an honest-but-curious attacker can deduce a self-join on $\mathbf{T_1}$ for all unblinded keys without being queried explicitly, and Second, extending this construction to multiple attributes predicates, it has an exponential memory-overhead in the number of attributes.

Our implementation addresses both drawbacks, with the following approaches: first, the functionality of equality checks is realized with SSE, thus rendering self-joins impossible, and second, we reduce the memory-overhead to be linear in the number of attributes by using KP-ABE. This approach is sketched in Figure 6.1.

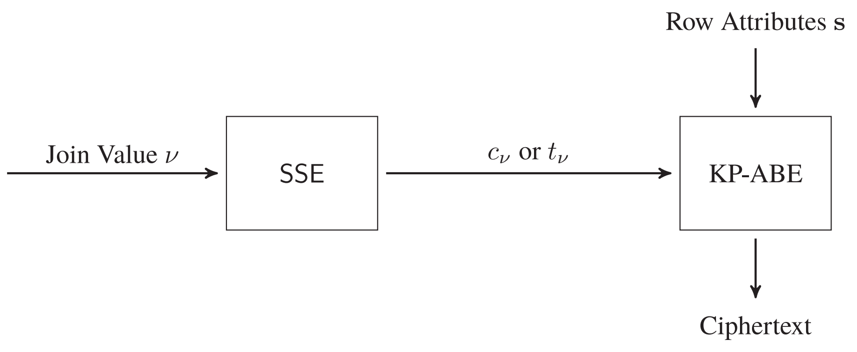


Figure 6.1: Our solution offering fine granular information leakage for secure joins. The ciphertext output by SSE is denoted as c_ν and the search tokens are denoted as t_ν.

Now we present our main construction based on a searchable symmetric encryption scheme $\mathsf{SSE} = (\mathsf{SSE\text{-}Setup}, \mathsf{SSE\text{-}Enc}, \mathsf{SSE\text{-}Token}, \mathsf{SSE\text{-}Match})$ (as described in Definition 9) and a key-policy attribute-based encryption scheme (as described in Definition 17) $\mathsf{ABE} = (\mathsf{ABE\text{-}Setup}, \mathsf{ABE\text{-}Enc}, \mathsf{ABE\text{-}Key}, \mathsf{ABE\text{-}Dec})$. Since we assume attribute based encryption without attribute privacy, we use a pseudorandom function $H : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$ to mask the plain values of these attributes. As discussed previously, this deterministic masking enables the server to build index structures for fast filtering operations. Further, we discuss an alternative construction based on attribute-hiding KP-ABE and formalize its security properties in Section 6.5. For the sake of brevity we abuse notation and write $\mathbf{a} = H(\mathrm{hk}, \mathbf{s})$ for $\mathrm{hk} \in \{0,1\}^\lambda$ and set $\mathbf{s}$ where $\mathbf{a}$ is defined as $\mathbf{a} = \{H(\mathrm{hk}, i, s_i) : \forall s_i \in \mathbf{s}\}$. Note that each $s_i \in \mathbf{s}$ is additionally salted with a column identifier i, hence the same attribute value in two different columns results in different hash values. Now we are ready to implement secure joins as specified in Definition 15 as follows:

$K \leftarrow \mathsf{Setup}(1^\lambda)$: Sample the following keys corresponding to the security parameter:

$$\begin{aligned} K_{\mathsf{SSE}} &\leftarrow \mathsf{SSE\text{-}Setup}(1^\lambda) \\ \mathrm{hk}_0 &\overset{\$}{\leftarrow} \{0,1\}^\lambda \\ \mathrm{hk}_1 &\overset{\$}{\leftarrow} \{0,1\}^\lambda \\ K_{\mathsf{ABE}0} &\leftarrow \mathsf{ABE\text{-}Setup}(1^\lambda) \\ K_{\mathsf{ABE}1} &\leftarrow \mathsf{ABE\text{-}Setup}(1^\lambda). \end{aligned}$$

Output master key $K = (K_{\mathsf{SSE}}, \mathrm{hk}_0, \mathrm{hk}_1, K_{\mathsf{ABE}0}, K_{\mathsf{ABE}1})$.

$c \leftarrow \mathsf{EncRow}(K, \mathsf{b}, \nu, \mathbf{s})$: Parse $K = (K_{\mathsf{SSE}}, \mathrm{hk}_0, \mathrm{hk}_1, K_{\mathsf{ABE}0}, K_{\mathsf{ABE}1})$ and create an SSE-value for the join value ν and encrypt it using attribute based encryption under the blinded attributes derived from $\mathbf{s}$

$$\begin{aligned} \mathrm{sseVal} &\leftarrow \begin{cases} \mathsf{SSE\text{-}Token}(K_{\mathsf{SSE}}, \nu) & \text{if } \mathsf{b} = 0 \\ \mathsf{SSE\text{-}Enc}(K_{\mathsf{SSE}}, \nu) & \text{if } \mathsf{b} = 1 \end{cases} \\ \mathbf{a} &= H(\mathrm{hk}_{\mathsf{b}}, \mathbf{s}) \\ c &\leftarrow \mathsf{ABE\text{-}Enc}(K_{\mathsf{ABEb}}, \mathrm{sseVal}, \mathbf{a}). \end{aligned}$$

Output ciphertext c.

$\mathbf{C}_{\mathsf{b}} \leftarrow \mathsf{EncTab}(K, \mathbf{T}_{\mathsf{b}})$: For every row $r^j = (\nu^j, \mathbf{s}^j)$ in $\mathbf{T}_{\mathsf{b}}$ with join value ν^j and attribute values $\mathbf{s}^j$ run $\mathsf{EncRow}(K, \mathsf{b}, \nu^j, \mathbf{s}^j)$ returning the SSE-value encapsulated by the ABE-encryption in form of an encrypted table $\mathbf{C}_{\mathsf{b}}$.

$\tau_\iota \leftarrow \mathsf{GenToken}(K, \iota)$: Parse $K = (K_{\mathsf{SSE}}, \mathrm{hk}_0, \mathrm{hk}_1, K_{\mathsf{ABE}0}, K_{\mathsf{ABE}1})$ and let $\iota = (\iota_0, \iota_1)$ be the filtering values in the where clause corresponding to columns in tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. Compute the ABE private keys for the blinded attribute sets $\mathbf{p}_0, \mathbf{p}_1$ derived from the filtering values ι_0, ι_1

$$\begin{aligned} \mathbf{p}_0 &= H(\mathrm{hk}_0, \iota_0) \\ \mathbf{p}_1 &= H(\mathrm{hk}_1, \iota_1) \\ \mathrm{SK}_{\iota_0} &\leftarrow \mathsf{ABE\text{-}Key}(K_{\mathsf{ABE}0}, \mathbf{p}_0) \\ \mathrm{SK}_{\iota_1} &\leftarrow \mathsf{ABE\text{-}Key}(K_{\mathsf{ABE}1}, \mathbf{p}_1). \end{aligned}$$

Output the join token $\tau_\iota = (\mathrm{SK}_{\iota_0}, \mathrm{SK}_{\iota_1})$.

$M_{\mathbf{T_0}\to\mathbf{T_1}} \leftarrow \mathsf{Join}(\mathbf{C_0}, \mathbf{C_1}, \tau_\iota)$**:** Parse $\tau_\iota = (\mathrm{SK}_{\iota_0}, \mathrm{SK}_{\iota_1})$ and remove the ABE-blinding for all rows matching the specified filtering attributes ι_0, ι_1 as follows:

$$\textbf{toks} = \{t^i \mid \exists\, \mathrm{encToken}^i \in \mathbf{C_0} : t^i = \mathsf{ABE\text{-}Dec}(\mathrm{SK}_{\iota_0}, \mathrm{encToken}^i)\}$$
$$\textbf{ciphs} = \{c^j \mid \exists\, \mathrm{encCiph}^j \in \mathbf{C_1} : c^j = \mathsf{ABE\text{-}Dec}(\mathrm{SK}_{\iota_1}, \mathrm{encCiph}^j)\}$$

If either **toks** or **ciphs** is empty, then the equi-join result is empty, hence the algorithm returns $\perp$. Otherwise, define a map $M_{\mathbf{T_0}\to\mathbf{T_1}}$ where for every t_i search the matching ciphertexts, that is,

$$M[i]_{\mathbf{T_0}\to\mathbf{T_1}} = \{j : c^j \in \textbf{ciphs} \text{ with } \mathsf{SSE\text{-}Match}(c^j, t^i) = 1 \text{ for } t^i = \textbf{toks}[i]\}$$

Finally, return $M_{\mathbf{T_0}\to\mathbf{T_1}}$.

Recall the assumption that table $\mathbf{T_0}$ has its primary keys as join values, hence we can assume that the join column contains only *unique values*. In this construction, we assume concretely that such values are the ones in $\mathbf{T_0}$'s join column, and thus we replace them with their corresponding SSE-tokens. The reasoning behind this is that SSE-ciphertexts are always randomized, and thus multiple encryptions of the same word cannot be recognized as such without a valid SSE-token. This is not the case with SSE-tokens, since they are generated deterministically. Further, all join-values that cannot be contained in the result set for the join-query, due to not-matching the where clause, remain obfuscated by attribute-based encryption.

6.4 Evaluation

In this section we evaluate the theoretical security properties in the formal framework. Further, we examine the performance both from a theoretical point of view but also with a practical benchmark based on an implementation of our protocol.

6.4.1 Formal Security Proof

Utilizing the simulation-based security proof as discussed in Section 4.1.1 we quantify the leakage of our implementation for a secure join schemes. First, we adapt the general definition in a way fitting the framework for secure joins stated in Definition 15. Recall that the leakage given to simulator $\mathcal{S}$ must enable $\mathcal{S}$ to simulate the encryption algorithm EncRow and the algorithm for join-tokens generation $\mathsf{GenToken}$. Stating an algorithm simulating these algorithms is sufficient since the other operation with which the client can encrypt data, i.e. EncTab, is founded on EncRow.

We prove security against a non-adaptive adversary as stated in the following definition.

Definition 19. *Let* $\mathsf{SecJoin} = (\mathsf{Setup}, \mathsf{EncRow}, \mathsf{EncTab}, \mathsf{GenToken}, \mathsf{Join})$ *be a secure join scheme and* λ *the security parameter. Consider the following probabilistic experiments with a stateful adversary* $\mathcal{A}$*, a stateful simulator* $\mathcal{S}$*, and a stateful leakage function* $\mathcal{L}$*:*

$\mathbf{Real}^{\mathsf{SecJoin}}(\lambda)$**:** *the challenger runs* $\mathsf{Setup}(\lambda)$ *to generate the master key* K*. The adversary* $\mathcal{A}$ *generates a polynomial set of non-adaptive encryption requests* $r_1, \ldots, r_q$ *with* $r_j = (i, \nu, \mathbf{s})$ *and* $i \in \{0, 1\}$*.* $\mathcal{A}$ *also generates a polynomial set of non-adaptive join queries* $\iota_1, \ldots, \iota_{q'}$*.* $\mathcal{A}$ *then sends the tuples* $(r_1, \ldots, r_q)$ *and* $(\iota_1, \ldots, \iota_{q'})$ *to the challenger. For each encryption request, the challenger generates a ciphertext* $c \leftarrow \mathsf{EncRow}(K, i, \nu, \mathbf{s})$*, and for each join query, the challenger generates* $\tau_\iota \leftarrow \mathsf{GenToken}(K, \iota)$*. The challenger then returns all ciphertexts* $c_1, \ldots, c_q$ *and all join tokens* $\tau_{\iota_1}, \ldots, \tau_{\iota_{q'}}$ *to* $\mathcal{A}$*. Finally,* $\mathcal{A}$ *returns a bit* b *that is the output of the experiment.*

Ideal$^{\mathsf{SecJoin}}(\lambda)$*: the simulator sets up its internal environment. The adversary $\mathcal{A}$ generates a polynomial set of non-adaptive row encryption requests $r_1, \ldots, r_q$ with $r_j = (i, \nu, \mathbf{s})$ and $i \in \{0,1\}$. $\mathcal{A}$ also generates a polynomial set of non-adaptive join queries $\iota_1, \ldots, \iota_{q'}$. The simulator $\mathcal{S}$ is given the appropriate leakage, i.e. $\mathcal{L}\left((r_1, \ldots, r_q), (\iota_1, \ldots, \iota_{q'})\right)$. Using this leakage, $\mathcal{S}$ simulates and returns the appropriate ciphertexts $\widetilde{c_1}, \ldots, \widetilde{c_q}$ and join tokens $\widetilde{\tau_{\iota_1}}, \ldots, \widetilde{\tau_{\iota_{q'}}}$ to $\mathcal{A}$. Finally, $\mathcal{A}$ returns a bit b that is the output of the experiment.*

We say SecJoin *is $\mathcal{L}$-secure if there exists a non-uniform polynomial-sized simulator $\mathcal{S}$, so that the advantage for all probabilistic polynomial-sized (in λ) non-adaptive adversaries $\mathcal{A}$ defined as*

$$\left|\Pr\left[\mathbf{Real}^{\mathsf{SecJoin}}(\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}^{\mathsf{SecJoin}}(\lambda) = 1\right]\right|$$

is negligible in λ.

In order to benefit from all further progress achieved in both active research topics, attribute-based encryption and searchable symmetric encryption, we use these cryptographic tools as black boxes in our security proof. As a result, the information leakage we define for the security proof may be too pessimistic for some possible tools in the sense that the quantified leakage is an over-estimation. Further, we emphasize that we prove security against non-adaptive attackers due to our black box applications of attribute based encryption. While it is relatively easy to build efficient SSE systems that are secure even against adaptive attackers as shown in Chapter 5, the construction of ABE schemes that are secure against adaptive attackers is really challenging as discussed by Boneh et al. [28]. The flexibility in the choice of actual implementations for attribute-based encryption and searchable symmetric encryption results in several possible tradeoffs between performance and information leakage. Depending on the offered properties of the tools actually implementing our construction, the information leakage might be lower than the leakage we state in the following; we give a more detailed discussion for attribute-hiding ABE schemes in Section 6.5 with reduced information leakage.

We will now proceed to show how to simulate the output for a tuple of encryption requests $(r_1, ..., r_q)$ and join queries $(\iota_1, \ldots, \iota_{q'})$ given the following leakage:

$$\mathcal{L}\left((r_1, ..., r_q), (\iota_1, ..., \iota_{q'})\right) = \left(\mathbf{b}, \rho(r_1, \ldots, r_q), \mathtt{acc}, ID_0, ID_1\right).$$

Here, $\mathbf{b} = \{0,1\}^q$ is the indicator bit string for all row encryption queries, that is, for each encryption query r_i submitted by the adversary $\mathbf{b}_i$ indicates if the join-value submitted in the i-th query is a primary or a foreign key. Let $q = n_0 + n_1$, where n_0 is the number of rows containing primary keys for $\mathbf{T_0}$, i.e. the number of zeros in $\mathbf{b}$; analogously n_1 is the number of foreign keys for $\mathbf{T_1}$, i.e. the number of ones in $\mathbf{b}$. We denote $\mathbf{s}[i][j]$ as the j-th attribute value submitted in encryption query r_i. Further, we define $\rho(r_1, \ldots, r_q) = (\rho_0, \rho_1)$ as the attribute pattern, where ρ_{b} indicates which predicates are shared for one specific column in what rows in the corresponding table $\mathbf{T}_{\mathsf{b}}$. That is, we can imagine ρ_0 as a table with n_0 rows and with $\mathtt{len}(\mathbf{s})$ columns and denote $\rho_0[i][j]$ as the j-th attribute identifier in column i. As also discussed in Section 5.4.1 regarding the search history of searchable symmetric encryption, these attribute identifiers allow the adversary to distinguish between different attributes contained in the same column, however, the adversary cannot extract the actual attribute values.

Further, we re-interpret the scenario of SSE handling encrypted documents to our scenario handling exactly one encrypted join value per table row. Note that – compared to the definition stated by Curtmola et al. and the construction given in the previous Chapter 5 – we assume a non-adaptive adversary hence we can omit the specific point of time t. Recall that the access pattern $\mathtt{acc}$ consists of sets $\mathbf{f}(w) = \{ID(f_j) \mid w \in f_j \text{ and } f_j \in \mathbf{f}\}$ containing the documents matching the keyword w. In our case, $\mathbf{f}(w_i)$ in

$\mathtt{acc}$ will contain the identifiers of those rows whose join values match exactly the value w_i and that have been decrypted with ABE. Since we assume SSE-tokenization for unique primary keys we avoid multiple occurrences of the same deterministically generated SSE-token allowing us to omit the search pattern in the leakage. Additionally, we SSE-encrypt single join values, hence $\mathtt{len}(f_j) = 1$ is true for all j and we can omit it in the leakage definition.

Finally, let ID_0 and ID_1 represent all attribute identifiers specified in the join queries for table $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. More specifically, for every join query ι_i, ID_0 contains an attribute identifier for all attribute values specified for $\mathbf{T_0}$ in the where clause. These attribute identifiers are consistent with the attribute identifiers stated in ρ_0, i.e. if a constraint is given for a row attribute value that has also been used in one encryption query it contains the same identifier, whereas novel attributes values contain different identifiers. We denote $ID_0[i][j]$ as the attribute identifier for column j in the i-th join query; if no constraint for the j-th column is stated in query ι_i set $ID_0[i][j]$ to special symbol $\perp$. ID_1 is similarly constructed, referring to columns and attribute identifiers specified for table $\mathbf{T_1}$.

Theorem 2. *If the used SSE scheme is $(\mathcal{L}_1^{\mathsf{SSE}}, \mathcal{L}_2^{\mathsf{SSE}})$-secure as stated in Theorem 1, the used ABE scheme is selectively secure against chosen-plaintext attacks according to Definition 18 and H is a pseudorandom function, then the secure join scheme* SecJoin *as described in Section 6.3.3 is $\mathcal{L}$-secure against non-adaptive adversaries.*

Proof. We describe a polynomial-time simulator $\mathcal{S}$ for which the advantage of any PPT attacker $\mathcal{A}$ to distinguish between the output of $\mathbf{Real}^{\mathsf{SecJoin}}$ and $\mathbf{Ideal}^{\mathsf{SecJoin}}$ is negligible. The stated simulator non-adaptively simulates the encrypted tables and join tokens given the additional information defined by the leakage functions.

Setting up the environment: Given λ, the simulator $\mathcal{S}$ initializes the following data structures:

- Empty tables $\widetilde{\mathbf{C_0}}$ and $\widetilde{\mathbf{C_1}}$, which will contain the simulated join values. These are returned as simulated versions of the encrypted tables
- Empty sets X_0, X_1 and $\widetilde{X}$, which will contain ABE-keys (X_0 and X_1) and join tokens $\widetilde{X}$. The latter will be returned as the set of simulated join tokens
- Empty tables A_0, A_1, that map attribute identifiers to randomly sampled values. Here table A_b contains all attribute identifiers for both, attribute identifiers ID_b leaked for join queries and the attribute pattern ρ_b.
- $K_{\mathsf{SSE}} \leftarrow \mathsf{SSE\text{-}Setup}(\lambda)$
- $K_{\mathsf{ABE}0} \leftarrow \mathsf{ABE\text{-}Setup}(\lambda)$ and $K_{\mathsf{ABE}1} \leftarrow \mathsf{ABE\text{-}Setup}(\lambda)$

Simulating encrypted rows: Given leakage $\mathcal{L} = (\mathbf{b}, \rho(r_1, \ldots, r_q), \mathsf{acc}, ID_0, ID_1)$ the simulator $\mathcal{S}$ proceeds as follows:

1. First simulate all SSE-tokenized unique primary keys and program the foreign keys that are matched in at least one join query to be consistent.
   For each row $i \in [1, n_0]$:
   - $s_i \overset{\$}{\leftarrow} \{0,1\}^\lambda$
   - $t_i \leftarrow \mathsf{SSE\text{-}Token}(K_{\mathsf{SSE}}, s_i)$
   - $\widetilde{\mathbf{C_0}}[i] = t_i$

- $\forall j \in \mathbf{f}(w_i)$ where $\mathbf{f}(w_i)$ is part of $\mathtt{acc}$:
  - $c_j \leftarrow \mathsf{SSE\text{-}Enc}(K_{\mathsf{SSE}}, s_i)$
  - $\widetilde{\mathbf{C_1}}[j] = c_j$

2. Next simulate all remaining SSE-ciphertexts that are never unveiled for any join query or whose matching SSE-token are not unveiled. All rows that have successfully been joined in at least one join query have already been filled in the previous step.
   For each row $i \in [1, n_1]$: if $\widetilde{\mathbf{C_1}}[i]$ is undefined, do:
   - $s_i \xleftarrow{\$} \{0,1\}^\lambda$
   - $\widetilde{\mathbf{C_1}}[i] = c_i$
3. Now ABE-encrypt the entries based on attribute pattern ρ_0. Assuming that table $\mathbf{T_0}$ has n_0 rows as highlighted in $\mathbf{b}$ do the following:
   For each row $i \in [1, n_0]$:
   - $\widetilde{\mathbf{s}_i} = \{\}$
   - Extract attribute identifiers for the i-th encryption query: $\mathbf{r}_i = \rho_0[i]$
   - For each attribute identifier $r_j \in \mathbf{r}_i$:
     If this attribute identifier r_j has never been used before, i.e. $A_0[j][r_j]$ is undefined:
     - $s \xleftarrow{\$} \{0,1\}^\lambda$
     - $A_0[j][r_j] = s$
   - $\widetilde{\mathbf{s}_i} = \widetilde{\mathbf{s}_i} \cup A_0[j][r_j]$
   - Retrieve the simulated SSE-token $t_i = \widetilde{\mathbf{C_0}}[i]$
   - $\widetilde{\mathbf{C_0}}[i] \leftarrow \mathsf{ABE\text{-}Enc}(K_{\mathsf{ABE}0}, t_i, \widetilde{\mathbf{s}_i})$
4. Analogously to previous Step 3, encrypt entries in table $\widetilde{\mathbf{C_1}}$ using the attribute pattern ρ_1.

Simulating join tokens: Based on the attribute identifiers for the predicate conditions ID_0 and ID_1 the simulator creates consistent join tokens as follows:

1. Simulate the ABE-keys that are generated for $\mathbf{C_0}$
   For each query $i \in [1, q']$:
   - Extract attribute identifiers for constraints stated in the i-th join query: $\mathbf{q}_i = ID_0[i]$
   - $\widetilde{\iota_i} = \{\}$
   - For each constraint $q_j \in \mathbf{q}_i$ that is not $\perp$:
     If this attribute identifier q_j has never been used before, i.e. $A_0[j][q_j]$ is undefined:
     - $s \xleftarrow{\$} \{0,1\}^\lambda$
     - $A_0[j][q_j] = s$
   - $\widetilde{\iota_i} = \widetilde{\iota_i} \cup A_0[j][q_j]$
   - Generate the corresponding ABE-key $k_{i,0} \leftarrow \mathsf{ABE\text{-}Key}(K_{\mathsf{ABE}0}, \widetilde{\iota_i})$ and set $X_0[i] = k_{i,0}$
2. Analogously to Step 1, generate ABE-keys compatible with $\mathbf{C_1}$ and store them in X_1.
3. The overall simulated join token for each query $\widetilde{\tau_{\iota_i}} = (X_0[i], X_1[i])$ is added to $\widetilde{X}[i] = \widetilde{\tau_{\iota_i}}$.

The indistinguishability from the real encrypted rows containing primary keys and the simulated ones in Step 1 follows from the $(\mathcal{L}_1^{\mathsf{SSE}}, \mathcal{L}_2^{\mathsf{SSE}})$-security provided by SSE. Specifically, each primary key is assumed to appear exactly once hence each search token is unique either for the real protocol or the simulated version resulting in the same search pattern leakage induced by SSE. Also in this step simulating the encrypted rows containing foreign keys for all rows that match (at least) one join-query are filled with the same random values used for the primary keys instead of real join values. Again, this is indistinguishable for $\mathcal{A}$ due to the SSE $(\mathcal{L}_1^{\mathsf{SSE}}, \mathcal{L}_2^{\mathsf{SSE}})$-security; more specifically the access pattern leakage is the same for the real and the simulated protocol. The difference between the real protocol and the simulation in Step 2 simulating foreign keys with no matching primary key as random values instead of actual foreign keys is indistinguishable for $\mathcal{A}$ either due to security offered by ABE or by SSE. In the first case, a matching ABE-key is never unveiled hence the random values remain ABE encrypted; they are indistinguishable due to selective CPA-security provided by ABE. In the second case, no matching SSE-token is known to the adversary since the corresponding row containing this token either is never ABE-decrypted or is not contained in the encrypted table; in these cases the SSE ciphertexts of random values are indistinguishable due to $(\mathcal{L}_1^{\mathsf{SSE}}, \mathcal{L}_2^{\mathsf{SSE}})$-security provided by SSE. The difference between the real protocol and the simulation of masked attribute values used for encrypting the rows in Step 3 is indistinguishable for $\mathcal{A}$. Here $\mathcal{S}$ samples random values s_i instead of calculating the actual masked attribute values; this is indistinguishable since H is assumed to be a pseudorandom function.

Simulation of the join tokens in Step 1 for fake attribute predicates that are consistent with the join-queries is indistinguishable due to the pseudorandomness of H. Masked values are sampled consistently based on the attribute identifiers ID_i or are generated using fresh randomness, hence the ABE-keys are correctly simulated join-tokens. □

6.4.2 Amortized Analysis

Assuming n_0 matching rows for the filtering query specified in the where clause for attributes in $\mathbf{T_0}$, the implementation has an upper bound of n_0 $\mathsf{ABE\text{-}Dec}$ calls (upper bounded since some might already be unblinded due to previous queries). Analogously, assuming n_1 matching rows for the filtering query specified in the where clause for attributes in $\mathbf{T_1}$, the implementation has an upper bound of n_1 $\mathsf{ABE\text{-}Dec}$ calls. Based on the unveiled SSE values, the join mapping is computed as follows: For each of the n_0 SSE-tokens t_i, the n_1 SSE-ciphertexts are scanned for matches calling $\mathsf{SSE\text{-}Match}$, resulting in an upper bound of $n_0 \cdot n_1$ calls of $\mathsf{SSE\text{-}Match}$. Hence, the overall runtime is bounded by $n_0 + n_1 + n_0 n_1 \in O(n_0 n_1)$.

Although the theoretical analysis indicates that the runtime is dominated by the number of performed SSE-comparisons, the practical runtime is heavily influenced by the number of performed ABE-decryption operations. While the implementation of $\mathsf{SSE\text{-}Match}$ operations is based on fast symmetric cryptographic operations such as pseudorandom functions, the implementation of $\mathsf{ABE\text{-}Dec}$ operations requires computationally expensive pairing-based operations as elaborated in more details in the following section.

6.4.3 Practical Benchmark

We have implemented a prototype of our scheme and have tested its efficiency for both parties, the trusted client side and the honest-but-curious server side.

On the client side, we evaluate the performance for the initial encryption step in Paragraph "Encryption" and the token generation in Paragraph "Token Generation". For the server side, we measure the actual join execution time preformed in the untrusted environment as described in Paragraph "Trace Evaluation". Initially completely obfuscated by the encryption step of our scheme, entries are gradually ABE-decrypted with every passing join query. As a result, the performance impact of ABE-decryption operations decreases with time, and queries with similar result sets tend to have decreasing cryptographic overhead reducing the join computation time.

The following experiments were implemented in Java 8. All operations, i.e. client and server, were executed on a machine with 32 processor cores, each a 64-bit Intel Xeon E5-2670 @2.60 GHz, with 240GB RAM and running SUSE Linux Enterprise Server 11. In the operations involving a client and a server, the server used a MySQL Server 5.7 instance for storage of the encrypted data. Moreover, our implementation makes use of the following libraries: The Secure Computation API (SCAPI)[2] for all symmetric cryptographic primitives (e.g. AES, SHA-X, HMAC, PRFs, etc), and jPBC for all group and pairing-based operations. Both are available as Java native code.

As SSE scheme we used a variation of the construction from Chapter 5 but omitted the inverted index, where each "file" is a join value. As PRF for the SSE scheme we used a CBC-MAC-based PRF with AES as building block. As KP-ABE scheme we used the one proposed by Hohenberger and Waters [80], with PBC's symmetric Type A pairings[3] over a group with a 160-bit-long prime number of elements, and a CBC-MAC-based PRF as the hash function necessary for the support of large universes.

Encryption

In order to test the efficiency and scalability of the EncTab implementations, we generated random sets of rows, with different number of attribute columns, ranging from 3 to 20. For every row we assumed it as part of $\mathbf{T_1}$, hence SSE-encrypted each join value followed by ABE-encryption of the resulting ciphertext with the other row values as attributes. We evaluated SSE-encryption and not tokenization, since the encryption operation contains the generation of a corresponding token as described in the AddToken algorithm in Section 5.3 Thus, SSE-encryption contains extra work compared to simple tokenization, hence we evaluate the worst-case situation for table encryption. Recall that SSE-operations are all symmetric encryption, and thus require small computational effort.

The results of these tests are given in Figure 6.2. It is evident that the performance of the encryption is linearly correlated with both the number of attribute columns ascribed to every ABE-ciphertext as well as with the number of rows to be encrypted. These presented tests were executed purely and sequentially in Java, and only take into account the computational effort for the client to execute the necessary SSE and ABE operations, and do not include any transmission costs or I/O overhead, which would be observed when submitting the encrypted data to a SQL server. As such, it can be interpreted as the computational effort invested by a client into the encryption of a join column before outsourcing it. We emphasize that this process can also be easily parallelized.

Key Generation

Once the client has finished encrypting the table rows and outsourcing them, join tokens are generated by the client and transferred to the server to delegate the join operation. In order to do so, the specified query's `WHERE` clause is parsed into two ABE-keys which are sent to the server. For synthetically generated data, we measured the performance of generating a single ABE-key with varying number of restrictions (i.e. attributes), the results are presented in Figure 6.3. The key generation does not pose a serious challenge to any modern processor and can thus be efficiently computed within a reasonable amount of time. Furthermore, this effort is linear in the number of conditions placed by the client in the where clause.

[2] https://cyber.biu.ac.il/scapi/
[3] https://crypto.stanford.edu/pbc/manual/ch08s03.html

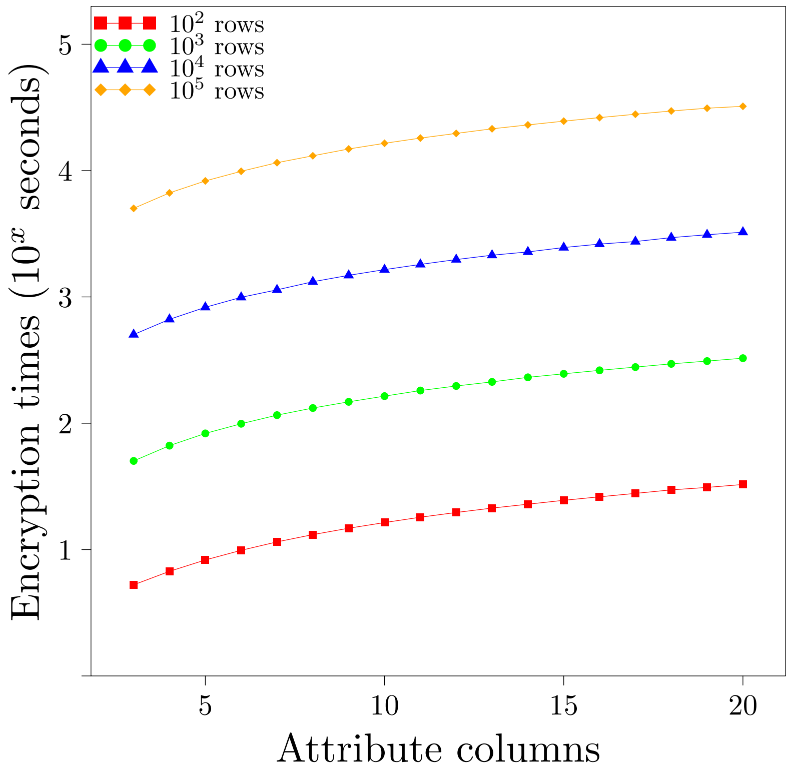


Figure 6.2: Encryption times with varying rows and attribute columns

Trace Evaluation

The evaluation on the server side is based on data produced by the TPC-H Benchmark[4] for analytical queries. Using this benchmark's data generator with a scaling factor of 0.1, we took the table `PART` (20,000 rows and 6 attribute columns) with its primary key `P_PARTKEY` and the table `LINEITEM` (600,000 rows and 8 attribute columns) with the foreign key `L_PARTKEY` as our test tables. After encrypting them with our secure joins scheme (cf. Section 6.4.3) a random trace of join executions was generated. Iterating over said trace, the client parsed each join-query into the corresponding join-token (cf. Section 6.4.3) and sent them to the server (running our protocol in Java, storing the databases in MySQL). Given this join-token, the secure join operation was computed and evaluated as discussed in the following.

In our tests we assume that the server is able to quickly identify the rows satisfying the where clause. Thus the server proceeds to ABE-decrypt the identified rows if necessary, and compares the underlying SSE-values for the join computation. In this evaluation scenario, we took advantage of the 32 processors available in the test machine and parallelized internally each join query. This was done in such a way that first, the (Java) server retrieved all rows from the (MySQL) tables $\mathbf{T_0}$ and $\mathbf{T_1}$ matching the corresponding restrictions placed in the where clause. Once retrieved, all ABE-decryption operations were executed in parallel. Next, a "full join" was built between all matching rows of both tables, and the resulting set of rows was evenly distributed among the available threads. Each thread then proceeded to compute a "local" result set, which was then returned to the main thread once the computation was finished, so that all local results could be combined in a global result set. Finally, all values that needed ABE-decryption operations were replaced with their underlying SSE-values in the corresponding MySQL tables, and the result set of the join operation returned to the client.

[4] `http://www.tpc.org/tpch/`

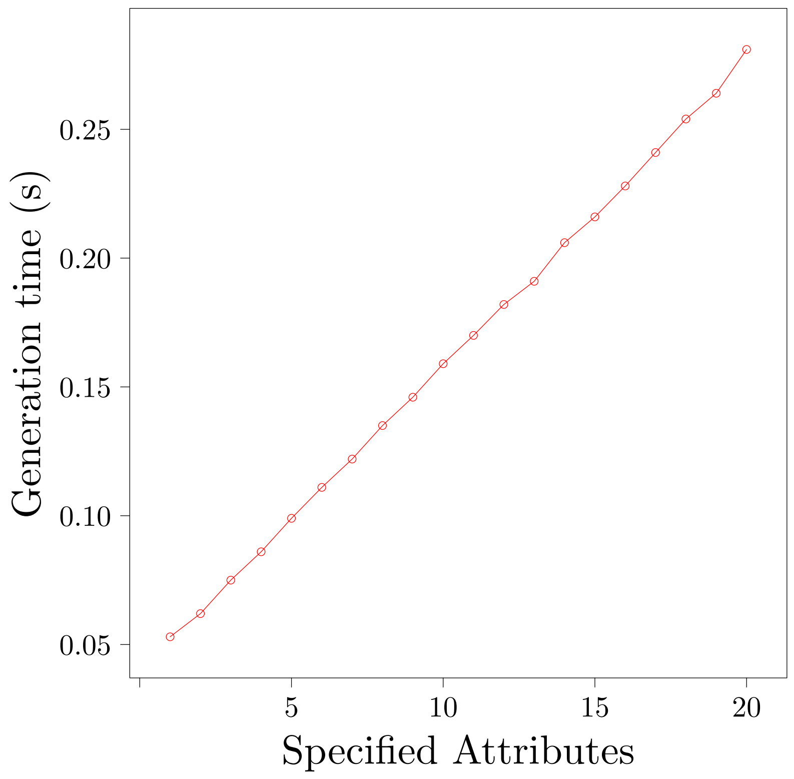
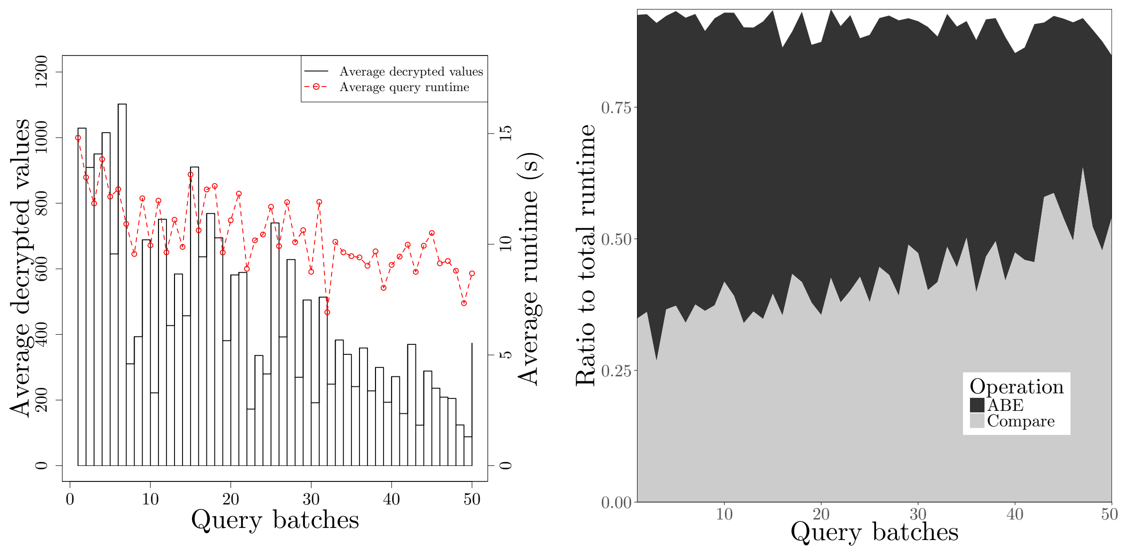


Figure 6.3: ABE-key generation times with varying attribute restrictions

The results of executing a trace with 10^3 join operations, with the server acting as specified before, is given in Figure 6.4a. For the sake of readability, we took the average runtime results from every 20 consecutive join queries, also referred to as "batch", and plotted them in red dashed lines, whereas the bars represent the averaged number of ABE-values that needed to be decrypted per join batch. As we can see, in time (implicit in the x-axis) both plots tend to sink, since the queries will increasingly need to ABE-decrypt less values. This results in a lower average query runtime as highlighted in Figure 6.4a. Since the impact of ABE lessens in time, the dominating factor in later sets of queries (i.e. queries executed towards the end of the trace) is the number of necessary SSE-comparisons. This effect can be explicitly observed in Figure 6.4b, which depicts (as before, averaged per batch) the ratios of the time invested by a single join query in executing ABE and SSE operations, compared to the operation's total runtime. As we can see, when starting the trace's execution, a join query invests close to 30% of its execution time on SSE-comparisons, and most of the rest is spent in ABE operations. This is contrasted with queries in later stages of the trace, where SSE comparisons take up more than 50% of the execution time, up to even 60%. Since SSE-operations are much more efficient than ABE-operations, this results in lower runtime characteristics for the queries executed later in the trace.

6.5 Tradeoff between Runtime and Predicate Security

Assuming the row attributes to be sensitive the distinguishability of different row attributes might be a security risk, e.g. frequency analysis on the row attributes is possible for our construction given in Section 6.3. Recall that this attack vector exists due to the application of KP-ABE that does provide attribute security by default. This shortcoming has been identified by Katz et al. [86] resulting in a stronger ABE



(a) Average decrypted values and runtime for a trace with 10^3 joins queries.

(b) Average ratio between processing time for ABE-decryption and SSE-operations.

Figure 6.4: Evaluation results for batches of 20 join queries.

construction with the additional security property called attribute-hiding. We can swap the general ABE encryption scheme with one providing the additional attribute-hiding property allowing us to simplify the construction. Compared to the construction given in Section 6.3 one can omit the hash function H for blinding the attribute rows. However, the additional security property induces additional computation and storage overhead: The decryption algorithm without explicit attribute-privacy proposed by Hohenberger and Waters requires a constant number of pairing operations. In contrast, the decryption algorithm proposed by Katz et al. with explicit attribute-privacy requires n pairing operations where n is the number of attributes used for encryption.

Given a searchable symmetric encryption scheme $\mathsf{SSE} = (\mathsf{SSE\text{-}Setup}, \mathsf{SSE\text{-}Enc}, \mathsf{SSE\text{-}Token}, \mathsf{SSE\text{-}Match})$ (as described in Definition 9) and a key-policy attribute-based encryption scheme (as described in Definition 17) $\mathsf{ABE} = (\mathsf{ABE\text{-}Setup}, \mathsf{ABE\text{-}Enc}, \mathsf{ABE\text{-}Key}, \mathsf{ABE\text{-}Dec})$ with the *attribute-hiding property* we modify the Setup, the EncRow and the GenToken operations from Section 6.3 as follows:

$K \leftarrow \mathsf{Setup}(1^\lambda)$: Let λ be the security parameter. Run the following algorithms:

$$\begin{aligned} K_{\mathsf{SSE}} &\leftarrow \mathsf{SSE\text{-}Setup}(1^\lambda) \\ K_{\mathsf{ABE}0} &\leftarrow \mathsf{ABE\text{-}Setup}(1^\lambda) \\ K_{\mathsf{ABE}1} &\leftarrow \mathsf{ABE\text{-}Setup}(1^\lambda) \end{aligned}$$

and return $K = (K_{\mathsf{SSE}}, K_{\mathsf{ABE}0}, K_{\mathsf{ABE}1})$.

$c \leftarrow \mathsf{EncRow}(K, \mathsf{b}, \nu, \mathsf{s})$: Parse $K = (K_{\mathsf{SSE}}, K_{\mathsf{ABE}0}, K_{\mathsf{ABE}1})$ and create an SSE-value for the join value ν and encrypt it using attribute-hiding ABE-encryption under the attributes s

$$\mathrm{sseVal} \leftarrow \begin{cases} \mathsf{SSE\text{-}Token}(K_{\mathsf{SSE}}, \nu) & \text{if } \mathsf{b} = 0 \\ \mathsf{SSE\text{-}Enc}(K_{\mathsf{SSE}}, \nu) & \text{if } \mathsf{b} = 1 \end{cases}$$

$$c \leftarrow \mathsf{ABE\text{-}Enc}(K_{\mathsf{ABE}_\mathsf{b}}, \mathrm{sseVal}, \mathsf{s})$$

and return c.

$\tau_\iota \leftarrow \mathsf{GenToken}(K, \iota)$**:** Let $\iota = (\iota_0, \iota_1)$ be the attributes in the where clause corresponding to columns in tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. Generate the corresponding ABE-keys:

$$\mathrm{SK}_{\iota_0} \leftarrow \mathsf{ABE\text{-}Key}(K_{\mathsf{ABE0}}, \iota_0)$$
$$\mathrm{SK}_{\iota_1} \leftarrow \mathsf{ABE\text{-}Key}(K_{\mathsf{ABE1}}, \iota_1)$$

and return these keys as join token $\tau_\iota = (\mathrm{SK}_{\iota_0}, \mathrm{SK}_{\iota_1})$.

Following Definition 19, this construction does not leak the equality of attributes used for ABE-encryption and results in the following reduced leakage function:

$$\mathcal{L}\left((r_1, ..., r_q), (\iota_1, ..., \iota_{q'})\right) = \left(\mathbf{b}, \mathsf{acc}, ID_0, ID_1, R_0, R_1\right).$$

Again, $\mathbf{b}$ indicates if the row encryption query contains a primary key or a foreign key andacc denotes the access pattern matching queried keywords. Recall that a join query can be split as $\iota = (\iota_0, \iota_1)$, where ι_0 and ι_1 represent the restrictions placed in ι regarding tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively. Using definitions from above, let ID_0 and ID_1 be the information leaked through the join queries regarding tables $\mathbf{T_0}$ and $\mathbf{T_1}$, respectively.

While in our previous construction the simulator could extract the row IDs matching the constraints specified in one join query based on the given leakage ID_0 and ρ_0 this is not possible here since ρ_0 is hidden due to attribute-privacy (analogously the same is true ID_1 and ρ_1, respectively). Hence we modify the leakage function with result sets R_0 mapping for each join query ι_i the set of row IDs from table $\mathbf{T_0}$ matched by the restrictions stated in ι_i (and analogously R_1 for table $\mathbf{T_1}$).

Theorem 3. *If the used SSE scheme is $(\mathcal{L}_1^{\mathsf{SSE}}, \mathcal{L}_2^{\mathsf{SSE}})$-secure, the used ABE scheme provides attribute-privacy and is selectively secure against chosen-plaintext attacks then the secure join scheme* $\mathsf{SecJoin}$ *based on the description given in Section 6.3.3 with the modifications described in this section is $\mathcal{L}$-secure against non-adaptive adversaries.*

Proof. The proof is based on the same idea as Theorem 2, however, simulator $\mathcal{S}$ has no access to the complete attribute pattern. Thus, the simulator generates the join tokens first and encrypts the row attributes afterwards consistently based on attributes leaked by these join tokens. Due to the mutual simulation of encrypted rows and join tokens the whole simulation step is given in one paragraph. We will now proceed to show how to non-adaptively simulate a set of encryption requests and join queries given this leakage.

Setting up the environment: Given λ, the simulator $\mathcal{S}$ initializes:

- Empty tables $\widetilde{\mathbf{C_0}}$ and $\widetilde{\mathbf{C_1}}$, which will map row IDs to the encrypted values. These are returned as simulated versions of the encrypted tables.
- Empty sets X_0, X_1 and $\widetilde{X}$, which will contain ABE-keys (X_0 and X_1) and join tokens $\widetilde{X}$. The latter will be returned as the set of simulated join tokens.
- Empty tables $\widetilde{\mathbf{T_0}}$, $\widetilde{\mathbf{T_1}}$, which map row IDs to sets of attributes.
- Empty tables A_0 and A_1, which map attribute IDs to bitstrings.
- Initialize $K_{\mathsf{SSE}} \leftarrow \mathsf{SSE\text{-}Setup}(\lambda)$.
- Initialize $K_{\mathsf{ABE0}} \leftarrow \mathsf{ABE\text{-}Setup}(\lambda)$ and $K_{\mathsf{ABE1}} \leftarrow \mathsf{ABE\text{-}Setup}(\lambda)$.

Simulation: Given the leakage from above, $\mathcal{S}$ proceeds as follows:

1. First simulate all SSE-tokenized unique primary keys and program the foreign keys that are matched in at least one join query to be consistent.
   For each row $i \in [1, n_0]$, do:
   - $s_i \xleftarrow{\$} \{0,1\}^\lambda$
   - $t_i \leftarrow \mathsf{SSE\text{-}Token}(K_{\mathsf{SSE}}, s_i)$
   - $\widetilde{\mathbf{C_0}}[i] = t_i$
   - $\forall j \in \mathbf{f}(w_i)$ where $\mathbf{f}(w_i)$ is part of $\texttt{acc}$:
     - $c_j \leftarrow \mathsf{SSE\text{-}Enc}(K_{\mathsf{SSE}}, s_i)$
     - $\widetilde{\mathbf{C_1}}[j] = c_j$
2. Next simulate all remaining SSE-ciphertexts that are never unveiled for any join query or whose matching SSE-token are not unveiled. All rows that have successfully been joined in at least one join query are already filled in the previous step.
   For each row $i \in [1, n_1]$ if $\widetilde{\mathbf{C_1}}[i]$ is undefined, do:
   - $s_i \xleftarrow{\$} \{0,1\}^\lambda$
   - $c_i \leftarrow \mathsf{SSE\text{-}Enc}(K_{\mathsf{SSE}}, s_i)$
   - $\widetilde{\mathbf{C_1}}[i] = c_i$
3. Simulate the ABE-keys that are generated for $\mathbf{C_0}$ while also filling table $\widetilde{\mathbf{T_0}}$ and maintaining attribute consistency.
   For each query $i \in [1, q']$:
   - Extract attribute identifiers for constraints stated in the i-th join query: $\mathbf{q}_i = ID_0[i]$
   - $\widetilde{\iota_i} = \{\}$
   - For each constraint id $q_j \in \mathbf{q}_i$ that is not $\perp$:
     If this attribute identifier q_j has never been used before, i.e. $A_0[j][q_j]$ is undefined:
     - $s \xleftarrow{\$} \{0,1\}^\lambda$
     - $A_0[j][q_j] = s$
   - $\widetilde{\iota_i} = \widetilde{\iota_i} \cup A_0[j][q_j]$
   - Fill all matching rows with the current attribute identifier:
     $\forall r \in R_0[i]$: $\widetilde{\mathbf{T_0}}[r][j] = A_0[j][q_j]$
   - Generate the corresponding ABE-key $k_{i,0} \leftarrow \mathsf{ABE\text{-}Key}(K_{\mathsf{ABE}0}, \widetilde{\iota_i})$ and set $X_0[i] = k_{i,0}$
4. Analogously to Step 3, generate ABE-keys from the leakage in ID_1 for table $\widetilde{\mathbf{C_1}}$ using key $K_{\mathsf{ABE}1}$ and store them in X_1, while also filling table $\widetilde{\mathbf{T_1}}$ and maintaining attribute consistency with table A_1.
5. Fill the remaining cells in $\widetilde{\mathbf{T_0}}$. For each row $i \in [1, n_0]$:
   - For each attribute value $j \in \texttt{len}\left(\widetilde{\mathbf{T_0}}[i]\right)$:
   - If $\widetilde{\mathbf{T_0}}[i][j]$ is undefined:
     - $\widetilde{\mathbf{T_0}}[i][j] \xleftarrow{\$} \{0,1\}^\lambda$
   - Retrieve the simulated SSE-token $t_i = \widetilde{\mathbf{C_0}}[i]$

- Encrypt this token with simulated attributes $\widetilde{\mathbf{C_0}}[i] \leftarrow \mathsf{ABE\text{-}Enc}(K_{\mathsf{ABE}0}, t_i, \widetilde{\mathbf{T_0}})$

6. Analogously to Step 5, encrypt the entries in table $\widetilde{\mathbf{C_1}}$ using the data stored in $\widetilde{\mathbf{T_1}}$ with the key $K_{\mathsf{ABE}1}$.
7. $\forall i \in [1, q']$: $\widetilde{\tau_{\iota_i}} := (X_0[i], X_1[i])$ and set $\widetilde{X}[i] = \widetilde{\tau_{\iota_i}}$
8. Return $(\widetilde{\mathbf{C_0}}, \widetilde{\mathbf{C_1}}, \widetilde{X})$

As in the previous proof, all rows in Step 1 that match (at least) one join-query are filled with random values that match the characteristics leaked by the SSE leakage; all remaining rows are filled in Step 2 This is not distinguishable by any attacker due to $(\mathcal{L}_1^{\mathsf{SSE}}, \mathcal{L}_2^{\mathsf{SSE}})$ security provided by SSE. In Step 3 and Step 4 $\mathcal{S}$ creates attribute predicates that are consistent with the join-queries and their attribute patterns, i.e. the values are added to the correct columns and the corresponding rows share the same attributes. Further, ABE-keys are created for such fake predicates forming the simulated join-tokens. This is not distinguishable by any attacker due to the provided ABE security for ABE-keys and their attributes. In Steps 5 and Step 6 the simulator ABE-encrypts the two simulated tables with values that are consistent with the queries, as generated in the previous step, while also filling remaining empty attribute predicate cells with random values, i.e. all values that contain attributes matching no join-query. This is not distinguishable by any attacker due to the provided ABE security for ciphertexts. □

6.6 Summary

In this section we presented a novel approach for cryptographically protected database joins with fine granular security. While schemes based on private set intersection require uniqueness of the join-values per database table and alternative solutions without this constraint only provide all-or-nothing security, our scheme provides full flexibility and advanced protection. Most join queries on databases contain additional filtering predicates, all previous schemes supporting secure joins unveil the complete inner join result, leaking unnecessary information. Taking the additional filtering process in consideration already during the encryption phase, our construction minimizes the information leakage of the join-operation by only unveiling the join values actually involved in the computation of the result set. Our constructions is based on a combination of searchable symmetric encryption and key-policy attribute-based encryption, both applied as generic black boxes. Due to this black-box construction, our solution benefits from all further improvements in these active research areas. Based on this black-box construction we proved two different leakage characteristics that might be applicable for different use cases.

7 Range Queries

In this chapter we present a protocol for privacy-preserving range queries on encrypted data. This functionality enables the database client to delegate range filtering to the untrusted database server. The DBMS on such server is then enabled to filter for all encrypted values that fall within the range specified by the client. In the following Section 7.1 we give an introduction, discuss the problem in more details and give a general interface that offers the functionality for range queries on encrypted data we strive to fulfill. An overview of related work addressing range queries on encrypted data is presented in Section 7.2. In Section 7.3 we review the cryptographic tools we use for our construction. Further, we describe the abstract idea of our solution on plaintext data and give a concrete construction how these ideas can be transferred for encrypted data utilizing the tools described before. Based on this concrete construction a detailed evaluation is given in Section 7.4 regarding both formal security and performance. Performance evaluation is done in two ways, that is, theoretical runtime numbers are given together with practical benchmark results founded on a prototypical implementation. Finally, Section 7.5 provides a summary of this chapter. The content of this chapter has been published in joint work with Florian Kerschbaum at CCSW 2016:

- Hahn, Florian ; Kerschbaum, Florian: Poly-Logarithmic Range Queries on Encrypted Data with Small Leakage. In: *Proceedings of the ACM on Cloud Computing Security Workshop*, 2016 (CCSW)

7.1 Introduction

Besides exact keyword matching, another functionality frequently required by database systems is the possibility to filter values that fall withing specified range values. For example, recall the database from Chapter 5 containing employees and their individual salary as sketched in Table 7.1. Again, the client wishes to outsource this table in encrypted form. Later, the client wishes to query all employees whose salary falls

EmpID	**Name**	**Salary**
1	Harry	4000
2	Sally	7000
3	Harry	5000
4	George	3500

Table 7.1: Example of plain database table *Emp*

within a specific range defined by the client, e.g. the corresponding SQL query retrieving all employees with a salary between 4000 and 6000 as given in Listing 7.1

```
SELECT * FROM Emp WHERE 4000 <= SALARY and SALARY <= 6000;
```

Listing 7.1: Example SQL query for range query on the Salary column.

yielding the result given in Table 7.2 on the client side.

Previous solutions adapted in real-world for encrypted databases that realize such functionality are based on order-preserving encryption (OPE) [5, 21]. An OPE scheme consists of the following three algorithms

EmpID	Name	Salary
1	Harry	4000
3	Harry	5000

Table 7.2: Plaintext result after query execution for table *Emp*

$(\mathsf{OPE\text{-}Setup}, \mathsf{OPE\text{-}Enc}, \mathsf{OPE\text{-}Dec})$ where $\mathsf{OPE\text{-}Enc}(x) \leq \mathsf{OPE\text{-}Enc}(y)$ if and only if $x \leq y$. A search index can then be constructed based on the preserved ordering relation, that is, all entries v_i are encrypted $c_i = \mathsf{OPE\text{-}Enc}(v_i)$ and then sorted by the yielded ciphertexts. The query transformation for a range $[q^{(s)}, q^{(e)}]$ is then the OPE-encryption of the range's boundary points, e.g. $(\mathsf{OPE\text{-}Enc}(4000), \mathsf{OPE\text{-}Enc}(6000))$. Given these OPE-ciphertexts to the server storing the previously sorted search index, she is able to obtain the set $\{\mathsf{OPE\text{-}Enc}(v_i) \mid \mathsf{OPE\text{-}Enc}(q^{(s)}) \leq \mathsf{OPE\text{-}Enc}(v_i) \text{ and } \mathsf{OPE\text{-}Enc}(v_i) \leq \mathsf{OPE\text{-}Enc}(q^{(e)})\}$ in logarithmic time. Particularly, the server scans the index, e.g. by executing binary search, for the boundary points $\mathsf{OPE\text{-}Enc}(q^{(s)})$ and $\mathsf{OPE\text{-}Enc}(q^{(e)})$ and all values indexed between these boundary points match the requested range query. As discussed in Section 3.2, leaking the index' complete ordering relation after the initial outsourcing step can be exploited for concrete attacks and might result in a total data breach in the worst case as demonstrated by Naveed et al. [115].

The construction given in this chapter provides the functionality for range queries but only leaks the access pattern, hence mitigates these attack vectors. Specifically, the ordering relation of all values is obfuscated initially by randomized encryption and the search for a specified range is enabled by unveiling a range token only if required, i.e. the range is queried by the client.

Our goal is a secure construction that is efficient and is scheme-agnostic about the underlying range predicate encryption scheme. The presented scheme has amortized poly-logarithmic runtime and our implementation shows the benefits of this change already after a short period of queries. Further, range predicate encryption can be implemented founded on functional encryption providing secure inner product evaluation, hence, we can profit from any performance improvements in this active research area. This flexibility is demonstrated by our prototype implementation utilizing two different functional encryption schemes for inner product evaluation, one published by Shen et al. [133] and an alternative published more recently by Bishop et al. [18].

We extend the idea applied for exact keyword queries presented in Chapter 5 to range queries where the search index is updated incrementally. While the initial search time for a range query is linear in the number of indexed values we can speed up subsequent queries requested in the future. In the first, initial search the database service provider learns the result set of the range query; given a range query in a second search request that is a subrange of the already queried range in the first step, it is sufficient to scan this previously learned result set. This downscaling of the possible search space results in a tremendous speed-up for the search operation. For every access pattern unveiled by a new range query, the server incrementally updates the encrypted search index resulting in decreased search time for values contained in this search index. On the contrary, ciphertexts that have never fallen within any queried range are not contained in any access pattern, hence, these ciphertexts are not indexed and remain semantically secure. Compared to the approach for exact keyword queries presented in Chapter 5, the approach for range queries described in this chapter causes additional complications. Particularly, any exact keyword query yields result sets that are exclusively valid for one specific keyword, rendering incremental updates for the search index straightforward. In contrast to range queries, where different queries might overlap or one queried range might be a subset of another previously queried range; for such cases, the structure of a search index supporting these variations and the incremental updating process of such search index require a more complex system design.

7.1.1 Framework

As already introduced in Section 5.1.1 the solution presented in this chapter can be extended to be applicable for files with additional filtering values, e.g. encrypted photos with their encrypted record dates as tags where the client should be able to filter for all photos recorded in a specified time period. Analogously to searchable encryption for exact keyword search, we assume each file f has a unique identifier $ID(f)$, further each file is indexed – or tagged – with at least one value point v, where v falls within a predefined domain $[0, d]$, e.g. $[0, 2^{32} - 1]$ the domain of unsigned integers used in common programming languages. We define a message $m = (v, ID(f))$ as the tuple containing the file identifier and the corresponding value point. For a fileset $\mathbf{f}$ and a search range $Q = [q^{(s)}, q^{(e)}]$ with $Q \subseteq [0, d]$ we denote $\mathbf{f}_Q$ as the collection of files that are indexed with a value v so that $q^{(s)} \leq v \leq q^{(e)}$. The set of all identifiers pointing to files that are indexed under a value $v \in Q$ is denoted as $\mathbf{I}_Q = \{ID(f_i) \mid f_i \in \mathbf{f}_Q\}$ or ID_Q. While encrypting the payload, i.e. the content of these files, does not require any special functionality besides IND-CPA security and is omitted in the following discussion, the encryption of these filtering tags with provided range filtering functionality is non-trivial. Recall from Chapter 5, that this approach can be used for encrypted and tagged files as well as for databases as motivated in the introduction of this chapter. More specifically, we strive to realize a framework for secure range queries described as follows.

In an initial step, the client creates public parameters and a master key for a desired value domain by executing SRQ-Setup. We assume the public parameters are known by all parties and omit them for the sake of simplicity in the remainder of this chapter. In the next step, a message collection is encrypted and indexed under given value points by executing SRQ-Enc; each value point is an element of the predefined value domain stated in the initial setup step. Note that this encryption step requires the complete file collection to be encrypted, that is, we assume a static database[1]. The output of this encryption algorithm, namely the encrypted search index is transferred to the database server. The client holding the master key is able to create range tokens by executing SRQ-Token. Given this range token to the server she can run SRQ-Search to filter all message identifiers associated with messages whose value points fall within the requested range. More formally, we implement a scheme for secure range queries as stated in the following definition.

Definition 20. *[Secure Range Queries Scheme] A scheme for secure range queries is a tuple of the following (probabilistic) polynomial-time algorithms* SRQ = (SRQ-Setup, SRQ-Enc, SRQ-Token, SRQ-Search) *such that:*

$K \leftarrow \mathsf{SRQ\text{-}Setup}(1^\lambda, [0, d])$ *is a probabilistic algorithm that takes a security parameter λ and value domain $[0, d]$ as input and outputs a master key K.*

$\gamma \leftarrow \mathsf{SRQ\text{-}Enc}(K, \mathbf{M})$ *is a probabilistic algorithm that takes a master key K and a message collection $\mathbf{M}$ as input. Each message $m_i \in \mathbf{M}$ is a tuple $m_i = (v_i, ID(f_i))$ containing a file identifier and a value point. A search index γ is output.*

$\tau_Q \leftarrow \mathsf{SRQ\text{-}Token}(K, Q)$ *is a probabilistic algorithm that takes master key K and range $Q = [q^{(s)}, q^{(e)}]$ with $Q \subseteq [0, d]$ as input and outputs a search token τ_Q for range Q.*

$\boldsymbol{I}_Q, \gamma' \leftarrow \mathsf{SRQ\text{-}Search}(\tau_Q, \gamma)$ *is a deterministic algorithm that takes a range token τ_Q for range Q and index γ as input. It outputs a sequence of identifiers $\boldsymbol{I}_Q$ and updated search index γ'.*

[1] This limitation is required to prove the security of our construction, but functionality can be provided for dynamic databases as well. However, the security properties cannot be proven formally due to the limitations raised by the range predicate encryption scheme.

7.2 Related Work

We refer to Chapter 3 for an overview of general related work on searchable symmetric encryption and encrypted databases. In this section we review related work particularly applicable to the functionality of range queries on encrypted data.

The idea of order-preserving encryption was introduced by Agrawal et al. [5]. In more detail, such kind of encryption has the following characteristic: given two plaintexts x and y with property $x \leq y$, the same property $\mathsf{OPE\text{-}Enc}(x) \leq \mathsf{OPE\text{-}Enc}(y)$ holds for their corresponding ciphertexts. A first concrete implementation of order-preserving encryption was introduced by Boldyreva et al. [21] with further improvements and additional security considerations published later [22]. An interactive scheme with client state fulfilling ideal security definitions stated by Boldyreva et al. has been published by Popa et al. [122] and further optimizations for the client state have been published by Kerschbaum and Schröpfer [91]. However, privacy properties of such deterministic property-preserving encryption schemes might be questionable for highly sensitive data. Recent work published by Naveed et al. [115] demonstrate concrete attacks on deterministic order-preserving encryption. Although Kerschbaum introduced the first randomized order-preserving encryption scheme [89] improvements of such attacks even applicable to randomized order-preserving encrypted values with limited entropy have been published subsequently by Grubbs [72].

The paradigm of searchable encryption for exact pattern matching, i.e. hide as much information as possible and only unveil tokens corresponding to requested predicates, can be transferred to encryption schemes supporting secure range queries as demonstrated by the following proposals. Solutions for the public key setting exist and have been published by Boneh [29] and improved by Shi et al. [134]. One solution in the secret key setting has been published by Shen et al. [133]. This construction has been revisited by Demertzis et al. [51], they realized a searchable encryption scheme for range queries using SSE for exact keyword matching internally as building block. Since their approach is solely based on symmetric cryptography, this approach results in faster execution time but leakage from queries increases, e.g. information about the covered subranges is unveiled. All these constructions have search costs that are linear in the database size since no privacy-preserving search index is created.

The first approach of building such encrypted search index for range queries has been introduced by Lu [106]. However, Lu's index reveals the order relation of all indexed elements initially after the outsourcing step, thus degrading its security properties to order-preserving encryption. A tradeoff between privacy and performance for range queries is proposed by Hore et al. [81] using bucketization of indexed ciphertexts. Other tree index approaches have been published by Wang et al. [143], however, again bucketization of indexed ciphertexts is leaked. Wang et al. published an encryption scheme that leaks the relative distance of all indexed ciphertexts to build an R-tree as index for ciphertexts [144]. The leakage of all these indexes is disclosed after the initial outsourcing step hence they are vulnerable to the before mentioned attacks as those published by Naveed et al.

Independently of our work, Roche et al. [128] published an approach with a similar high-level idea compared to our construction, that is, the search index is refined incrementally for each range query. They proposed the deployment of a general purpose semantically secure encryption scheme for the search index and all outsourced values. For each range query, the client downloads the search index and executes the range query locally. With the locally computed access pattern, the client can incrementally refine this search index, that is then updated on the server reducing the client's overhead for future queries. In Roche et al.'s solution the service provided by the untrusted server is general outsourced data storage; in contrast, we designed our construction with the aim to outsource as much computational effort as possible to the server side.

7.3 Implementation

From an abstract level we start with a range predicate encryption scheme, that is, a searchable encryption scheme providing search functionality for range queries but with linear search time since no encrypted index exists. We modify the range predicate encryption (RPE) range tokens in a way enabling the server to compare range tokens from different queries. By doing so, the server can use the access pattern together with the search pattern unveiled previously to accelerate future search queries. On the one hand, values that have fallen within a queried range, are stored in an interactively built index decreasing the processing time for future requests. On the other hand, values that have not been queried do not leak any information to the cloud service provider and stay semantically secure.

In the following Section 7.3.1 we elaborate on range predicate encryption and review the security properties thereof. Next, we describe the details of the search index on plain data before we give a comprehensive description how these indexing ideas can be realized on encrypted data using RPE.

7.3.1 Range Predicate Encryption

Founded on techniques published by Shen et al. [133] providing inner product encryption (IPE) and inspired by the construction given by Shi et al. [134], Lu has constructed an encryption scheme for range queries called RPE [106]. We refer to Appendix A.3 for a review of this construction. Specifically, an RPE scheme consists of the following algorithms.

Definition 21 (Predicate-only range-predicate encryption scheme from [106]). *A symmetric key range predicate encryption scheme* RPE *consists of the following four (probabilistic) polynomial-time algorithms.*

$k \leftarrow \mathsf{RPE\text{-}Setup}\left(1^\lambda, [0,d]\right)$ *is a probabilistic algorithm with security parameter 1^λ and a domain range $[0,d]$ as input. It outputs a private key k.*

$c_v \leftarrow \mathsf{RPE\text{-}Enc}\left(k, v\right)$ *is a probabilistic algorithm with key k and an value point v as input. It outputs a ciphertext c_v.*

$tk_Q \leftarrow \mathsf{RPE\text{-}Token}\left(k, Q\right)$ *is a probabilistic algorithm with key k and search range Q as input. It outputs range token tk_Q.*

$\{0,1\} \leftarrow \mathsf{RPE\text{-}Match}\left(tk_Q, c_v\right)$ *is a deterministic algorithm with range token tk_Q and ciphertext c_v as input. It outputs 1 if $v \in Q$ and 0 otherwise.*

Security properties for an RPE scheme guarantees plaintext privacy and predicate privacy. On the one hand, plaintext privacy guarantees that an attacker cannot distinguish between encrypted value points as formally stated in the following definition.

Definition 22 (RPE plaintext-privacy from [106]). *Let* RPE *be a range predicate encryption scheme as stated in Definition 21. Consider the following security experiment $\mathrm{Exp}_{\mathsf{RPE},\mathcal{A}}^{\mathrm{PlainPriv}}$ between adversary $\mathcal{A}$ and a challenger consisting of the phases described below:*

Init: *$\mathcal{A}$ submits two values $v_0, v_1 \in [0,d]$ where she wishes to be challenged.*

Setup: *The challenger generates a secret key k by running* $\mathsf{RPE\text{-}Setup}(1^\lambda, [0,d])$.

Query Phase 1: *$\mathcal{A}$ adaptively issues queries, where each query is one of two types:*

1. *Token query: On the i-th query, $\mathcal{A}$ submits range $Q_i \subset [0,d]$ with the following condition: either $(v_0 \notin Q_i \wedge v_1 \notin Q_i)$ or $(v_0 \in Q_i \wedge v_1 \in Q_i)$. The challenger generates a token by running $tk_{Q_i} \leftarrow \mathsf{RPE\text{-}Token}(k, Q_i)$ and outputs it.*

2. *Ciphertext query: On the i-th query, $\mathcal{A}$ submits a value z_i. The challenger encrypts this value point z_i by running $c_i \leftarrow \mathsf{RPE\text{-}Enc}(k, z_i)$ and returns the output.*

Challenge: *The challenger flips a random coin $b \overset{\$}{\leftarrow} \{0,1\}$ and outputs $\mathsf{RPE\text{-}Enc}(k, v_b)$.*

Query Phase 2: *$\mathcal{A}$ adaptively issues further queries with the same restrictions as stated in Query Phase 1.*

Guess: *$\mathcal{A}$ outputs a guess b'.*

The experiment outputs 1 if b equals b' and 0 otherwise.

We say RPE *has* selective secure plaintext privacy*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ running this security experiment, it holds that*

$$\left|\Pr\left[\mathrm{Exp}^{\mathrm{PlainPriv}}_{\mathsf{RPE},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

is negligible in λ.

On the other hand, predicate privacy guarantees that an attacker cannot distinguish between range tokens with the same access pattern as formally stated in the following definition.

Definition 23 (RPE predicate-privacy from [106])**.** *Let* RPE *be a range predicate encryption scheme as stated in Definition 21. Consider the following security experiment $\mathrm{Exp}^{\mathrm{PredPriv}}_{\mathsf{RPE},\mathcal{A}}$ between adversary $\mathcal{A}$ and a challenger consisting of the phases described below:*

Init: *$\mathcal{A}$ submits two ranges $R_0, R_1 \subset [0, d]$ where she wishes to be challenged.*

Setup: *The challenger generates a secret key k by running $\mathsf{RPE\text{-}Setup}(1^\lambda, [0, d])$.*

Query Phase 1: *$\mathcal{A}$ adaptively issues queries, where each query is one of two types:*

1. *Token query: On the i-th query, $\mathcal{A}$ submits range $Q_i \subset [0, d]$. The challenger generates a token by running $tk_{Q_i} \leftarrow \mathsf{RPE\text{-}Token}(k, Q_i)$ and outputs tk_{Q_i}.*
2. *Ciphertext query: On the i-th query, $\mathcal{A}$ submits a value z_i with the following condition: either $(z_i \in R_0 \wedge z_i \in R_1)$ or $(z_i \notin R_0 \wedge z_i \notin R_1)$. The challenger encrypts value point z_i by running $\mathsf{RPE\text{-}Enc}(k, z_i)$ and returns the output.*

Challenge: *The challenger flips a coin $b \overset{\$}{\leftarrow} \{0,1\}$ and outputs $\mathsf{RPE\text{-}Token}(k, R_b)$.*

Query Phase 2: *$\mathcal{A}$ adaptively issues further queries with the same restrictions as stated in Query Phase 1.*

Guess: *$\mathcal{A}$ outputs a guess b'.*

The experiment outputs 1 if b equals b' and 0 otherwise.

We say RPE *has* selective secure predicate privacy*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$ running this security experiment, it holds that*

$$\left|\Pr\left[\mathrm{Exp}^{\mathrm{PredPriv}}_{\mathsf{RPE},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}\right|$$

is negligible in λ.

Note that the adversary has to commit to her challenges before she can submit queries, that is, we assume selective privacy only. Further, the restrictions for the query phases prevent trivial distinguishability between the adversary's challenges.

It is straightforward that given such RPE scheme, one can construct a secure range query scheme SRQ as stated in Definition 20 with small leakage but linear runtime. Specifically, for message any $m_i = (v_i, ID(f_i))$ the attribute v_i is encrypted to $c_i = \text{RPE-Enc}(k, v_i)$ and the tuple $(c_i, ID(f_i))$ is indexed. For a range query of range Q a token tk_Q is created by the data owner holding the master key using RPE-Token. Given this token, the server retrieves $\mathbf{I}_Q$ simply as entries $ID(f_j)$ with $\text{RPE-Match}(tk_Q, c_j) = 1$. However, it is necessary to scan the complete index, hence runtime is linear in the number of all indexed files; in the remainder of this chapter we strive to decrease the amortized runtime to be logarithmic.

7.3.2 Protocol

Now we describe how we organize the search index in order to accelerate search executions while keeping the leakage of the indexed encrypted values minimal. We tackle these contradictory requirements by updating the index every time the server learns new information from the access pattern. This newly gained knowledge is then used to refine the encrypted search index for subsequent searches queried in the future. First, we explain our ideas and design decisions on plaintext data and transfer this approach on encrypted values in the upcoming section.

Organizing an Index for Plaintext Values

As already mentioned, we extend the idea from Chapter 5, that is, the searchable data is organized in a linear search index together with a result cache that can be accessed more efficiently. Particularly, the search index γ consists of the following two components:

The point list denoted as **P** is a linear list of all indexed points. This list enables the server to answer any range query in linear time.

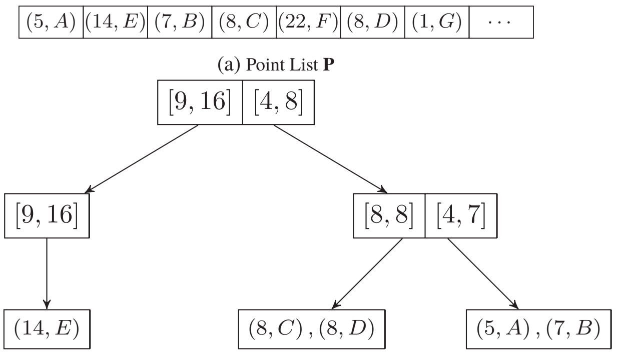
The tree list denoted as **T** is a list of search trees, each tree covering one coherent and already searched range. Whenever a new search is executed, existing trees are updated or a new tree is added to the list. This enables the server to answer range queries that are subranges of already queried ranges in logarithmic time.

Since we unveil range tokens and strive to reuse them, we decided for search trees keeping ranges as search keys for their nodes. More particular, **T** contains R-trees [73], where each R-tree Γ covers one coherent range completely. For simplicity we write $\Gamma \subseteq S$ for a range S, meaning that the range covered by Γ is a subset of S, vice versa $S \subseteq \Gamma$ and $S = \Gamma$. For each R-tree we define the threshold t, that is, any node holds up to t range entries; each entry has the form (p, R), where R is a range and p is a pointer to another node (either an inner node or a leaf) covering this range; hence pointer p points to a subtree. We denote $\Gamma[p]$ as the subtree of Γ pointed to by p. In addition, for any two entries (p_1, R_1) and (p_2, R_2) contained in the same node it holds that $R_1 \cap R_2 = \emptyset$, i.e. the ranges in one node are distinct and do not overlap. For every entry (p, R) it holds that the subtree rooted at the node pointed to by p covers the complete range R, i.e. $\Gamma[p] = R$. Finally, all leaves contain up to t entries, every entry has the form (obj, R), where R is a range and obj points to identifiers that are indexed under values falling within range R, denoted as ID_R. Queried range $Q = [q^{(s)}, q^{(e)}]$ and given an R-tree Γ covering a superset of Q (i.e. $Q \subset \Gamma$) the server can traverse this tree for Q in logarithmic time. More formally, the server is able to retrieve $\mathbf{I}_Q$ in logarithmic time using Algorithm 7.1.

An example for a search index γ containing **P** is depicted in Figure 7.1a and **T** with one tree Γ covering range $[4, 16]$ is depicted in Figure 7.1b. Assume the client submits a query for range $[17, 23]$. In the initial step, the server checks if there exists a tree $\Gamma \in \mathbf{T}$ that covers the queried range $[17, 23] \subset \Gamma$ to search in

Algorithmus 7.1 : `SearchForRange`

```
    Input : R-Tree $\Gamma$, Queried range $Q$
    Output : $\mathbf{I}_Q$
1:  Initialize temporary result list $L$
2:  for all $e_i = (p_i, R_i)$ in root of $\Gamma$ do
3:      if $R_i \subseteq Q$ then
4:          Add all values indexed by $\Gamma[p_i]$ to $L$
5:      end
6:      else if $q^{(s)} \in R_i$ or $q^{(e)} \in R_i$ then
7:          if $p_i$ points to another node then
8:              Add output of SearchForRange($\Gamma[p_i], Q$) to $L$
9:          end
10:         if $p_i$ points to a list $ID_{R_i}$ then
11:             Add all values with $v \in Q$ to $L$
12:         end
13:     end
14: end
15: return list $L$
```



(a) Point List **P**

(b) Initial R-Tree $\Gamma \in \mathbf{T}$

Figure 7.1: Example of plain search index γ consisting of **P** and **T**

logarithmic time. Since this is not the case, the server scans all entries in **P** linearly yielding the queried result set $ID_{[17,23]}$. Next, this fresh information is added to the search index for future queries and results an updated version of Γ covering $[4, 23]$ as depicted in Figure 7.2.

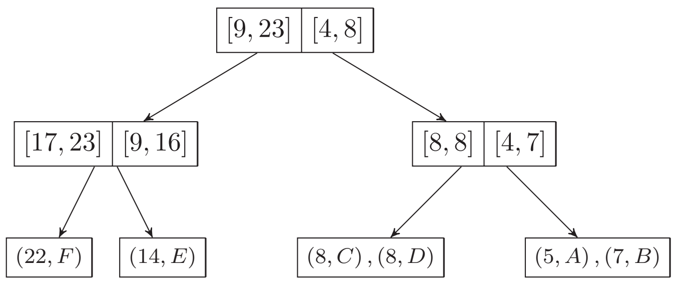


Figure 7.2: Updated R-Tree Γ after query $[17, 23]$

We refer to the following paragraph in for a comprehensive description of this updating process.

Building an Encrypted Range Index

Note that the following functionality is sufficient for answering such range queries by the execution of Algorithm 7.1:

1. check if range Q is a subrange of another range R required for identifying matching R-tree Γ in **T** and
2. check if range R and range Q intersect for tree traversal and
3. check if a value v falls withing range Q for a linear scan of values in matching leaves and the linear scan of point list **P**.

All required functionality can be provided by a slightly modified RPE scheme. A token for range $Q = [q^{(s)}, q^{(e)}]$ created by RPE-Token is supplemented by encrypted limiting points, that is start point and end point of the range. Particularly SRQ-Token outputs the following tuple $\big(\mathsf{RPE\text{-}Enc}(k, q^{(s)}), \mathsf{RPE\text{-}Enc}(k, q^{(e)}), \mathsf{RPE\text{-}Token}(k, Q)\big)$ as part of range token τ_Q.

Given two tokens $\tau_Q = \left(c_Q^{(0)}, c_Q^{(1)}, tk_Q\right)$ and $\tau_R = \left(c_R^{(0)}, c_R^{(1)}, tk_R\right)$ generated with this approach, the server is able to check for the following relation of these ranges, as required for tree traversal on encrypted R-trees.

Subrange: Q is a subrange of R if $\mathsf{RPE\text{-}Match}(tk_R, c_Q^{(0)}) = 1$ and $\mathsf{RPE\text{-}Match}(tk_R, c_Q^{(1)}) = 1$.

Intersection: R and Q intersect if $\mathsf{RPE\text{-}Match}(tk_Q, c_R^{(i)}) = 1$ or if $\mathsf{RPE\text{-}Match}(tk_R, c_Q^{(i)}) = 1$ for $i \in \{0, 1\}$.

Equality: Q and R are equal if R is a subrange of Q and Q is a subrange of R.

Such modified range predicate encryption scheme combined with an interactively and incrementally updated search index can be used to define an SRQ scheme with poly-logarithmic runtime and small leakage. Given a range predicate encryption scheme RPE consisting of algorithms $\big(\mathsf{RPE\text{-}Setup}, \mathsf{RPE\text{-}Enc}, \mathsf{RPE\text{-}Token}, \mathsf{RPE\text{-}Match}\big)$ and a secret-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ we construct an SRQ scheme as follows:

$K \leftarrow \mathsf{SRQ\text{-}Setup}\left(1^\lambda, [0, d]\right)$: on input of security parameter 1^λ and value domain $[0, d]$ create the following keys corresponding to the security parameter

$$k_1 \leftarrow \mathsf{RPE\text{-}Setup}\left(1^\lambda, [0, d]\right)$$
$$k_2 \leftarrow \mathsf{Gen}\left(1^\lambda\right)$$

Output master key $K = (k_1, k_2)$.

$c \leftarrow \mathsf{SRQ\text{-}Enc}(K, \mathbf{M})$: on input of master key $K = (k_1, k_2)$ and message collection $\mathbf{M}$ containing n messages do the following:

1. Initialize an empty search index $\gamma = (\mathbf{P}, \mathbf{T})$ that contains an empty point list **P** and an empty tree list **T**.
2. For each message $m_i = (v_i, ID(f_i))$ encrypt $c_i \leftarrow \mathsf{RPE\text{-}Enc}(k_1, v_i)$ and append $(c_i, ID(f_i))$ to point list **P**.

Output the encrypted search index γ.

$\tau_Q \leftarrow \mathsf{SRQ\text{-}Token}(K, Q)$**:** on input of master key $K = (k_1, k_2)$ and range $Q = [q^{(s)}, q^{(e)}]$ flip a coin $b \xleftarrow{\$} \{0, 1\}$ and use RPE-Enc for encrypting the limiting points:

$$c_Q^{(b)} \leftarrow \mathsf{RPE\text{-}Enc}(k_1, q^{(s)})$$
$$c_Q^{(1-b)} \leftarrow \mathsf{RPE\text{-}Enc}(k_1, q^{(e)}).$$

Furthermore create range token $tk_Q \leftarrow \mathsf{RPE\text{-}Token}(k_1, Q)$ and encrypt Q to $c_Q \leftarrow \mathsf{Enc}(k_2, Q)$. Output $\tau_Q = \left(c_Q^{(0)}, c_Q^{(1)}, tk_Q, c_Q\right)$ as range token.

$\mathbf{I}_Q, \gamma' \leftarrow \mathsf{SRQ\text{-}Search}(\tau_Q, \gamma)$**:** on input of range token $\tau_Q = \left(c_Q^{(0)}, c_Q^{(1)}, tk_Q, c_Q\right)$ for range Q and encrypted search index γ initialize an empty list $\widehat{\mathbf{T}}$ for trees that are completely covered by the queried range. For each indexed search tree $\Gamma_i \in \mathbf{T}$:

1. Check if there exists one tree $\Gamma_i \in \mathbf{T}$ containing *both* $c_Q^{(0)}$ *and* $c_Q^{(1)}$. If this is the case, retrieve $\mathbf{I}_Q$ by calling Algorithm 7.1 and set $\Gamma^{(0)} = \Gamma^{(1)} = \Gamma_i$.
2. Otherwise, set tree $\Gamma^{(0)}$ with $c_Q^{(0)} \in \Gamma^{(0)}$ and $\Gamma^{(0)} = \perp$ if no such tree exists. Analogously, do the same with $c_Q^{(1)}$ and denote $\Gamma^{(1)}$ as the corresponding result. For each tree that is covered completely by Q, append it to list $\widehat{\mathbf{T}}$ resulting in $\widehat{\mathbf{T}} = \{\Gamma_i \mid \Gamma_i \subseteq Q\}$. Scan all ciphertexts $(c_i, ID(f_i)) \in \mathbf{P}$ using $\mathsf{RPE\text{-}Match}(tk_Q, c_i)$ and store $ID(f_i)$ if their corresponding value falls withing the queried range.

In order to maintain logarithmic search time for future queries, call an interactive procedure $\gamma' \leftarrow \mathsf{SRQ\text{-}UpdateIndex}(\tau_Q, \mathbf{I}_Q, \Gamma^{(0)}, \Gamma^{(1)}, \widehat{\mathbf{T}})$ described comprehensively in the following section. Finally, output $\mathbf{I}_Q$ as result and the updated search index γ'.

Recall that the server is able to compare different range tokens without knowing the master key or underlying plaintext values. The server getting a range token τ_Q for range $Q = [q^{(s)}, q^{(e)}]$ executes SRQ-Search and retrieves all files associated with values falling within range Q. In the initial Step 1, the server checks if she has extracted enough information from previous queries to answer the current query in logarithmic time. Alternatively, in Step 2 the server checks if any information about range Q is already indexed in at least one search tree. All tokens contained in the entries $(p_1, \tau_{R_1}), \ldots, (p_{n_i}, \tau_{R_{n_i}})$ contained in the root node of Γ_i are compared with range token τ_Q. Here $\Gamma_i \subset Q$ if all $R_1, \ldots, R_{n_i}$ are subranges of Q. A list $\widehat{\mathbf{T}} = \{\Gamma_i \mid \Gamma_i \subset Q\}$ of all trees covering a subrange of Q is created. Further, partial intersections of indexed search trees and the queried range are checked, defining $\Gamma^{(0)}$ as the tree intersecting with $c_Q^{(0)}$ and analogously $\Gamma^{(1)}$ as the tree intersecting with $c_Q^{(1)}$. Finally, the server decides how to update the search index based on the result of these checks, that is, depending on the nature of $\widehat{\mathbf{T}}$ and $\Gamma^{(0)}, \Gamma^{(1)}$ different update strategies for SRQ-UpdateIndex must be applied.

1. One tree covers the complete queried range Q, that is $\Gamma^{(0)} = \Gamma^{(1)}$, so $Q \subset \Gamma$. If this is the case, the server does not need to perform a search over the complete point list **P** but searching over the value points indexed by $\Gamma^{(0)}$ is sufficient. This is done using Algorithm 7.1.

   Finally, SRQ-UpdateIndex has to *refine* indexed ranges by using information gained from the current range query.

2. No intersection of the current range query and all previously queried ranges, so $\Gamma^{(0)} = \Gamma^{(1)} = \perp$ and $\widehat{\mathbf{T}} = \emptyset$. If this is the case, the server does not know anything about the current range query. As a result, the server has to scan all points indexed in point list **P**.

   Finally, SRQ-UpdateIndex has to *create* a new search tree that is added to tree list **T** covering the queried range.

3. Only one part of the queried range is covered by indexed search trees. Either $\Gamma^{(0)} = \perp$ or $\Gamma^{(1)} = \perp$. If this is the case, the server cannot know if there are values in point list **P** falling within Q but are not covered by $\Gamma^{(0)}$ resp. $\Gamma^{(1)}$. As a result, the server has to scan all points indexed in point list **P**.

   Finally, SRQ-UpdateIndex has to *extend* the one tree covering the queried range partly, i.e. either $\Gamma^{(0)}$ or $\Gamma^{(1)}$ that is not $\perp$.

4. The values fall within different trees, that is $c^{(0)} \in \Gamma^{(0)}$ and $c^{(1)} \in \Gamma^{(1)}$ where $\Gamma^{(0)} \neq \Gamma^{(1)}$. If this is the case, the server cannot be sure that there is no "index gap" between the two trees, i.e. there could be values in **P** falling neither within $\Gamma^{(0)}$ nor $\Gamma^{(1)}$ but that fall within queried range Q. As a result, the server has to scan all points indexed in point list **P**.

   Finally, SRQ-UpdateIndex has to *merge* these trees $\Gamma^{(0)}$ and $\Gamma^{(1)}$ closing the gap revealed by the current range query.

In the following subsection we elaborate on these different updating cases and describe the updating procedure in a formal protocol.

Updating the Encrypted Index

From a high-level perspective, a new range token contains novel information given to the server, namely the result set ID_Q and the relation of Q to all previous queries. This newly gained information is implicitly leaked by the access pattern, that is, given $ID_Q \subset ID_U$ the server can deduce that $Q \subset U$. Remember that all efficient searchable encryption schemes with few exceptions leak this information. Our construction utilizes this leakage to update the encrypted search index for accelerating future queries. Update operations require the creation of fresh range tokens used to extend the encrypted search trees. Since this creation is only possible given the master key, these update procedures are interactive protocols between server and data owner; in the following discussion we highlight operations on the client side with the prefix"@Client: Some Operation". All these tree modifications append new entries to an existing tree which might result in a number of entries in one node that is larger than t. Thus, a rebalancing step is required such that every node's size is lower than threshold t afterwards. Protocol 7.2 formalizes this interactive rebalancing step for encrypted R-trees. Further details on how SRQ-UpdateIndex works internally are discussed in the following paragraphs.

Refining a tree. An existing encrypted search tree Γ can be refined given range token τ_Q if this search tree and the range token intersect, i.e. $Q \cap \Gamma \neq \emptyset$. The server sends range token τ_Q and the encrypted R-tree Γ intersecting with that range token to the data owner asking for help. Recall that the client has access to the master key, hence she is able to decrypt the range tokens together with the tree's intersecting leaves. Using these intersecting leaves, she creates (up to) four non intersecting but more refined ranges and sends back their tokens generated by SRQ-Token. The server replaces the old range tokens with the novel, more refined tokens and the indexed file lists are segmented according to these novel tokens. A formal description of this protocol is given in Protocol 7.3. Since this replacement increases the number of entries in one node, the server finally runs Protocol 7.2.

Creating a tree. A fresh encrypted R-tree is created if no information on the novel queried range is cached in any search tree indexed in **T**. In such case, the server creates a new tree Γ with one node only containing the novel range token τ_Q and indexed items ID_Q retrieved from the linear point list **P**. This tree Γ is added to tree list **T**.

Protocol 7.2 : `RebalanceTree`

```
Input : Tree $\Gamma$ with leaf $l$ such that len$(l) > t$.
Output : Rebalanced tree $\Gamma$.
1: Set cur_node $= l$
2: while len(cur_node) $> t$ do
3:     Send all entries $e_i = (p_i, \tau_{R_i}) \in$ cur_node to client
                                  /* Split this node in two nodes of the same size */
4:     @Client: Sort all entries $\{e_i = (p_i, \tau_{R_i})\}$ according to their range $R_i$
5:     @Client: Create two tokens $\tau_U, \tau_V$ using SRQ-Token for ranges $U = \bigcup_{i=0}^{\lceil \frac{n}{2} \rceil - 1} R_i$,
       $V = \bigcup_{i=\lceil \frac{n}{2} \rceil}^{n} R_i$ and set nodes $N_U = \{e_i \mid R_i \subseteq U\}$, $N_V = \{e_i \mid R_i \subseteq V\}$
6:     @Client: Send permutations of $N_V, N_U$ with corresponding tokens $\tau_V, \tau_U$ to server
7:     if cur_node is not root then
8:         Replace cur_node $\in \Gamma$ with $N_V, N_U$ indexed with tokens $\tau_V, \tau_U$ in cur_node.parent
9:         Set cur_node to cur_node.parent
10:    end
11:    else
12:        Replace cur_node with $L_V, L_U$
13:        Create new root only containing tokens $\tau_V, \tau_U$ pointing to $L_V, L_U$
14:    end
15: end
```

Protocol 7.3 : Refining an encrypted R-tree.

```
Input : Tree $\Gamma$; range token $\tau_Q$.
Output : Refined Tree $\Gamma$.
1: for $q \in \{q^{(s)}, q^{(e)}\}$ do
2:     Search leaf containing token $\tau_R$ with $q \in R$ in $\Gamma$
3:     Send $\tau_R$ and $\tau_Q$ to the client
4:     @Client: For $R = [r^{(s)}, r^{(e)}]$ and $q$ set $Q_1 = [r^{(s)}, q]$ and $Q_2 = [q+1, r^{(e)}]$
5:     @Client: Send back $\tau_{Q_1}, \tau_{Q_2}$ generated with SRQ-Token
6:     Divide $ID_R$ that is pointed to by leaf into two lists $ID_{Q_1}, ID_{Q_2}$
7:     In leaf replace $(ID_R, \tau_R)$ with two new entries $(ID_{Q_1}, \tau_{Q_1}), (ID_{Q_2}, \tau_{Q_2})$
8:     RebalanceTree($\Gamma$, leaf)
9: end
```

Extending a tree. An encrypted tree Γ can be extended if a queried range token τ_Q intersects with one tree Γ or covers this tree completely, i.e. $q^{(s)} \in \Gamma$ and $q^{(e)} \notin \Gamma$ (or vice versa) or even $\Gamma \subset Q$. The server sends the newly learned range token τ_Q and the root node to the client. The client decrypts all ranges contained in the root to reconstruct the whole range currently covered by this tree. A new range token for the gap between the range covered by the current tree and the limiting points of the new range token extending the current tree is created and added to the updated tree. We refer to Protocol 7.4 for a formal description. The resulting tree must be rebalanced after tree extension since at least one leaf got a new entry.

Merging two trees. Two trees Γ_1, Γ_2 can be merged if they both intersect with the queried range token τ_Q, i.e. $q^{(s)} \in \Gamma_1$ and $q^{(e)} \in \Gamma_2$. Note that these two trees must not have a value gap between them. In more details, the range $R_1 = [r_1^{(s)}, r_1^{(e)}]$ covered by one tree must be directly followed by second range $R_2 = [r_1^{(s)}, r_1^{(e)}]$ covered by the other tree, that is, $r_1^{(e)} + 1 = r_2^{(s)}$ This can be achieved by the execution of tree extension as described in Algorithm 7.4. The algorithm integrates the tree with the lower height into the tree with greater height resulting in logarithmic merge time. That is, a new entry inner node is created in the higher tree pointing to the root of the lower tree. This newly covered range must then be propagated

Protocol 7.4 : Extending encrypted R-tree.

```
Input : Tree $\Gamma$; range token $\tau_Q$.
Output : Updated tree $\Gamma$ covering $\tau_Q$ completely.
1: Set root_node to the root node of $\Gamma$
2: Send root_node and range token $\tau_Q$ to client
3: @Client: Set $R = [r^{(s)}, r^{(e)}] = \bigcup_i R_i$ for all entries $(p_i, R_i)$ in root_node
                    /* Compute the complete range covered by current tree.  */
4: for $r \in \{r^{(s)}, r^{(e)}\}$ do
5:     @Client: Ask server for all nodes in $\Gamma$ containing $r$, retrieve node set $N = \{n_j \mid r \in n_j\}$
6:     for cur_node $\in N$ do
7:         @Client: Extract lowest resp. greatest range $R' = [r'^{(s)}, r'^{(e)}]$ in cur_node
8:         if cur_node is not a leaf then
9:             @Client: Extend this range to $Q' = [q^{(s)}, r'^{(e)}]$ resp. $Q' = [r'^{(s)}, q^{(e)}]$
10:            @Client: Replace inner entry $\tau_{R'}$ with $\tau_{Q'}$ generated by SRQ-Token
11:        end
12:        else
13:            @Client: Set range $Q' = [q^{(s)}, r'^{(s)} - 1]$ resp. $Q' = [r'^{(e)} + 1, q^{(e)}]$
14:            @Client: Create new token $\tau_{Q'}$ using SRQ-Token
15:            Add new entry $(\tau_{Q'}, ID_{Q'i})$ to leaf $n_j$
16:        end
17:    end
18:    RebalanceTree($\Gamma$, leaf)
19: end
```

through the inner nodes up to the root. See Algorithm 7.5 for a formal description. Again, rebalancing the resulting tree is required.

Merging multiple trees We can repeatedly combine the previously explained steps of tree extension and tree merging given a range token τ_Q that covers multiple indexed trees. In more details, the root nodes of all trees $\Gamma_i \in \widehat{\mathbf{T}}$ together with $\Gamma^{(0)}, \Gamma^{(1)}$ and the newly queried range token τ_Q are sent to the client. The client decrypts all root nodes retrieving ranges $R_i = [r_i^{(s)}, r_i^{(e)}]$ covered by tree Γ_i and sorts these ranges according to their start value $r_i^{(s)}$. Next, the client chooses two trees Γ_j, Γ_{j+1} she wants to merge. Without loss of generality lets assume Γ_j has greater height, so we extend Γ_j to cover $[r_j^{(s)}, r_{j+1}^{(s)} - 1]$ using Algorithm 7.4. Now Γ_j and Γ_{j+1} can be merged using Algorithm 7.5 and the number of different trees is reduced by one. This is done repeatedly until exactly one search tree is left covering the complete queried range.

7.4 Evaluation

In this section we evaluate the security properties of our construction. Particularly, in Section 7.4.1 we state the upper bound of the leakage induced by our construction and prove it in the formal framework. Further, we analyze performance numbers from a theoretical perspective in Section 7.4.2 and state practical runtime numbers in Section 7.4.3. In our practical evaluation we utilize the modular construction of RPE and analyze two different implementations for range predicate encryption. The first implementation is based on the inner product encryption proposed by Shen et al. [133] making this benchmark comparable to the work published by Lu [106]. In addition, we implemented a more recent scheme published by Bishop et al. [18] with better runtime numbers but larger encryption keys.

Protocol 7.5 : Merging two encrypted R-trees.

```
    Input : Two trees $\Gamma_1, \Gamma_2$.
    Output : One merged tree $\Gamma$.
1:  Set $\Gamma \in \{\Gamma_1, \Gamma_2\}$ to higher and $\Gamma' \in \{\Gamma_1, \Gamma_2\}$ to lower tree with height $h$ and $h'$
2:  Send roots of both trees $\Gamma, \Gamma'$ covering $R = [r^{(s)}, r^{(e)}]$ resp. $R' = [r'^{(s)}, r'^{(e)}]$ to client
3:  @Client: if $r^{(s)} > r'^{(s)}$ then
4:      Set $v = r^{(s)}$
5:  end
6:  @Client: else
7:      Set $v = r^{(e)}$
8:  end
9:  @Client: Create $\tau_{R'}$ using SRQ-Token and auxiliary value $c_v \leftarrow$ RPE-Enc$(k, v)$ used for tree
    traversal
10: Set $i = h$ and cur_node to root of $\Gamma$
11: while $i > h'$ do
12:     Send $\tau_{R'}$ in entry $e_i = (p_i, \tau_{R_i})$ in cur_node with $v \in R_i$ and $\tau_{R'}$ to client
13:     @Client: Compute $U = R_i \cup R'$ and send back token $\tau_U$ generated using SRQ-Token
14:     In entry $e_i$ replace $\tau_{R_i}$ with $\tau_U$
15:     Set cur_node to node pointed to by $p_i$
16:     $i = i - 1$
17: end
18: Insert entry $(p', \tau_{R'})$ in cur_node, where $p'$ points to tree $\Gamma'$
19: RebalanceTree($\Gamma$, cur_node)
```

7.4.1 Formal Security Proof

Founded on the simulation-based security proof as discussed in Section 4.1.1 we quantify the leakage of our implementation for secure range queries as formally stated in Definition 20 with poly-logarithmic runtime. Recall that – in order to achieve this logarithmic execution time – messages have to be indexed in a suitable way while the index should leak as little information as possible. The incremental generation of such index is based on the initially generated ciphertexts and subsequently queried range tokens. In the following we adapt the general simulation-based security framework fitting our specific construction and state the leakage induced by SRQ-Enc and SRQ-Token. We prove security against a non-adaptive adversary as defined in the following.

Definition 24. *Given a scheme for secure range queries* SRQ = (SRQ-Setup, SRQ-Enc, SRQ-Token, SRQ-Search) *and security parameter $\lambda \in \mathbb{N}$, we consider the following probabilistic experiments with a stateful adversary $\mathcal{A}$, a stateful simulator $\mathcal{S}$ and leakage function $\mathcal{L}$:*

$\mathbf{Real}^{\mathsf{SRQ}}(\lambda)$: *the challenger runs* SRQ-Setup$(1^\lambda, [0,d])$ *to generate a master key K and an empty search index γ. First the adversary sends a q-tuple of messages $\boldsymbol{M} = (m_1, \ldots, m_q)$ where each message is a tuple $m_i = (v_i, ID(f_i))$ with $v_i \in [0,d]$ for all $i \in \{1, \ldots, q\}$ and a q'-tuple of range queries $\mathbf{Q} = (Q_1, \ldots, Q_{q'})$ with $Q_j \subset [0,d]$ for all $j \in \{1, \ldots, q'\}$ to the challenger. The challenger returns a sequence of ciphertexts $\boldsymbol{C} = (\mathsf{SRQ\text{-}Enc}(K, m_1), \ldots, \mathsf{SRQ\text{-}Enc}(K, m_q))$ together with a sequence of range tokens $\boldsymbol{T} = (\mathsf{SRQ\text{-}Token}(K, Q_1), \ldots, \mathsf{SRQ\text{-}Token}(K, Q_{q'}))$ to the adversary. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.*

$\mathbf{Ideal}^{\mathsf{SRQ}}(\lambda)$: *the simulator sets up its internal environment for domain $[0,d]$. The adversary $\mathcal{A}$ sends a q-tuple of messages $\boldsymbol{M} = (m_1, \ldots, m_q)$ where each message is a tuple $m_i = (v_i, ID(f_i))$ with $v_i \in [0,d]$ for all $i \in \{1, \ldots, q\}$ and a q'-tuple $\mathbf{Q} = (Q_1, \ldots, Q_{q'})$ with $Q_j \subset [0,d]$ for all $j \in \{1, \ldots, q'\}$ to the challenger. The simulator $\mathcal{S}$ is given the appropriate leakage $\mathcal{L}(\boldsymbol{M}, \mathbf{Q})$ and* *returns a q-tuple $\widetilde{\mathbf{C}}$ of simulated ciphertexts and a q'-tuple $\widetilde{\mathbf{T}}$ of simulated range tokens to the adversary. Finally, $\mathcal{A}$ returns a bit b that is output by the experiment.*

We say SRQ *is $\mathcal{L}$-secure against non-adaptive chosen-range attacks if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a probabilistic polynomial-time simulator $\mathcal{S}$, so that advantage of $\mathcal{A}$ defined as*

$$\left|\Pr\left[\mathbf{Real}^{\mathsf{SRQ}}(1^\lambda) = 1\right] - \Pr\left[\mathbf{Ideal}^{\mathsf{SRQ}}(1^\lambda) = 1\right]\right|$$

is negligible in λ.

Note that the stated leakage is based on black-box implementation of range predicate encryption. That is, the stated leakage is an upper bound assuming the internally used range predicate encryption scheme provides plaintext privacy as stated in Definition 22 and predicate privacy as stated in Definition 23. We state the following leakage function as an upper bound

$$\mathcal{L}(\mathbf{M}, \mathbf{Q}) = \left(\left(ID(f_1), \ldots, ID(f_q)\right), ID_{\mathbf{Q}}, \mathtt{RR}(\mathbf{Q})\right)$$

where $ID(f_i)$ is the identifier of the message that might be used as link to a potential payload as already discussed in Section 5.1.1. Further, the result set for each query in the query sequence $ID_{\mathbf{Q}} = \left(ID_{Q_1}, \ldots, ID_{Q'_q}\right)$ is unveiled. Finally, $\mathtt{RR}(\mathbf{Q})$ is a $q' \times q'$ *range relation matrix*, each element in this matrix is out of the set $\{\emptyset, \cap, \subset, =, \subset^{=}, \supset, \supset^{=}\}$. Here an element in row i and column j indicates the relation of ranges Q_i and Q_j given in queries i and j. Particularly, $\emptyset$ denotes no intersection, $\cap$ denotes a intersection but no range is a subrange of the other. $\subset$ denotes that range Q_i is a subset of Q_j but no limiting points are in common, $\subset^{=}$ denotes a subset relation with one limiting point in common, and the other way round $\supset$ denotes that range Q_i is a superset of Q_j, i.e. if $\subset$ is at position (i, j) than $\supset$ is at position (j, i). These range relations can be formalized by multiple inequations, as given in Table 7.3.

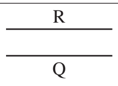
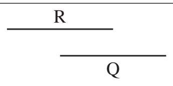
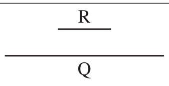
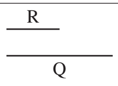
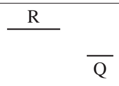
$=$	$\cap$	$\subset$	$\subset^{=}$	$\emptyset$
				
$r^{(s)} = q^{(s)} \wedge r^{(e)} = q^{(e)}$	$r^{(s)} < q^{(s)} \wedge$ $r^{(e)} \geq q^{(s)} \wedge$ $r^{(e)} < q^{(e)}$	$r^{(s)} > q^{(s)} \wedge r^{(e)} < q^{(e)}$	$r^{(s)} = q^{(s)} \wedge \quad r^{(e)} < q^{(e)}$ or $r^{(s)} > q^{(s)} \wedge \quad r^{(e)} = q^{(e)}$	$r^{(e)} < q^{(s)}$

Table 7.3: Illustration and formal representation of different relationships between ranges.

Note that this information can be extracted from the access pattern, namely if ID_Q intersects with ID_R then Q intersects with R as well. We emphasize that only encrypted values that fall within a queried range leak information, while encrypted values that have not been queried remain semantically according to Definition 22. Shuffling the encrypted limiting values as implemented in SRQ-Token for each range token results in an obfuscation of the order relation of overlapping queried ranges but preservers the fact that they overlap. Particularly, this construction does not leak the order relation of matching values but only a bucketization of these values.

Recall Definition 22 of selective secure plaintext privacy where the challenger only accepts challenges v_0, v_1 that both occur in the same subset of queried access patterns. In more detail, if message m_i indexed under v_i is contained in ID_{Q_j} it must hold that m_{1-i} indexed under v_{1-i} also is part of the result set ID_{Q_j} for $i \in \{0, 1\}$ and all token queries Q_j. By unveiling the range relation matrix to the simulator the simulator can construct varying range sequences resulting in indistinguishable range token sequences. That is, these sequences of tokens generated by SRQ-Token cannot be distinguished by any PPT adversary, assuming SRQ-Token is founded on an RPE encryption scheme with plaintext privacy and predicate privacy. We observe that given two range token sequences with the same range relation matrix (for their ranges), no attacker can distinguish these range token sequences generated by $\mathsf{SRQ\text{-}Token}$. A more formal statement of this property is stated in the following Lemma 1.

Lemma 1. *Assume* $\mathsf{SRQ\text{-}Token}$ *is built upon an IND-CPA secure secret-key encryption scheme* Π *and a range predicate encryption scheme* RPE *with selective secure plaintext privacy according to Definition 22 and selective secure predicate privacy according to Definition 23. Given a domain* $[0, d]$, *two query sequences* $(Q_1, \ldots, Q_n) = \mathbf{Q} \neq \mathbf{R} = (R_1, \ldots, R_n)$ *with* $Q_i \subset [0, d], R_i \subset [0, d]$ *and* $RR(\mathbf{Q}) = RR(\mathbf{R})$ *then the tuples* $\boldsymbol{T}_{\mathbf{Q}} = (\mathsf{SRQ\text{-}Token}(K, Q_1), \ldots, \mathsf{SRQ\text{-}Token}(K, Q_n))$ *and* $\boldsymbol{T}_{\mathbf{R}} = (\mathsf{SRQ\text{-}Token}(K, R_1), \ldots, \mathsf{SRQ\text{-}Token}(K, R_n))$ *are indistinguishable for any adversary* $\mathcal{A}$ *and any master key* $K \leftarrow \mathsf{SRQ\text{-}Setup}(1^\lambda, [0, d])$.

Proof. For the sake of brevity of this proof we omit the explicit notation of the secret key, e.g. we abbreviate $\mathsf{RPE\text{-}Token}(k_1, Q)$ by $\mathsf{RPE\text{-}Token}(Q)$. We prove that a simulator can extend, shrink and move the ranges, so that the probability of any adversary $\mathcal{A}$ to distinguish between a token $\tau_Q \leftarrow \mathsf{SRQ\text{-}Token}(Q)$ and token $\tau_{\widetilde{Q}} \leftarrow \mathsf{SRQ\text{-}Token}(\widetilde{Q})$ that is an extended, shrunk or moved version of Q is negligible. These modifications are then exploited to transform a complete range sequence $\mathbf{Q}$ to $\mathbf{R}$ such that any adversary can distinguish these sequences only with negligible probability.

Recall that algorithm $\mathsf{SRQ\text{-}Token}$ on input of a range Q outputs a range token τ_Q in form of the tuple $\left(c_Q^{(0)}, c_Q^{(1)}, tk_Q, c_Q\right)$ with $c_Q^{(0)} \leftarrow \mathsf{RPE\text{-}Enc}(q^{(s)})$, $c_Q^{(1)} \leftarrow \mathsf{RPE\text{-}Enc}(q^{(e)})$ (or vice versa), $tk_Q \leftarrow \mathsf{RPE\text{-}Token}(Q)$ and $c_Q \leftarrow \mathsf{Enc}(Q)$. Denote ε_Π as the negligible probability of an adversary $\mathcal{A}$ winning the IND-CPA security game for the secret-key encryption scheme Π. Denote ε_1 as the negligible probability of an adversary $\mathcal{A}$ winning the RPE plaintext privacy security game and ε_2 as the negligible probability of an adversary $\mathcal{A}$ winning the RPE predicate privacy game.

First we prove that, given a range token $\tau_Q = (c_Q^{(0)}, c_Q^{(1)}, tk_Q, c_Q)$, it is possible to extend range Q to range $\widetilde{Q}$. We present a series of games and show that the probability of any adversary $\mathcal{A}$ to distinguish two games is negligible.

In $\mathbb{G}_0$ the original token τ_Q is output.

In $\mathbb{G}_1$ replace c_Q with encryption $c_{\widetilde{Q}} = \mathsf{Enc}(\widetilde{Q})$. $\mathcal{A}$ can distinguish between $\mathbb{G}_0$ and $\mathbb{G}_1$ with probability ε_Π.

In $\mathbb{G}_2$ we replace tk_Q with this new RPE token $tk_{\widetilde{Q}} = \mathsf{RPE\text{-}Token}(\widetilde{Q})$. Note that $q^{(s)} \in \widetilde{Q}$ and $q^{(e)} \in \widetilde{Q}$ still holds. Hence, adversary $\mathcal{A}$ can distinguish between $\mathbb{G}_1$ and $\mathbb{G}_2$ with probability ε_2.

In $\mathbb{G}_3$ we move the limiting point $c_Q^{(i)}$ that encrypts $q^{(e)}$: Replace $c_Q^{(i)} = \mathsf{RPE\text{-}Enc}(q^{(e)})$ with $\widetilde{c_Q^{(i)}} = \mathsf{RPE\text{-}Enc}(\widetilde{q^{(e)}})$. $\mathcal{A}$ can distinguish between $\mathbb{G}_2$ and $\mathbb{G}_3$ with probability ε_1.

After $\mathbb{G}_3$ we have a valid token $\tau_{\widetilde{Q}}$ for the new range $\widetilde{Q}$. Putting it altogether, adversary $\mathcal{A}$ can distinguish between these tokens with probability $\widetilde{\varepsilon} = \varepsilon_\Pi + \varepsilon_2 + \varepsilon_1$ which is still negligible.

The analogue argument holds for shrinking a range Q to a range $\widetilde{Q} = [q^{(s)} + x, q^{(e)}]$, with swapped order of $\mathbb{G}_2$ and $\mathbb{G}_3$. Again, as a result any adversary $\mathcal{A}$ can distinguish between a token τ_Q and a token for a shrunk range $\tau_{\widetilde{Q}}$ with probability $\widetilde{\varepsilon} = \varepsilon_\Pi + \varepsilon_1 + \varepsilon_2$.

Combining these two techniques we can move any range $Q = [q^{(s)}, q^{(e)}]$ to a new range $\widetilde{Q} = [q^{(s)} + x, q^{(e)} + x]$ such that any attacker can only distinguish between token $\tau_Q \leftarrow \mathsf{SRQ\text{-}Token}(Q)$ and $\widetilde{Q} \leftarrow \mathsf{SRQ\text{-}Token}(\widetilde{Q})$ with negligible probability $2\widetilde{\varepsilon}$.

With this insight, we can complete the proof for Lemma 1. That is, we can move each range $Q_i \in \mathbf{Q}$ to a corresponding range $R_i \in \mathbf{r}$ while any adversary can distinguish between both with version only with probability $2\widetilde{\varepsilon}$; here it is crucial to do this without altering the relation matrix for any intermediate modified range. As there are at most n ranges to be modified, the overall probability to distinguish between tokens generated for $\mathbf{Q}$ and tokens generated for $\mathbf{R}$ is $\varepsilon \leq 2n\widetilde{\varepsilon} = 2n\varepsilon_\Pi\varepsilon_1\varepsilon_2$ and is still negligible for negligible $\varepsilon_\Pi, \varepsilon_1, \varepsilon_2$. □

Given Lemma 1 we can now formally prove the security for the complete search protocol SRQ as stated in Definition 24 with leakage $\mathcal{L}$. The major idea of the proof is that simulator $\mathcal{S}$ generates simulated data and query sequences based on the leaked range relation matrix. According to Lemma 1 and plaintext privacy provided by RPE, any adversary has only negligible probability to distinguish the real protocol and such simulated protocol execution. More formally, we state the following Theorem 4 for the implemented secure range query protocol SRQ and prove it giving a concrete description of a simulator $\mathcal{S}$.

Theorem 4. *Let* RPE *be a range predicate encryption scheme with* selective secure plaintext privacy *according to Definition 22 and* selective secure predicate privacy *according to Definition 23. Further let* Π *be an IND-CPA secure symmetric encryption scheme. Then our scheme for secure range queries* SRQ *as described in the previous Section 7.3.2 is* $\mathcal{L}$*-secure against non-adaptive chosen-range attacks according to Definition 24.*

Proof. We describe a polynomial-time simulator $\mathcal{S}$ for which the advantage of any PPT adversary $\mathcal{A}$ to distinguish between the $\mathbf{Real}^{\mathsf{SRQ}}$ experiment and $\mathbf{Ideal}^{\mathsf{SRQ}}$ experiment from Definition 24 is negligible. Particularly, $\mathcal{S}$ sets up the environment and simulates range tokens $\widetilde{\mathbf{T}}$ and ciphertexts $\widetilde{\mathbf{C}}$ using leakage $\mathcal{L}$ as follows:

Setting up the environment: $\mathcal{S}$ internally executes the setup algorithm SRQ-Setup$(1^\lambda, [0, d])$ and generates a master key K.

Simulating range token sequence $\widetilde{\mathbf{T}}$: Given leakage

$$\mathcal{L}(\mathbf{M}, \mathbf{Q}) = \left((ID(f_1), \ldots, ID(f_q)), \mathtt{RR}(\mathbf{Q})\right)$$

$\mathcal{S}$ extracts clusters of ranges that form one coherent range from $\mathtt{RR}(\mathbf{Q})$. That is, for each range Q_i all directly intersecting ranges Q_j or transitively connected ranges Q_u are grouped into one cluster, e.g. using Algorithm 7.6. Each cluster corresponds to one R-Tree in the implementation presented in Section 7.3.2. For each cluster $\mathcal{S}$ simulates ranges with the same range relation matrix as given by $\mathtt{RR}(\mathbf{Q})$. In more detail, for each cluster simulator $\mathcal{S}$ transforms the range relation matrix $\mathtt{RR}(\mathbf{Q})$ into a linear program that is solved. Every relation can be formulated as a number of inequations as sketched in Table 7.3. Repeating this transformation for all clusters, $\mathcal{S}$ gets simulated ranges $\widetilde{\mathbf{Q}}$ with $\mathtt{RR}(\mathbf{Q}) = \mathtt{RR}(\widetilde{\mathbf{Q}})$. Now $\mathcal{S}$ sets $\widetilde{\mathbf{T}} = (\text{SRQ-Token}(K, \widetilde{Q_i})_{i \in [1,q']})$ which is indistinguishable according to Lemma 4. Note that $\mathcal{S}$ can recover the simulated range $\widetilde{Q_i}$ given a range token SRQ-Token$(K, \widetilde{Q_i})$, since each token contains a common IND-CPA encrypted value that can be decrypted. This feature enables $\mathcal{S}$ to simulate a consistent ciphertext sequence as discussed in the following section.

Simulating ciphertext sequence $\widetilde{\mathbf{C}}$: $\mathcal{S}$ creates a set of simulated leaves $\widetilde{\mathbf{L}}$. More particular, $\mathcal{S}$ divides $ID_{\mathbf{Q}} = \left(ID_{Q_1}, \ldots, ID_{Q'_q}\right)$ in a set $\widetilde{\mathbf{L}}$ consisting of disjoint sets, where the union of all leaves in $\widetilde{L_i} \in \widetilde{\mathbf{L}}$ still covers the same values as $ID_{\mathbf{Q}}$. Particularly, two sets ID_{Q_i} and ID_{Q_j} with $ID_{Q_i} \cap ID_{Q_j} = ID_{Q_{i,j}}$ are divided in $ID_{Q_i} \setminus ID_{Q_{i,j}}$, $ID_{Q_j} \setminus ID_{Q_{i,j}}$ and $ID_{Q_{i,j}}$. Since the simulator has already generated the

Algorithmus 7.6 : Extracting range clusters.

```
Input : $\texttt{RR}(\mathbf{Q})$
Output : List of range clusters clusters.
1: Initialize empty list clusters containing multiple range lists
2: Initialize empty list $U$ of inspected rows of $\texttt{RR}(\mathbf{Q})$
3: while $\texttt{len}(U) < q'$ do
4:     Initialize empty list $N$ that is a queue for ranges to be clustered
5:     Initialize empty list $G$ containing the intermediate cluster
6:     Add random index from $[0, q'] \setminus U$ to $N$
7:     while $N$ not empty do
8:         choose random row index $i$ from $N$
9:         for $0 < j < q$ do
10:            if $RR(\mathbf{Q}[j]) \neq \emptyset$ and $j \notin U$ then
11:                Add $j$ to $N$
12:            end
13:        end
14:        Add $i$ to $U$ and $G$
15:        Remove $i$ from $N$
16:    end
17:    Add $G$ to clusters
18: end
19: return clusters
```

corresponding ranges $\widetilde{Q_i}$ in the previous step, $\mathcal{S}$ can create consistent ranges covered by the corresponding leaves, denoted as $\widetilde{Q}(\widetilde{L_i})$. Given the set $\widetilde{\mathbf{L}}$ of simulated leaves, $\mathcal{S}$ can simulate the ciphertexts $\widetilde{\mathbf{C}} = (\widetilde{c_1}, \ldots, \widetilde{c_q})$. $\mathcal{S}$ iterates over all tuples $ID(f_i)$ for $i \in [1, q]$ and does the following:

- If there exists a leaf $\widetilde{L_j} \in \widetilde{\mathbf{L}}$ with $ID(f_i) \in \widetilde{L_j}$, then $\mathcal{S}$ choses randomly a value point $\widetilde{v_i} \xleftarrow{\$} \widetilde{Q}(\widetilde{L_i})$. $\mathcal{S}$ sets $\widetilde{c_i} = \mathsf{RPE\text{-}Enc}(k_1, \widetilde{v_i})$ and adds tuple $(\widetilde{c_i}, ID(f_i))$ to $\widetilde{\mathbf{C}}$.
- Otherwise, there exists no simulated leaf $\widetilde{L_j} \in \widetilde{\mathbf{L}}$ with $ID(f_i) \in \widetilde{L_j}$, hence the encrypted file has no match with any queried ranges. Then $\mathcal{S}$ sets $\widetilde{c_i} = \mathsf{RPE\text{-}Enc}(k_1, r)$ with random value outside of all simulated ranges: $r \xleftarrow{\$} [0, d] \setminus \bigcup_{j=1}^{q'} \widetilde{Q_j}$. The simulator adds $(\widetilde{c_i}, ID(f_i))$ to $\widetilde{\mathbf{C}}$.

Due to selective secure plaintext privacy of RPE and Lemma 4 the probability for $\mathcal{A}$ to distinguish between $\mathbf{C}$ and $\widetilde{\mathbf{C}}$ generated by $\mathcal{S}$ is negligible.

Simulating update protocols: As seen before it is possible for $\mathcal{S}$ to simulate range queries $\widetilde{\mathbf{Q}}$ from given leakage. Simulator $\mathcal{S}$ is able to simulate all update protocols on these tokens $\widetilde{\mathbf{TK}}$. Since decrypting range token $\tau_{\widetilde{Q_i}}$ is possible for the simulator, $\mathcal{S}$ can run all update queries on the simulated ranges $\widetilde{\mathbf{Q}}$. Note that these update protocols do not contain new information, but all information is already included in $\mathcal{L}(\mathbf{M}, \mathbf{Q})$. □

7.4.2 Amortized Runtime Analysis

Given a value domain with D elements and n indexed items, there exist $\sum_{i=0}^{D} i = \frac{D+D^2}{2} = O(D^2)$ different coherent ranges that can be queried[2]. That is, after D^2 different queries all possible ranges have been queried and γ consists of exactly one tree containing all possible ranges.

Obviously, in this state any repeated range query can be answered in logarithmic time. However, assuming multiple queries for the same range before γ contains exactly one tree, these repetitions may raise problems.

[2] One range of size D, two ranges of size $D-1, \ldots,$ and D ranges of size 1.

Since these repeated queries do not contain new information the server is not able to update index γ. As a result, there exist search patterns that result in linear search time: First, $O(n)$ different, not coherent ranges are queried and indexed, e.g. $\frac{n}{2}$ different queries each of range size 1. Now these ranges are repeatedly queried – in average half of all indexed queries must be checked before retrieving the answer.

We can reduce the search time for such cases implementing a cache for already queried ranges that allows constant lookup. In more detail, using a hash table keyed with deterministic range identifiers, e.g. we implement $\Pi = \left(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}\right)$ as a deterministic encryption scheme that is part of every search token. This enables the server to reduce search time for repeated range queries to be in constant time $O(1)$. We emphasize that this cache does not induce additional leakage, since an adversary can identify two tokens τ_Q, τ_R that are generated for the same range; that is, the construction is fitting with the idea of constructing comparable tokens.

The runtime for one search operation is the sum of the actual search time t_s and the update time t_u. The height of the tree is bound by $\log(D)$ and the size of an operation on one predicate-encrypted ciphertext is also $O(\log(D))$ (due to the construction described in Appendix A.3). Hence, merging two trees, extending one tree, refining one tree or rebalancing one tree can be done in $O(\log^2(D))$. Consequently, r trees can be merged in $O(r \cdot \log^2(D))$. Furthermore, since any update operation adds at least one new boundary element, there can be at most n trees. As a result, the expected update time is bounded by $t_u = O(n \cdot \log^2(D))$.

Search time depends on the newly queried range Q, i.e. if the newly queried range Q is covered by exactly one tree completely. We denote the probability of this event by $\Pr\left[Q \subseteq \Gamma_i\right]$. If this is the case, search can be performed in $O(\log^2(D))$, because searching one tree is sufficient for learning the result set. Otherwise, the complete point list must be scanned and potentially updated, resulting in search time of $O(n \log^2(D))$. As a result, the expected search time is $t_s = \Pr\left[Q \subseteq \Gamma_i\right] \cdot O(\log^2(D)) + (1 - \Pr\left[Q \subseteq \Gamma_i\right]) \cdot O(n \log^2(D))$.

Any time a range is not completely covered by a single tree at least one element in D is added to a search tree. Hence, the size of the set Γ_i increases by at least 1. Consequently, we can have at most n times a search complexity of $O(n \log^2(D))$. The maximum total time spent for these searches is $n \cdot n \log^2(D)$ This time can be amortized over the events $Q \subseteq \Gamma_i$. Let x be the total number of searches until amortization occurs. Then we have

$$\frac{n \cdot n \log^2(D)}{x} = \log^2(D)$$

We conclude that latest after n^2 searches we have achieved amortized poly-logarithmic search time.

7.4.3 Practical Benchmarks

For the evaluation of our SRQ scheme we have implemented a prototype in Python 3 using bindings for the PBC library (version 0.5.14)[3]. The runtime benchmarks have been executed on a computer running Ubuntu 14.04 with 8GB RAM and an Intel Xeon 1230v3 CPU 3.30GHz.

All files are indexed under independently sampled random value points uniformly distributed among the complete value domain. In addition, every range query is sampled randomly with a size between 1 and a defined upper limit, starting from a uniformly sampled point in the domain. The upper limit is given as *query factor* of the complete domain size, for example given a domain size 1000 and a query factor 10^{-1} the range for one query may have a length between 1 and 100. By varying the range size we can modify the probability for two ranges to intersect, hence we can influence the probability of merging and extending trees. Furthermore, we analyzed the number of trees the search index consists of in dependence of the number of preceding range queries.

[3] `https://crypto.stanford.edu/pbc/`

First, we count the average number of comparisons for one range query, i.e. how often the server executes RPE-Match. We expect this number to decrease in dependence of the ranges queried before as the search index covers a larger domain. That is, the search tree contains more values and fewer linear scans are necessary. An additional effect can be observed in dependence of the threshold t describing the number of entries per node, i.e. RPE-Match is called more often for search trees with a greater t.

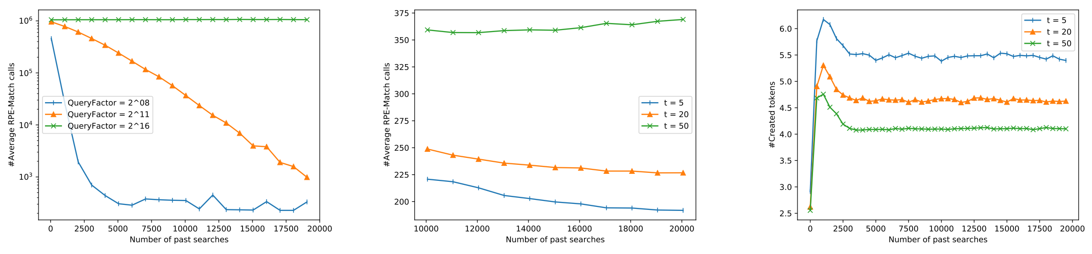
Next update costs on the client side are analyzed: we counted the number of range decryption operations and range creations. We expect this number to be smaller in dependence of node threshold t for the same number of searches. That is, the larger the number of ranges contained in one tree node the lower is the height of the tree requiring less freshly generated range tokens for an updated index.

Finally, we present micro benchmarks for encrypting one data point by running SRQ-Enc, creating a search token using SRQ-Token and checking two tokens generated for intersection. These micro benchmarks are evaluated with different domain sizes and varying security parameters. Since we use range predicate encryption as a black box, we can change the underlying implementation without modifying our construction. For demonstration purpose we implemented the schemes for secure inner-product evaluation proposed by Shen [133] and an alternative by Bishop [18] and utilized them for the construction of a range predicate encryption scheme as described in Appendix A.3.

Searching and updating trees

All measurements presented in this section are repeated five times and we state the average values. For this subsection we assume the domain $D = [0, 2^{26} - 1]$ and the number of indexed files to be 2^{20}. Furthermore, we grouped and averaged 50 values for one data point in the evaluation, e.g. the average runtime of 50 successive search queries is represented by one data point in Figure 7.3.

Modifying the maximum size of one range query results in varying probability for two ranges to intersect. As a result, the number of merge operations varies, as well as the number of different trees stored in the search index. With smaller ranges to be queried the trees cover smaller ranges, hence the server must scan the complete file list more often resulting in more RPE-Match calls, as depicted in Figure 7.3a. In this graph we do not distinguish between RPE-Match calls for comparing two range tokens as done for tree traversal or RPE-Match calls for checking if an indexed file falls within the queried range. In the worst case, there is a huge amount of trees to be scanned, each covers only a small range. Given a new small range token, all these indexed trees must be searched. If no match has been found, the complete point list must be scanned additionally, resulting in more RPE-Match calls than a linear scan of all files would require.



(a) Mean number of RPE-Match calls depending on the query factor.

(b) Mean number of RPE-Match calls depending on tree threshold t.

(c) Mean number of created tokens depending on the tree threshold t.

Figure 7.3: Mean number of different operations for one search.

Besides the query factor, we identified the node threshold t as an additional parameter that affects the number of RPE-Match calls: A greater threshold t results in more RPE-Match calls per node and consequently in the overall number of RPE-Match calls as depicted in Figure 7.3b. On the other hand, we

		Number of preceding searches				
		100	1000	5000	20,000	40,000
Query Factor	2^{-8}	82.2	144.6	1	1	1
	2^{-11}	98.8	784.2	1467	149.4	3.2
	2^{-16}	100	993.2	4213.8	17133.2	29448

Table 7.4: Mean number of trees after five runs.

can decrease the probability of a rebalancing step by increasing the number t of entries one node can hold. As a result, the server asks for help less often, hence the number of auxiliary token generations for tree updates can be decreased as presented in Figure 7.3c.

We further evaluated the performance for different query factors as summarized in Table 7.4. Here the number of indexed trees is given as a function of the number of already searched ranges and the query factor. The number of indexed trees decreases with the number of preceding searches for query factor 2^{-8} and increases for query factor 2^{-16}. That is, for a query factor 2^{-8} the probability of intersecting queries is high compared to query factor 2^{-16}. Recall that intersecting queries result in a merge of trees hence decreasing the number of indexed trees. The behavior of a border case can be observed for query factor 2^{-11}, where the number indexed trees first increases but decreases when further range queries are submitted.

Microbenchmarks

In our SRQ implementation we used the construction published by Shi et al. [134] utilizing functional encryption for inner products. For the secret key setting such a scheme was presented in Shen et al. [133] based on pairings and already proposed by Lu [106] for range queries[4]. In addition, we implemented the construction for range predicate encryption based on an alternative approach that has been published recently by Bishop et al. [18]. In the following we denote the implementation of SRQ utilizing RPE based on the scheme published by Shen, Shi and Waters [133] as $\mathsf{SRQ}^{\mathsf{SSW}}$ and the scheme published by Bishop, Jain and Kowalczyk [18] as $\mathsf{SRQ}^{\mathsf{BJK}}$.

In the following evaluation we have omitted the actual payload encryption operation using a general secret-key encryption since this is a well studied problem. For both implementations, the following two parameters affect the runtime, namely the chosen security parameter and the supported domain $[0, d]$.

Unsurprisingly, the computation time for each operation increases in dependence of the security parameter as highlighted in Table 7.5. For SSW, an increased security parameter has a stronger effect on the increase of computation time due to the underlying hardness assumption of factorization of the composite group order. The same is true for the absolute runtime numbers; BJK is assumed to be secure for prime group order, hence a smaller group order provides a comparable security level compared to SSW.

Second, the domain $[0, d]$ supported by the RPE scheme has impact on the computation time as benchmarked in Table 7.6. Particularly, the vector length to be supported by the underly IPE scheme increases logarithmically depending on the domain size supported by RPE as described in Appendix A.3. However, the underlying constructions differ in the constants they increase as a function of the vector length. According to Appendix A.3.1 the ciphertext size increases linear with the dimension of the vector for SSW, particularly, one additional vector component increases the ciphertext length by one component. In contrast, the ciphertext size for the BJK construction increases by two for each additional vector component. As

[4] We identified an implementation flaw by Lu during our evaluation: the composite group order must be chosen such that factorization is infeasible, however, Lu instantiated SSW with group order size of 128 bits. We have addressed this flaw in our implementation hence report much slower execution times.

	80 bits	128 bits	256 bits
SRQ-Enc^{SSW}	381 ms	1147 ms	5898 ms
SRQ-Token^{SSW}	9660 ms	34045 ms	143173 ms
SSW Token intersection	1553 ms	40625 ms	144512 ms
SRQ-Enc^{BJK}	18 ms	41 ms	141 ms
SRQ-Token^{BJK}	385 ms	854 ms	2466 ms
BJK Token intersection	210 ms	531 ms	1897 ms

Table 7.5: Microbenchmarks for domain size 2^{32}.

a result, the relative runtime difference for varying domain sizes is larger for BJK, however, the absolute values are orders of magnitude lower.

	2^{12}	2^{20}	2^{32}
SRQ-Enc^{SSW}	943 ms	945 ms	1147 ms
SRQ-Token^{SSW}	11685 ms	15071 ms	34045 ms
SSW Token intersection	11685 ms	14301 ms	40625 ms
SRQ-Enc^{BJK}	16 ms	26 ms	41 ms
SRQ-Token^{BJK}	114 ms	363 ms	854 ms
BJK Token intersection	58 ms	220 ms	531 ms

Table 7.6: Benchmark for fixed security parameter 128 bits.

Putting it all together

Finally, we present evaluation results for 5 runs of real search traces. We implemented the range predicate encryption with the faster inner product encryption published by Bishop et al. with 80 bits security parameter. We encrypt and index 2^{16} value points sampled independently at random out of domain $D = [0, 2^{12} - 1]$. Figure 7.4 shows the mean values of all runs, where five searches are aggregated in one bar. We measured the pure search time that is performed merely on the server side. Additionally, the needed update time was measured; here the index is updated in the interactive way, hence the client and the server are involved. Adding these times results in the complete execution time for one search. The execution time required of one linear scan of all files is depicted as the dashed line. As one can see, we profit from the interactive index construction already after 5 consecutive search operations. That is, the overall processing time for searching and updating the index as proposed in SRQ is lower than one linear scan of the complete search index.

7.5 Summary

In this chapter we proposed a novel approach for performing range queries on encrypted data. Based on the access patterns learned from previous queries the server can decrease search time for future subsequent queries by incremental updates of a search index. We analyzed this effect on the runtime theoretically and empirically and have presented a simulation based security proof. Compared to previous schemes for privacy-preserving range queries with poly-logarithmic runtime we achieve a smaller information leakage with our construction. Furthermore, our construction utilized functional encryption for inner product evaluation as a block-box functionality, so one can exchange the underlying algorithm without modifying

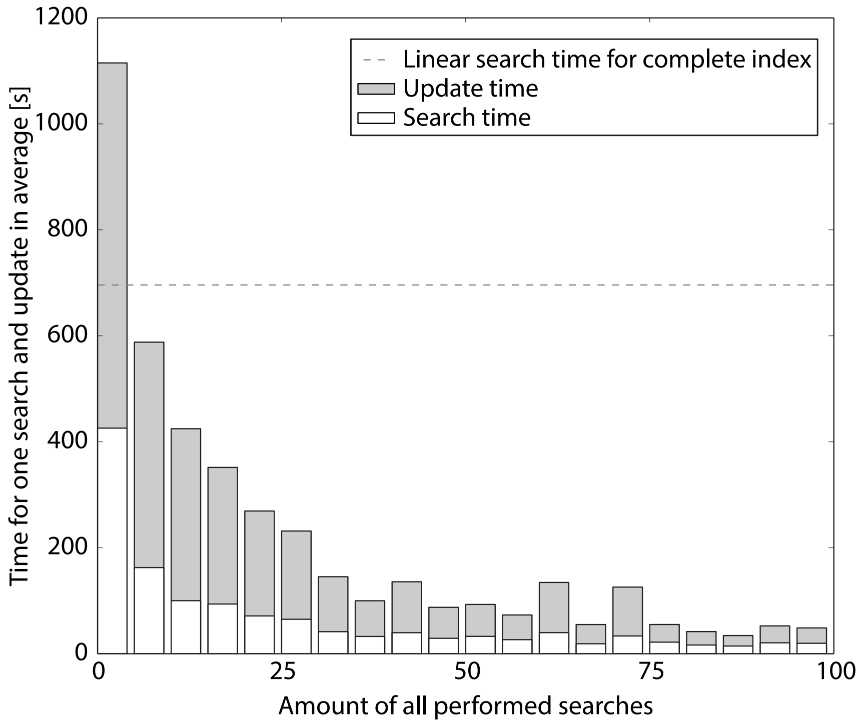


Figure 7.4: Average execution time for five searches.

our scheme as demonstrated in our implementation. As a result, the construction presented in this chapter profits from all future improvements in the are of inner-product encryption and range-predicate encryption. Based on a prototypical implementation, we demonstrate its feasibility and point out different parameters to adjust search time and complexity on the client side.

8 Substring Search

In this chapter we present a protocol for private substring search over encrypted data. Such functionality enables the database client to outsource encrypted strings and delegate substring queries to an untrusted database server. After the protocol execution between the client and the untrusted server the server returns encrypted string positions where the queried substring occurs. These encrypted string positions can then be decrypted by the client having access to the decryption key. We implement this functionality by a transformation of substring queries into range queries; hence this functionality can be implemented using techniques presented in previous Chapter 7. Due to better performance we will follow an alternative approach in this chapter based on frequency hiding order-preserving encryption also providing functionality of range queries. Further, this constructions is easy to integrate into existing databases, since no extra functionality must be implemented into the existing database. However, security properties are hard to formalize for these encryption schemes and their consequences on real-world data are difficult to foresee. In this chapter we assume a weaker attacker model in form of a snapshot attacker without capabilities to monitor all actions performed on the database and evaluate the practical consequences for encrypted text in English language.

We follow a slightly different approach in this chapter compared to previous chapters. Particularly, the remainder of this chapter is structured as follows: An overview of related work is given in Section 8.2. In Section 8.3 we present our protocols for secure substring search. First, we review the cryptographic building blocks used in our construction and introduce specific notation. Further, we present the encryption algorithm that allows to implement a protocol for substring search on encrypted data. Finally, we present different tweaks for varying scenarios. In Section 8.4 we evaluate the security properties and the performance of our construction. First, we revisit the formal security definition for frequency hiding order-preserving encryption and derive a related security notation for our construction. Additionally, we evaluate the practical consequences of this security notion for encrypted text in English language. A detailed performance evaluation demonstrates the practicability of our scheme. Further ideas how to extend and modify the presented constructions are discussed in Section 8.5. Finally, we conclude in Section 8.6.

The content of this chapter has been published in joint work with Florian Kerschbaum and Nicolas Loza at SIGMOD 2018:

- Hahn, Florian ; Loza, Nicolas ; Kerschbaum, Florian: Practical and Secure Substring Search. In: *Proceedings of the International Conference on Management of Data*, 2018 (SIGMOD)

8.1 Introduction

One filtering functionality that is still rarely addressed by security mechanisms is substring search over encrypted data. In real-world applications, this functionality is of great interest for fuzzy search, e.g. in cases where the exact spelling of the search term to be queried during runtime is unknown.

For example, recall the case of a database containing employees and their individual salary as sketched in Table 8.1 the client wishes to outsource in encrypted form. Later, the client wishes to query all employees whose surname contains a specific substring defined by the client, e.g. all employees with surname “Schmidt”, however, the client is not sure about the correct spelling, e.g. is the beginning might be spelled German starting with “Sch” or it might be spelled English beginning with “Sh”. Additionally, the name might be

EmpID	Forename	Surename
1	Harry	Maier
2	Sally	Schmitt
3	Gary	Mueller
4	Larry	Schmidt

Table 8.1: Example of plain database table *Emp*

spelled with “dt” or “tt” or another variation at the end. A SQL statement as given in the following Listing 8.1 can be used to address such uncertainty.

```
SELECT * FROM Emp WHERE surename LIKE %hmi%;
```

Listing 8.1: Example SQL query for range query on the Salary column.

yielding the result given in Table 8.2 on the client side.

EmpID	Forename	Surename
2	Sally	Schmitt
4	Larry	Schmidt

Table 8.2: Plaintext result after substring query execution stated in Listing 8.1 over table *Emp*

Although substring search is a functionality that is requested frequently by potential stakeholders for encrypted databases there exists only a little number of solutions applicable for this problem. Chase and Shen present a scheme that processes suffix trees in a private interactive protocol purely based on secure computation resulting in a scheme even secure against a malicious attacker [43]. While their solution offers a high security level, the practical deployment is debatable due to high modification efforts required on the underlying database system and the interactivity that is impractical in environments with high round-trip times. Further, even minimal database updates result in a complete re-initialization of the suffix tree. Unfortunately, no practical evaluation has been given in their work.

Another scheme presented by Faber et al. [55] is an extension of their searchable symmetric encryption (SSE) scheme supporting conjunctive queries [38]. They divide the text to be outsourced into k-grams outsourced using searchable encryption. Each substring search is then the conjunction of such k-grams with correct relative position offset. Evaluating the correct position offset requires modular exponentiations on the client side for each search query, thus hindering the deployment of this scheme on low powered client devices with limited computation power.

In this chapter we discuss the application of a frequency-hiding order-preserving encryption (FHOPE) [89] scheme for practical and secure substring searches. On the one hand, the advantage of applying such property-preserving encryption is the possibility to re-use common algorithms for building search indexes on plaintext databases without any internal system modification. More precisely, the indexing algorithms for common database systems rely on exactly these properties that are preserved even after the encryption process. This allows a straightforward and feasible integration of our scheme into existing databases without need of modifying the underlying database internals. Further, our construction allows substring queries with only one communication round.

On the other hand, security consequences of property-preserving encryption are uncertain and heavily depend on the plaintext data and the attacker model. As demonstrated recently by Naveed et al. [115], the preserved properties, namely ordering and plaintext frequency of *frequency-leaking* order-preserving encryption can be exploited by an attacker and enables him to reconstruct a large share of encrypted values. Grubbs et al. [72] even reached recovery rates up to 99%. In contrast, the experiments performed in this chapter based on the Enron dataset show that a successful attack on our *frequency-hiding* order-preserving encryption scheme tailored for secure substring search heavily depends on the background knowledge of an attacker. Particularly, the success of plaintext recovery of the bucketing attack suggested recently by Grubbs et al. [72] ranges between 1% and 15% success ratio in plaintext fragment recovery.

8.1.1 Framework

In this chapter we assume a string s with length l over an alphabet Σ to be outsourced, e.g. Σ is the set of ASCII characters and string $s \in \Sigma^l$. We denote s_i as the character of string s at position i and define the k-gram of this string with position i as sequence of characters with length k starting at position i, i.e. $s_i \dots s_{i+k-1} \in \Sigma^k$. Given a k-gram $\texttt{kg}$ contained in string s, denoted as $\texttt{kg} \in s$, we denote $\texttt{pos}_s[\texttt{kg}]$ as the ordered list of all positions where $\texttt{kg}$ occurs in s and $\texttt{len}(\texttt{pos}_s[\texttt{kg}])$ denotes the number of elements. Furthermore, we assume a total order over the alphabet Σ, so that it is possible to sort strings consisting of characters of the alphabet Σ, e.g. lexicographic order or an order that is based on the internal bit string representation.

Next, we sketch a scheme that supports substring search over encrypted data. In an initial step, the client creates a master key K by executing Gen. In the following, the construction is discussed with one string s to be encrypted and outsourced in one initial call of algorithm Enc. The output of Enc consists of two parts, particularly, one secret state ST to be kept on the client side and the encrypted search index γ to be transferred to the server. In order to execute substring queries, a protocol Query between the client and the server is executed after the initial outsourcing step. The client's input is the master key K and the secret state ST and the substring query q; the server's input is the search index γ. After a (possibly interactive multi-round) protocol execution between the client and the server the query result $\texttt{pos}_s[q]$ consisting of all string positions beginning with the queried substring is returned in encrypted form to the client.

Note that this framework addresses one static string to be encrypted and outsourced. We outline a method how the constructions presented in this chapter support databases with multiple strings in Section 8.3.2. Possible solutions enabling dynamic strings and databases and how the secret state ST might be updated are sketched in Section 8.5.1.

Formally, a substring searchable encryption scheme over an alphabet Σ consists of the following algorithms.

Definition 25 (Practical and Secure Substring Protocol)**.** *A scheme for* practical secure substring search *over encrypted strings consists of a tuple of three (possibly probabilistic) polynomial-time algorithms* $\mathsf{PSSS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Query})$ *with the following characteristics.*

$K \leftarrow \mathsf{Gen}(1^\lambda)$ *is a probabilistic algorithm that takes a security parameter* 1^λ *as input and outputs a master key* K.

$ST, \gamma \leftarrow \mathsf{Enc}(K, s)$ *is a probabilistic algorithm that takes a master key* K *and plaintext string* $s \in \Sigma^*$ *as input and outputs a secret state* ST *and a privacy-preserving search index* γ.

$\boldsymbol{I}_q \leftarrow \mathsf{Query}(K, ST, q, \gamma)$ *is a (possibly multi-round) protocol between a client and a server. The client's input is a master key* K*, a secret state* ST *and a substring* $q \in \Sigma^*$ *and the server's input is a privacy-preserving search index* γ*. The client's output is a query result* $\boldsymbol{I}_q$ *comprising of* $\texttt{pos}_s[q]$ *and the server has no output.*

Note that a scheme implementing this substring searchable encryption framework does not require an explicit decryption algorithm but can be supplemented by encrypting the complete plaintext s with a general IND-CPA secure symmetric encryption scheme. Further, we assume the query length is small compared to the message length, i.e. $\mathtt{len}(q) \ll \mathtt{len}(s)$.

8.2 Related Work

Again, we refer to Chapter 3 for general related work on searchable symmetric encryption and encrypted databases. In the following section we discuss related work particularly applicable to the functionality of substring queries on encrypted data. Our construction is implemented using frequency-hiding order-preserving encryption, hence we discuss this line of work as well.

Order-preserving encryption supports range-queries and sorting operations directly on encrypted data. The idea of such encryption scheme was formulated by Agrawal et al. [5] based on the idea of modifying the plaintext distribution into a uniformly distributed ciphertext domain. In addition, their intuitive description includes a plaintext to ciphertext mapping table as also adopted in our construction, however, their construction lacks a compression step we apply in addition to achieve smaller memory consumption of the client state.

The first formal security definition for order-preserving encryption, namely indistinguishability under ordered chosen plaintext attacks (IND-OCPA) was presented by Boldyreva et al. [21]. Additionally, they proved that no stateless order-preserving encryption scheme can fulfill this definition and gave a weaker definition of pseudorandom order-preserving function POPF that can be fulfilled by a stateless encryption scheme. The first specific construction of a stateful order-preserving encryption achieving the strong security definition of IND-OCPA has been published by Popa et al. [122]. The linear size state (linear in the number of distinct plaintexts) is then outsourced to a separate OPE server and can be retrieved in a multi-round protocol between client and OPE server. This required multi-round communication renders their solution practically inefficient due to possible network delay. An optimized version that eliminates the necessity of a separate server but still with client state linear in the number of distinct plaintexts has been published by Kerschbaum and Schröpfer [91]. All these order-preserving encryption schemes are deterministic, hence they do not only preserve the order but also the plaintext distribution is preserved for the ciphertexts. These characteristics can be exploited by an attacker who has access to auxiliary information about the deterministically OPE-encrypted plaintext data, such as publicly available statistics, e.g. census data.

In order to mitigate these kind of attacks, Kerschbaum proposed a frequency-hiding order-preserving encryption scheme that does not only fulfill the IND-OCPA security but the strictly stronger security notion indistinguishability under frequency-analyzing ordered chosen plaintext attacks [89]. Kerschbaum proved negligible probability for the rebalancing operation of the client's state (that might also induce a re-encryption process) assuming a databases with a huge number of data queries but moderate number of data inserts. Partial order-preserving encoding is an alternative scheme fulfilling this strong security notion utilizing a trusted comparison oracle and has been published recently by Roche et al. [128]. In contrast to Kerschbaum's scheme, the assumption for the underlying database operations is orthogonal for partial order-preserving encoding (POPE), that is, Roche et al. assume a huge number of data inserts but moderate number of data queries. Due to the simple integration of Kerschbaum's scheme into common DBMS and the additional comparison oracle required for POPE we decided to apply Kerschbaum's scheme for our construction. We emphasize, however, that both techniques provide functionality for range queries on encrypted data and hence are applicable for our construction; especially for database with a high number of text inserts but a moderate number of substring queries POPE is a suitable alternative.

Order-revealing encryption enables the comparison of encrypted values but is based on a special comparison function. The first formulation of such order-revealing encryption schemes has been presented by Boneh et al. [27] based on multilinear maps. Chenette et al. proposed constructions solely based on a pseudorandom function but with lower security guarantees [44] and it has been further improved by Lewi and Wu [97]. In contrast to order-preserving encryption schemes, the comparison function is not the same as on plaintext data, the ordering property is not preserved but can be revealed using this special, publicly computable comparison function. This enables the database systems to index all encrypted based on the revealed property, however, the deployment is more invasive since the special comparison function needs to be implemented into this database.

A series of practical attacks against (not only deterministic) order-preserving and order-revealing encrypted values have been published recently by Grubbs et al. [72]. Particularly the bucket attack is the first attempt of exploiting only the ordering characteristics without the need of frequency information, hence it is applicable on randomized order-preserving encryption. We will evaluation the security of our construction using this bucket attack as benchmark.

We are only aware of two different approaches that provide a direct solution for substring search functionality over encrypted data. Faber et al. [55] followed the line of work started by Cash et al. with support of conjunctive keyword search [39]. Faber divided the text to be encrypted into k-grams together with encrypted index information where the corresponding k-gram occurs and this information is outsourced using searchable encryption. A substring search is then the conjunction of such k-grams where the correct k-gram position is evaluated directly on encrypted index information. A different approach has been studied by Chase et al. [43]; they build a privacy-preserving suffix tree for the indexed text that is outsourced to the server. For each substring query, this suffix-tree is then evaluated in an interactive protocol between the client and the server.

Both approaches require extensive modifications of the underlying database system, hence prevent easy deployment into existing database systems.

8.3 Implementation

The main idea of our construction is based on the observation that prefix queries for strings can be implemented with range queries assuming a total order over the underlying character alphabet, i.e. substring queries starting at the beginning of a string. For example, assume lexicographic order and lower case alphabetic characters “a-z”then one can check if string s with length $\texttt{len}(s)$ starts with prefix q with length $\texttt{len}(q)$ by a range query with lowest value $q||a^{\texttt{len}(s)-\texttt{len}(q)}$ and with greatest value $q||z^{\texttt{len}(s)-\texttt{len}(q)}$. This approach is extended to support substring queries with intermediate positions of the string by chopping the complete string into overlapping parts, each part with varying starting position relative the overall string. Each such string chop can be used queried for a prefix using the method described before.

The constructions presented in this chapter use a symmetric encryption scheme with semantic security consisting of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as introduced in Section 2.2.1. One specific construction is based on *deterministic encryption* denoted as $\mathsf{Enc}^{\mathrm{Det}}$ with the additional property that $\mathsf{Enc}^{\mathrm{Det}}(k, m_1) = \mathsf{Enc}^{\mathrm{Det}}(sk, m_2)$ if and only if $m_1 = m_2$.

8.3.1 Order-Preserving Encryption

In addition, we make use of a frequency-hiding order-preserving encryption (FHOPE) scheme [89] that consists of three polynomial-time algorithms.

$ST \leftarrow \mathsf{Gen}^{\mathsf{FHOPE}}(1^\lambda)$ is a probabilistic algorithm that takes a security parameter λ as input and outputs a secret state ST.

$ST', y \leftarrow \mathsf{Enc}^{\mathsf{FHOPE}}(ST, x)$ is a probabilistic algorithm that takes a secret state ST and a plaintext x as input and outputs an updated secret state ST' and ciphertext y.

$x \leftarrow \mathsf{Dec}^{\mathsf{FHOPE}}(ST, y)$ is a deterministic algorithm that takes a secret state ST and a ciphertext y as input and outputs a plaintext m.

Here, the order-preserving property requires that the order of the plaintexts is preserved on the ciphertexts output by $\mathsf{Enc}^{\mathsf{FHOPE}}$, that is, $y_1 \geq y_2 \Rightarrow x_1 \geq x_2$ with $y_i \overset{\$}{\leftarrow} \mathsf{Enc}^{\mathsf{FHOPE}}(ST, x_i)$.

The formal security for frequency-hiding order-preserving encryption as introduced by Kerschbaum is based on the (not necessarily unique) randomized order of two plaintext sequences defined in the following.

Definition 26 (Randomized Order from [89])**.** *Let n be the number of not necessarily distinct plaintexts in sequence $X = x_1, \ldots, x_n$ and all for all i it holds $x_i \in \mathbb{N}$. For a* randomized order $\Gamma = \gamma_1, \ldots, \gamma_n$ *(with $\forall i : 1 \leq \gamma_i, \leq n, \forall i, j : i \neq j \Rightarrow \gamma_i \neq \gamma_j$) of sequence X it holds that*

$$\forall i, j : x_i > x_j \Rightarrow \gamma_i > \gamma_j$$

and

$$\forall i, j : \gamma_i > \gamma_j \Rightarrow x_i \geq x_j$$

Founded on this randomized order, the security experiment $\mathrm{Exp}^{\mathrm{FAOCPA}}_{\mathrm{FHOPE},\mathcal{A}}$ for a frequency-hiding order-preserving encryption scheme is defined between an adversary $\mathcal{A}$ and a challenger as follows:

Query: Adversary $\mathcal{A}$ chooses two sequences X_0, X_1 such that they have at least one common randomized order Γ.

Challenge: The challenger flips a coin b and encrypts X_b and sends this encrypted sequence back to $\mathcal{A}$.

Guess: Finally, the adversary $\mathcal{A}$ outputs a guess b'.

The experiment outputs 1 if b equals b' and 0 otherwise.

Definition 27 (IND-FAOCPA from [89])**.** *We say FHOPE is IND-FAOCPA (indistinguishable under frequency-analyzing ordered chosen plaintext attack) secure if the adversary's advantage of a correct guess is negligible, that is, if it holds that*

$$\left| \Pr\left[\mathrm{Exp}^{\mathrm{FAOCPA}}_{\mathit{FHOPE},\mathcal{A}}(\lambda) = 1 \right] - \frac{1}{2} \right|$$

is negligible in λ.

We emphasize that our construction does not require any decryption functionality so we can weaken the requirements and it is sufficient to assume a frequency-hiding order-preserving one-way function.

8.3.2 Protocol

For the goals of our encryption scheme, namely, easy deployment into existing database management systems and fast execution time for practical adoption, we propose different approaches that all provide the functionality of secure substring searches. These approaches have different characteristics in the overall runtime (as comprehensively evaluated in Section 8.4) as well as in the amount of computation overhead required on the client side, either induced by intermediate steps for an interactive protocol or after the (non-interactive) protocol execution to remove false positives. We describe the main idea in the next section from a high-level perspective and dive into more details of three different variations in the subsequent section.

Basic Encryption

Recall that for simplicity and comprehensible notation only the case of encrypting a single string is addressed in the framework stated in Definition 25. This framework can easily be extended to support encryption of multiple strings with secure substring search, that is, concatenate all strings and extend the position information for this large string with identifiers pointing to the original strings. For example in the case of encrypted databases the content of the complete column providing substring search is interpreted as one string during the preprocessing step and each position information is enriched with unique row identifiers.

Given a string s to be outsourced, the client divides this string into $\texttt{len}(s)$ overlapping k-grams denoted $g_1, \ldots, g_{\texttt{len}(s)}$. In the following, we distinguish between the k-gram sequence g_i forming the string to be encrypted, and the set of *unique* k-grams $\texttt{kg}_i$ occurring in this string. The k-grams g_j are then encrypted using a very simple FHOPE encryption implementation resulting in their corresponding FHOPE ciphertexts denoted as o_j. The general encryption algorithm can be summarized with the following steps:

- Build a map where each unique k-gram $\texttt{kg}_i$ is mapped to the list containing all position where said k-gram appears, i.e. $\texttt{pos}_s[\texttt{kg}_i]$.
- Each position list $\texttt{pos}_s[\texttt{kg}_i]$ is permuted.
- Sort this map lexicographically according to its keys, i.e. the set of all unique k-grams $\texttt{kg}_i$ appearing in string s.
- Iterating over this sorted k-gram map, all positions are enumerated, resulting in one coherent ciphertext range for each k-gram $[o_1^{(\texttt{kg}_i)}, o_{\texttt{len}(\texttt{pos}_s[\texttt{kg}_i])}^{(\texttt{kg}_i)}]$. The secret state ST must be maintained at the client side in order to enable the client to query this search index. That is, a map of all unique k-grams $\texttt{kg}_i$ together with the corresponding FHOPE-range, i.e. the lowest ciphertext $o_1^{(\texttt{kg}_i)}$ and the highest ciphertext $o_{\texttt{len}(\texttt{pos}_s[\texttt{kg}_i])}^{(\texttt{kg}_i)}$.

Each value o_j occurs exactly once, hence even the same k-gram is mapped to different order-preserving ciphertexts resulting in a frequency-hiding scheme for k-grams. Each FHOPE encrypted k-gram is additionally equipped with encrypted position information. Using a common symmetric encryption scheme the client encrypts the particular position information $\texttt{pos}_s[\texttt{kg}_i]$ for each k-gram $\texttt{kg}_i$ resulting in $c_j^{\texttt{kg}_i} = \mathsf{Enc}(K, p_j)$ for all $p_j \in \texttt{pos}_s[\texttt{kg}_i]$. The set of tuples $\left(o_j^{\texttt{kg}_i}, c_j^{\texttt{kg}_i}\right)_{j=1,\ldots,\texttt{len}(\texttt{pos}_s[\texttt{kg}_i])}$ for all unique k-grams $\texttt{kg}_i$ forms then the most simple privacy-preserving search index γ. The secret state ST kept at the client is then a compressed description of the mapping from k-grams to order-preserving ciphertexts. We evaluate the practical viability of this client state in Section 8.4.3. A formal description of the preprocessing and encryption step for one string is given in Algorithm 8.1. We assume k-gram size k to be public and omit it in the remainder of the algorithm descriptions.

A toy example for encrypting the string 'bananas' with k-gram size $k = 3$ with the resulting client state given in Table 8.3 and the encrypted search index given in Table 8.4.

Basic Tokenization

After the initial string encryption step, the resulting privacy-preserving search index γ is transferred to the database located at the untrusted server and the secure state ST is kept on the client side or outsourced into

Algorithmus 8.1 : Encryption of one string

```
Data : A string $s = s_1 \dots s_l$, $k$-gram length $k$, master key $K$
Result : @DB: search index $\gamma$. @CL: secret state $ST$.
1: $\forall i \in \{1, \dots, n-k\}$ $g_i = s_i \dots s_{i+k}$
2: $\forall i \in \{n-k+1, \dots, n\}$ $g_i = s_i \dots s_n$
3: Define pos as Map< $k$-gram, List < Integer >>
                                          /* Map each $k$-gram to its positions in $s$ */
4: Define $g$ as list containing all unique $k$-grams
5: Sort $g$ lexicographically
6: Define $ST$ as empty list
7: Define $o = 0$
                                                    /* current FHOPE ciphertext */
8: foreach $\mathtt{kg}_j \in g$ do
9:     define $start_o = o$
10:    $l \leftarrow$ shuffle(pos[$\mathtt{kg}_j$])
11:    foreach $p \in l$ do
12:        $c \leftarrow \mathsf{Enc}(sk, p)$
13:        $\gamma.add((o, c))$
14:        $o = o + 1$
15:    end
16:    $\rho^{\mathtt{kg}_i} = (start_o, o - 1)$
17:    $ST.add(\mathtt{kg}_i, \rho^{\mathtt{kg}_i})$
                                          /* $k$-gram and FHOPE ciphertext range */
18: end
19: Return $ST, \gamma$
```

kGram	start	end
ana	0	1
as_	2	2
ban	3	3
nan	4	4
nas	5	5
s__	6	6

Table 8.3: Secret State ST

FHOPE	Position
0	$\mathsf{Enc}_{\mathsf{sk}}(4)$
1	$\mathsf{Enc}_{\mathsf{sk}}(2)$
2	$\mathsf{Enc}_{\mathsf{sk}}(6)$
3	$\mathsf{Enc}_{\mathsf{sk}}(1)$
4	$\mathsf{Enc}_{\mathsf{sk}}(3)$
5	$\mathsf{Enc}_{\mathsf{sk}}(5)$
6	$\mathsf{Enc}_{\mathsf{sk}}(7)$

Table 8.4: Search Index γ

a trusted environment. Recall that the underlying database system can be any common database system like MySQL without further modifications; the secret state can be stored in another (trusted) database as well as in a plain textfile. Given a substring query $q = q_1, \dots, q_l$ the client holding the secret states tokenizes this query to be compatible with the privacy-preserving search index as described in the following.

For simplicity, first assume $l \leq k$, that is, the queried substring is at most as long as the k-gram length stated during the encryption step. The client accesses the secret state and looks up the last indexed k-gram $\mathtt{kg}_i$ that is strictly smaller than q and the first indexed k-gram $\mathtt{kg}_j$ that is strictly greater than q (according to the defined order over alphabet Σ). Since the client state is stored in a sorted structure, this search can be completed in logarithmic time, e.g. by applying binary search. The corresponding FHOPE-range $\rho^q = [\dot{\rho}^q, \bar{\rho}^q]$, beginning at $\dot{\rho}^q = o^{\mathtt{kg}_i}_{\mathtt{len(pos}[kg_i])}$ ending at $\bar{\rho}^q = o^{\mathtt{kg}_j}_1$ is then evaluated on the database and results in all encrypted position informations where this substring occurs. This encrypted result set is then transferred to the client and decrypted there.

Now we are ready for a description of the more general construction for a substring query $q = q_1 \dots q_l$ with $l > k$. In order to support such queries, the client transforms the substring query q into multiple (if possible disjoint) k-grams with size of at most k. All k-grams have to overlap or follow directly of each other, i.e. their relative distance is smaller or equal than k. For that reason, the client chooses a reference k-gram $\texttt{kg}_{\text{ref}}$, and assigns it the relative position $\delta_{\text{ref}} = 0$. The relative positions δ of all other k-gram$s$ in the queried substring are then given relatively to this reference k-gram. If any of these k-grams are not found in the secret state this k-gram was not part of the original text, hence the query cannot be a substring of the indexed text. Otherwise, the client knows that all k-grams are part of string s, but cannot be sure if they form the desired substring. Thus, the set of returned positions for each k-gram query is either decrypted on the client side and filtered for the correct positions offsets or processed in a second subsequent protocol step directly on the server side as discussed in the following Section 8.3.3.

We will use the statement

$$\boldsymbol{\tau}, \boldsymbol{\rho} \leftarrow \texttt{convert}(ST, q)$$

to refer to the process happening on the client side before the actual database queries, that is, `convert` converts the substring query into range queries. In this case, $\boldsymbol{\tau}$ contains the tuples $\tau_i = (\texttt{kg}_i, \delta_i)$ and $\boldsymbol{\rho}$ is a map where every k-gram $\texttt{kg}_i$ is mapped to a FHOPE-range ρ_i. Note that the result of this process is not unique and the same substring query can result in different k-gram queries even consisting of a different number of k-grams. For example, the outsourced string 'bananas' with k-gram size $k = 3$ results in the secret state ST and search index γ as given in Table 8.3 and in Table 8.4. Assume the client is searching for the substring "anana", then one possible tokenization is the following:

$$\{(\text{'nan'}, 0), (\text{'ana'}, -1), (\text{'ana'}, 1)\}$$
$$\{\text{'ana'} : [0, 1], \text{'nan'}[4, 4]\} \leftarrow \texttt{convert}(ST, \text{'anana'}).$$

However, this results in 3 tokens being generated, and none of them are disjoint from their neighbors. This can be simplified, for example, by generating the tokenization with maximal offset k

$$\{(\text{'ana'}, 0), (\text{'na'}, 3)\}$$
$$\{\text{'ana'} : [0, 1], \text{'na'} : [4, 5]\} \leftarrow \texttt{convert}(ST, \text{'anana'}).$$

Moreover, the covered values of the FHOPE range indicates how often a certain k-gram appears in the original text, e.g. k-grams like "the" or "of" might appear much more often than others. This observation allows the client to optimize the `convert` process with respect to the filtering overhead. The server is queried for all FHOPE-ranges $\boldsymbol{\rho}$ computed by `convert` via common database queries. These FHOPE-range queries can be evaluated efficiently on standard databases due to preserved order of the k-grams after applying Algorithm 8.1 and indexing techniques common for database and suitable for range queries such as, e.g. search trees.

8.3.3 Different Filtering Algorithms

In this section we discuss different approaches for filtering the result sets matching each FHOPE-range query individually. For demonstration purpose we give examples of the resulting database queries after the query transformation in SQL. We state three different approaches, with varying filtering complexity for the client or the server. On the one hand, the filter process can be executed solely on the client resulting in a one-round protocol, that is, all database queries can be sent in one batch without waiting for intermediate result sets but false positives might be corrected in a post-processing step. On the other hand, the server side evaluation is based on a two-round protocol but omits any post-processing (except decryption) required by the client. The performance impact is evaluated in detail in Section 8.4.3.

Position Set Reduction

In this subsection we discuss the most straightforward solution, namely, every FHOPE ciphertext-range ρ_i is queried separately on the database, resulting in position sets $\texttt{pos}_s[\texttt{kg}_i]$ for each unique k-gram $\texttt{kg}_i$. Note that these FHOPE-ciphertext range queries can be submitted in one (parallel) batch denoted as $batchQuery()$ in Protocol 8.2 with the corresponding SQL queries

```
SELECT Pos FROM Index WHERE ($\dot{\rho}_0$ < FHOPE < $\bar{\rho}_0$)
SELECT Pos FROM Index WHERE ($\dot{\rho}_1$ < FHOPE < $\bar{\rho}_1$)
  ......
```

The complete position filtering process is performed afterwards on the client side according to their position offset δ_i. In more details, given the position set $\texttt{pos}_s[\texttt{kg}_{\text{ref}}]$ of the reference k-gram, each other position set $\texttt{pos}[\texttt{kg}_i]$ is corrected by adding δ_i. The intersection of all these corrected position sets contains the actual positions the queried substring occurs $\bigcap_{(\texttt{kg}_i,\delta_i)\in\boldsymbol{\tau}}\{p+\delta_i|p\in\texttt{pos}[\texttt{kg}_i]\}$. The complete filtering algorithm is described more formally in Protocol 8.2.

Protocol 8.2 : Client Side Position Set Reduction

```
Data : @DB: search index $\gamma$.
@CL: Query $q = q_1 \dots q_l$, State $ST$
Result : Set $\mathbf{I}_q$ of matching positions for $q$
1:  $\boldsymbol{\tau}, \boldsymbol{\rho} \leftarrow \texttt{convert}(ST, q)$
                                          /* We assume all tokens exist in $ST$.  */
2:  $\texttt{pos} \leftarrow \gamma.batchQuery(\boldsymbol{\rho})$
3:  $< \texttt{kg}_{\text{ref}}, \delta_{\text{ref}} > \leftarrow \boldsymbol{\tau}.removeFirst()$
                                                              /* Here $\delta = 0$ */
4:  $\texttt{pos}_{\text{ref}} \leftarrow \boldsymbol{\rho}.get(\texttt{kg}_{\text{ref}})$
5:  foreach $< kg_i, \delta_i > \in \boldsymbol{\tau}$ do
6:      $\texttt{pos}[\texttt{kg}_i]$
                                          /* Matches for current $k$-gram  */
7:      foreach $base \in pos_{ref}$ do
8:          if not $pos[kg_i].contains(base + \delta_i)$ then
9:              $\texttt{pos}_{\text{ref}}.remove(base)$
10:         end
11:     end
12: end
13: Return $\texttt{pos}_{\text{ref}}$
```

Note that each separate k-gram query with a large result set size increases the filtering overhead on the client side linear in this result set size, hence the runtime is dominated by the k-gram with the most frequent occurrence.

Filtering on the Server Side

Next, we present a solution that decreases the filtering overhead on the client side to be linear in the result set size of the *least frequent* k-gram, but is two round interactive. For this approach we slightly modify the encryption algorithm. More particular, in Algorithm 8.1 line 12, the occurrence positions for each k-gram in the outsourced string is encrypted using a deterministic encryption scheme as outlined in Section 8.3.

Note that encrypting the positions with deterministic encryption does not weaken the security of the privacy-preserving index (since each position is unique) but provides the server the ability to check for equality on encrypted data.

In the first round, the client queries the k-gram with the FHOPE-range covering the least values as reference token $\mathtt{kg}_{\text{ref}}$. The number of values covered by a range directly correlates with the result set size as highlighted previously, that is, each k-gram occurs as many times in string s as the FHOPE-range margin. The result set containing all matching positions $\mathtt{pos}[\mathtt{kg}_{\text{ref}}]$ is returned to the client. This set of matching positions is then decrypted on the client side and further processed in order to match for remaining k-grams' positions. For each k-gram $\mathtt{kg}_i$ the offset δ_i is added $\mathtt{pos}[\mathtt{kg}_i] = \{p + \delta_i | p \in \mathtt{pos}[\mathtt{kg}_{\text{ref}}]\}$ and encrypted, resulting in a set of encrypted positions denoted as $\mathsf{Enc}^{\text{Det}}(\mathtt{pos}[\mathtt{kg}_i]) = \{\mathsf{Enc}^{\text{Det}}(p + \delta_i) | p \in \mathtt{pos}_{\text{ref}}\}$. For each k-gram the FHOPE-range ρ_i is then queried at the server together with the calculated position information $\mathsf{Enc}^{\text{Det}}(\mathtt{pos}[\mathtt{kg}_i])$ labeled as $\mathtt{queryInSet}(\rho_i, \mathsf{Enc}^{\text{Det}}(\mathtt{pos}[\mathtt{kg}_i]))$, e.g. using SQL syntax

```
SELECT Pos FROM SearchIndex WHERE $\dot{\rho}_1$ < FHOPE < $\bar{\rho}_1$
        AND Pos IN $\mathsf{Enc}^{\text{Det}}(\mathtt{pos}[\mathtt{kg}_1])$
        AND $\dot{\rho}_2$ < FHOPE < $\bar{\rho}_2$
        AND Pos IN $\mathsf{Enc}^{\text{Det}}(\mathtt{pos}[\mathtt{kg}_2])$
    ......
```

The complete protocol is formally stated in Protocol 8.3.

Protocol 8.3 : Evaluation on the Server Side

```
Data : @DB: search index $\gamma$.
@CL: Query $q = q_1 \dots q_l$, State $ST$, master key $K$
Result : Set $\mathbf{I}_q$ of matching positions for $q$.
1: $\boldsymbol{\tau}, \boldsymbol{\rho} \leftarrow \mathtt{convert}(ST, q)$
                                    /* We assume all tokens exist in $ST$.  */
2: $< \mathtt{kg}_{\text{ref}}, \delta_{\text{ref}} > \leftarrow \boldsymbol{\tau}.getSmallestRange()$
                         /* Choose $k$-gram with fewest matches as reference */
3: $\mathtt{pos}_{\text{ref}} \leftarrow \gamma.query(\rho_{\text{ref}}))$
4: foreach $< kg_i, \delta_i > \in \boldsymbol{\tau}$ do
5:     $\mathtt{pos}_i = \mathtt{pos}_{\text{ref}} + \delta_i$
                                   /* Retrieve matches for current $k$-gram  */
6:     $\mathtt{pos}_i \leftarrow \mathtt{queryInSet}(\rho_i, \mathsf{Enc}^{\text{Det}}(K, \mathtt{pos}[\mathtt{kg}_i])$
7: end
8: $correctIndices(\mathtt{pos}_{\text{ref}})$
9: return $baseMatches$
```

Fragment search

In this section we strive for reduction of the filtering overhead on the client side by reducing the probability of false positives compared to the first approach but without the necessity of a two-round protocol. Again, the client starts with the FHOPE-encryption as described in Algorithm 8.1 but now the actual k-gram positions are omitted. Instead, string s to be outsourced, is chopped in multiple string fragments f_j of length $\mathtt{len}\,(f)$ that overlap by length l, i.e. $f_j = s_i, \dots, s_{i+\mathtt{len}(f)}$ and $f_{j+1} = s_{i+\mathtt{len}(f)-l}, \dots, s_{i+2\mathtt{len}(f)-l}$. This overlapping length determines the maximal possible length for one substring query, otherwise substrings that are chopped into two different fragments are not correctly retrieved. Each fragment f_j is encrypted using a general IND-CPA secure symmetric encryption scheme and outsourced together with all FHOPE-encrypted k-grams said fragment consists of.

Given the FHOPE-ranges $\boldsymbol{\rho}$ output by `convert`(ST, q) the client queries the fragments that are indexed with FHOPE-ciphers and fall within all $\rho_i \in \boldsymbol{\rho}$ stated as queryAll($\boldsymbol{\rho}$) in Protocol 8.4. This can be realized using SQL join queries as follows:

```
SELECT fID FROM Frags WHERE ($\dot{\rho}_0$ < FHOPE < $\bar{\rho}_0$) AS T1
       JOIN
SELECT fID FROM Frags WHERE ($\dot{\rho}_1$ < FHOPE < $\bar{\rho}_1$) AS T2
      ON T1.fID = T2.fID
    ......
```

The result set consists of all encrypted string fragments that contain each k-gram in $\boldsymbol{\tau}$. As already seen in the previous section on Position Set Reduction, this result set can raise false positives, due to wrong position offsets. That is, although all k-grams occur in the string fragment they might not coherently form the queried substring q. Due to the distinction of different fragments, however, the probability of false positives is reduced. These false positives are filtered on the client side based on the decrypted fragments. The corresponding formal description of this approach is given in Protocol 8.4.

Protocol 8.4 : Partitioned Search Algorithm

```
  Data : @DB: search index $\gamma$.
  @CL: Query $q = q_1 \dots q_l$, State $ST$, master key $K$
  Result : Set $\mathbf{I}_q$ of matching positions for $q$.
1: $\boldsymbol{\tau}, \boldsymbol{\rho} \leftarrow$ convert$(ST, q)$
                                      /* We assume all tokens exist in $ST$. */
2: encFragments = $\gamma$.queryAll($\boldsymbol{\rho}$)
3: Define pos as empty List
4: foreach $encF \in encFragments$ do
5:     $fragment \leftarrow$ Dec$(K, encF)$
6:     p ← fragment.find(q)
                         /* We assume find returns $-1$ if $q$ is not contained */
7:     if $p >= 0$ then
8:         pos.add(pos + fragmentOffset)
9:     end
10: end
11: Return pos
```

8.4 Evaluation

In this section we give a short theoretical discussion on the security properties provided by our encryption algorithm. Compared to previous chapter in this thesis, we utilize a property-preserving encryption scheme in our construction; more specifically we use frequency-hiding order-preserving encryption. Since the real protection provided by property-preserving encryption heavily depends on the underlying data structure to be encrypted, we additionally evaluate the security in a practical analysis. Further, we evaluate the performance of the different approaches presented in previous Section 8.3.3 based on an implementation. We refrain from a theoretical runtime evaluation since it depends on the data structures provided by the underlying database system used for storing the encrypted search index.

8.4.1 Theoretical Security Evaluation

Our indexing scheme is IND-FAOCPA secure as stated in Definition 27 since all k-gram$s$ are ordered during the encryption step, hence in practice *all* possible k-gram sequences of length n have the same randomized order, namely $1, \ldots, n$. Following the cryptographic approach of indistinguishability proposed by Kerschbaum for FHOPE we state security of our construction founded on the following definition.

Definition 28 (IND-CPA-IOQ)**.** *Let* $\mathsf{PSSS} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Query})$ *be a scheme with support for substring search over encrypted data implementing the framework stated in Definition 25. For* PSSS *we define the security experiment* $\mathrm{Exp}^{\textit{CPA-IOQ}}_{\mathsf{PSSS},\mathcal{A}}(\lambda)$ *between adversary* $\mathcal{A}$ *and a challenger as follows:*

- *The challenger creates a master key* $K \overset{\$}{\leftarrow} \mathsf{Gen}(1^\lambda)$.
- *Adversary* $\mathcal{A}$ *chooses two strings* s_0, s_1 *with* $|s_0| = |s_1|$.
- *The challenger flips a coin b, calls* $ST_b, I_b \overset{\$}{\leftarrow} \mathsf{Enc}(K, s_b)$ *and sends* I_b *to* $\mathcal{A}$.
- *Adversary* $\mathcal{A}$ *submits two query sequences* Q_0, Q_1 *with the same length. Each sequence must be transformable into (multiple) range queries* $\boldsymbol{\rho}_0, \boldsymbol{\rho}_1$ *such that* $\boldsymbol{\rho}_0 = \boldsymbol{\rho}_1$ *(relative to* ST_b*) and result in the same sized access pattern.*
- *The challenger simulates* $\mathsf{Query}(K, ST_b, q_b, I_b)$ *and sends the transcript VIEW of these query executions to* $\mathcal{A}$.
- *Adversary* $\mathcal{A}$ *outputs a guess* b' *and the experiment outputs* 1 *if* $b = b'$.

The encryption scheme PSSS *with support for substring search over encrypted data is* indistinguishable under chosen plaintext attacks for identically ordered queries *if all probabilistic adversaries* $\mathcal{A}$ *win this experiment probability*

$$\left| \Pr\left[\mathrm{Exp}^{\textit{CPA-IOQ}}_{\mathsf{PSSS},\mathcal{A}}(\lambda) \right] - \frac{1}{2} \right|$$

that is negligible in λ.

Note that the restriction on queries (Q_0, Q_1) with one common randomized order relative to ST_0, ST_1 is required, otherwise an adversary could win the game trivially. For example, assume $k = 3$ and two strings (over the English alphabet with lexicographic order) $s_0 =$ “beefs” and $s_1 =$ “$lulua''$ resulting in $ST_0 =$ (bee, eef, efs, fs_, s__) and ST_1 =(a__, lul, lua, ua_, ulu). Two valid query sequences for the experiment are $Q_0 =$ (e__,s__) and query Q_1 =(lu_,ulu) both transformed to range queries $\boldsymbol{\rho}_0 = \boldsymbol{\rho}_1 = ([1-2],[5])$. The restriction of same sized access pattern requires that for each substring query out of set Q_b all k-grams forming these queries have the same number of occurrences.

Further, the transcript VIEW is the view of a semi-honest server, consisting of all messages sent from the client to the server.

Theorem 5. *The two round interactive protocol for substring queries over encrypted data with filtering on the server side as described in Protocol 8.3 is IND-CPA-IOQ secure, if the underlying deterministic encryption is secure and the frequency-hiding order-preserving encryption scheme is IND-FAOCPA.*

Proof. We use the security of pseudorandom permutations together with the formalization of FHOPE as stated in Definition 26 to give an intuition of the security proof for Theorem 5.

For this proof we present a sequence of games $\{\mathbb{G}_0, \mathbb{G}_{1,i}, \mathbb{G}_{2,j}\}$, each outputting a transcript VIEW$_0(b)$, VIEW$_{1,i}(b)$, VIEW$_{2,j}(b)$. The games $\mathbb{G}_{1,i}$ are hybrid games where we modify the i-th encrypted position information returned by any k-gram query. The games $\mathbb{G}_{2,j}$ are hybrid games where we modify the j-th encrypted position information never returned by any k-gram query but stored in the encrypted index. By i-th and j-th encrypted position information we assume an implicit order over ciphertexts according to their bit representation. Each game gradually differs, until the transcript of the final game is independent of the sampled bit b by the experiment, hence the adversary can only guess b' with probability $\frac{1}{2}$ in the final game. We argue that each game is indistinguishable from the previous game except with negligible probability, thus the view of the first game and the final game is also indistinguishable except with negligible probability.

$\mathbb{G}_0$: In this game we follow the experiment for IND-CPA-IOQ (cf. Definition 28) hence output the real transcript VIEW$_0(b)$ the attackers observes.

$\mathbb{G}_{1,1}$: In this game we simulate the first encrypted position information returned by any k-gram query. That is, we replace the first returned encrypted positions (both in the query result and the encrypted search index) with a randomly sampled bit string in $\{0,1\}^n$. Denote the modified transcript with VIEW$_{1,1}(b)$. Note that positions returned multiple times, e.g. because a substring query is repeated, are always replaced with the same sampled bit string.

$\mathbb{G}_{1,i}$: In this game, we simulate all encrypted position information up to the i-th value returned by any k-gram query.

$\mathbb{G}_{2,1}$: In this game we simulate the first encrypted position information stored in the encrypted search index but never returned by any k-gram query. That is, we replace the first returned encrypted position in the search index with a randomly sampled bitstring in $\{0,1\}^n$. Denote the modified transcript with VIEW$_{2,1}(b)$.

$\mathbb{G}_{2,j}$: In this game we replace the deterministic encryption of the j-th positions never been returned with randomly sampled bitstrings $\{0,1\}^n$.

The transition from one game to the next game is indistinguishable for the adversary except with negligible probability ϵ, otherwise the adversary could attack the random permutation. Denoting n as the number of replaced encrypted values, the overall probability for an adversary to distinguish $\mathbb{G}_0$ from $\mathbb{G}_{2,n}$ is $n\epsilon$. In the last game $\mathbb{G}_{2,l}$ all deterministically encrypted values are replaced with random strings and hence are independent from the sampled bit b. Since the range queries Q_0, Q_1 have the same ordering by definition of the security, this completes the proof. □

8.4.2 Practical Security Evaluation

For a better understanding of the practical implications using an IND-FAOCPA secure FHOPE scheme for outsourcing k-gram$s$, we attacked FHOPE-encrypted k-gram indexes with the best known attack on FHOPE schemes published by Grubbs et al. [72]. This attack can be performed by a snapshot attacker, for example an external attacker that has been able to download one version of the encrypted databases. However, this attacker class is not empowered to constantly monitor activities on the sever. In the following we revise the attack by Grubbs et al. called *bucketing attack* then we describe our evaluations settings and finally present the results.

The bucketing attack is based on the assumption that an attacker has access to auxiliary data with similar structure as the FHOPE-encrypted target data. That is, the attacker's auxiliary data and target data are drawn from the same value domain (in our string example the same k-gram distribution over the same alphabet Σ) with a similar underlying distribution. Given encrypted target data of length n and sufficient (i.e. with length greater than n) auxiliary data, the attacker samples n values from the auxiliary data. These values are classified corresponding to their prefix of length β, every bucket is labeled with such a prefix. Then the upper and lower bound on the rank of all elements in each bucket is calculated; following our construction these ranks are the same as their FHOPE-ciphertext values. So these buckets give an approximation of all ciphertexts that share the same prefix with length β. This data sampling and bucketing process is repeated l times and the border rank values for each bucket are averaged. Finally, the most common plaintext for each averaged bucket is the guess for the target ciphertext that falls within that averaged bucket range.

For our practical security analysis we evaluate the bucketing attack revised in the previous paragraph. Each guess by the attacker is counted as successful if the mapping from the FHOPE-ciphertexts to the corresponding k-gram is correct. The attacker's success ratio is the number of correct guesses divided by the overall FHOPE-encrypted k-grams. Each measurement has been repeated 100 times and we calculated the mean value. All our attacks are based on the Enron dataset[1]. More particular, both the auxiliary data and the challenge data is chosen out of the same dataset collection.

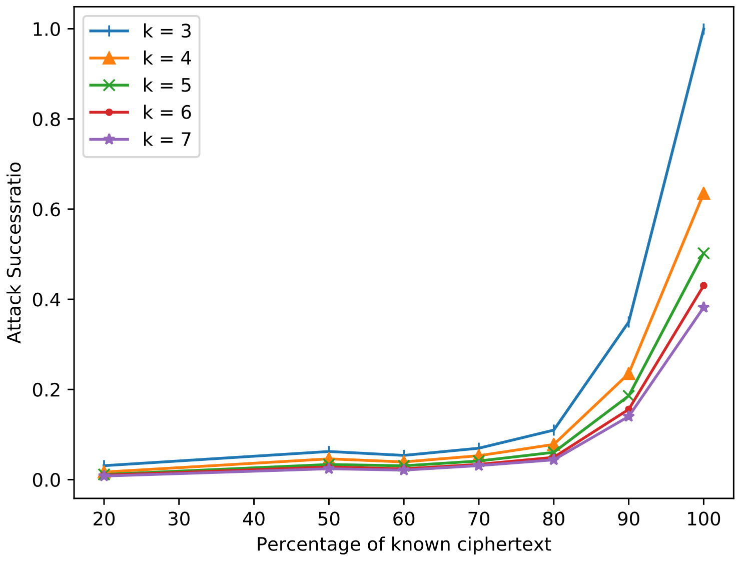


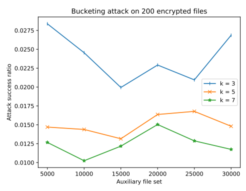
Figure 8.1: Attacker's advantage with partly know plaintext.

As a first baseline evaluation, we perform an attack where the attacker can access parts of the challenge data as auxiliary data and we increased this known part successively. In more details, we evaluate how successful the bucketing attack is with auxiliary data chosen as 500 random files and partly used *the same file set* as challenge data. We set the bucketing prefix parameter $\beta = 3$ and varied the k-gram size between 3 and 7. Note that $\beta = k = 3$ is a special case in which each bucket has only one element, hence the bucketing attack corresponds to the sorting attack on frequency-hiding order-preserving encryption as discussed by Grubbs [72]. In the case of full knowledge about the known challenge text, called *dense* knowledge, the sorting attack has 100% success rate as already highlighted by Naveed et al. [115]. The attacker's advantage for different k-gram sizes and different fractions of known plaintext is shown in Figure 8.1.

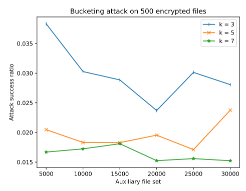
Further, we executed series of more comprehensive attacks where we fixed the dataset size for values within $\{200, 500, 1000, 2000\}$ and increased the amount of auxiliary data the attacker has access to. We evaluated the effect of increased alphabet size by filtering the text for all special characters in Figure 8.2 and 8.3, and by ignoring case-sensitivity in Figure 8.4 and 8.5. We chose different k-gram sizes k, and fixed the prefix size $\beta = 2$ with bucketing sampling process repeated $l = 100$ times as suggested in [72]. The attacker's success ratio decreases with increased k-gram size k for all data sets. For the case sensitive

[1] `https://www.cs.cmu.edu/~enron/`

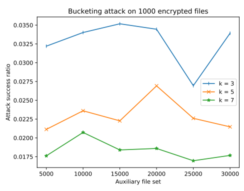
attacks we report attack success ratio between 1% and 3.5%, depending on the k-gram size chosen during the encryption step. As expected, the filtering process increases the attack success and the same is true for case-insensitive encryption since the target alphabet size decreases. That is, for the case insensitive attacks we report attack success ratio between 3% and 15%, depending on the k-gram size chosen during the encryption step. In conclusion, the attacker's success ratio heavily depends on the challenge dataset. The alphabet size and the k-gram size affects the number of different k-gram combinations and hence the plaintext space. It is intuitive that a larger plaintext space decreases the success probability of a snapshot adversary mounting the bucketing attack, however, increases the storage overhead required for the client state.



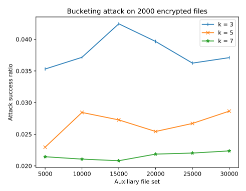
(a) 200 target files



(b) 500 target files

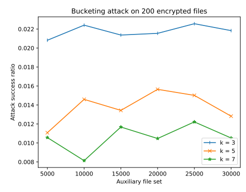


(c) 1000 target files

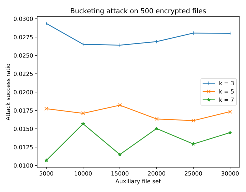


(d) 2000 target files

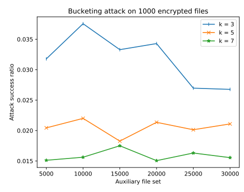
Figure 8.2: Case sensitive attacks on dataset filtered for special characters and different auxiliary dataset sizes.



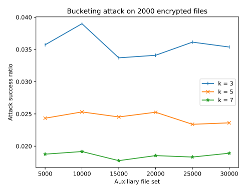
(a) 200 target files



(b) 500 target files

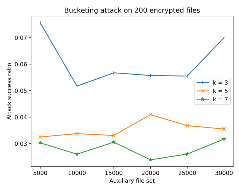


(c) 1000 target files

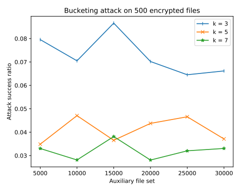


(d) 2000 target files

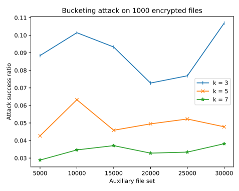
Figure 8.3: Case sensitive attacks on unfiltered dataset and different auxiliary dataset sizes.



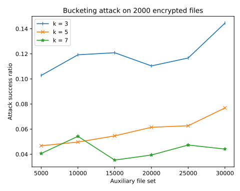
(a) 200 target files



(b) 500 target files



(c) 1000 target files

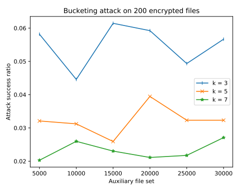


(d) 2000 target files

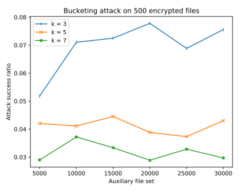
Figure 8.4: Lower case attacks on dataset filtered for special characters and different auxiliary dataset sizes.

8.4.3 Practical Benchmarks

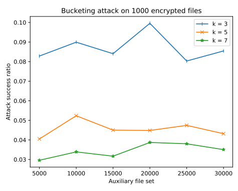
We have prototypically implemented our substring search protocols in Oracle's Java 1.8. All client operations have been executed on Windows 10 with an Intel i7 6600U CPU @ 2.6 GHz and 16 GB main memory. As database system we chose MySQL running in the same LAN with 4 Intel XEON E5-2670 each @ 2.6 GHz processors and 256 GB main memory. We ran all our evaluations on subsets of the Enron dataset; the subsets are sampled randomly for each run.



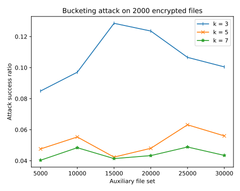
(a) 200 target files



(b) 500 target files

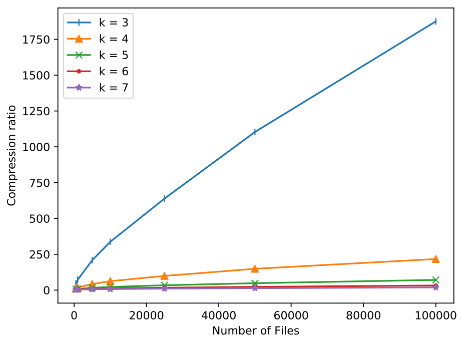


(c) 1000 target files

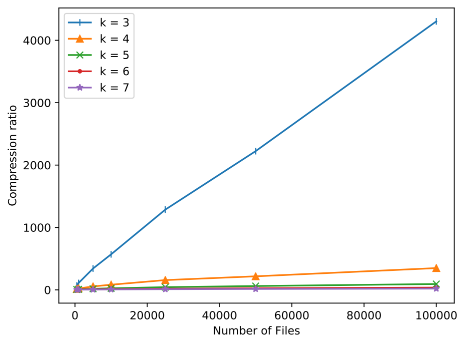


(d) 2000 target files

Figure 8.5: Lower case attacks on unfiltered dataset and different auxiliary dataset sizes.



(a) Compression ratio with no preprocessing step.



(b) Compression ratio with initial preprocessing step.

Figure 8.6: Compression ratio for different k-gram sizes and indexed files.

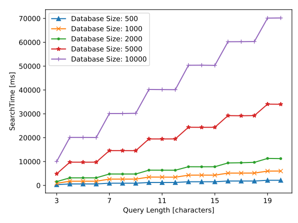
Viability of the client state

Recall that the client needs to store a secret state mapping each k-gram to a range of FHOPE ciphertexts. In a first step we analyzed the compression ratio for the client state depending on the used k-gram size and the outsourced amount of files. We have randomly sampled different numbers of files and counted the number of overall k-grams and the number of unique k-grams that are stored in the client's state. The compression ratio is the overall k-gram number divided by the number of unique k-grams. We repeated each file sampling 10 times and averaged the compression ratio for all runs. This analysis was performed with and without a preprocessing step in which all special characters have been filtered out. As we can see in Figure 8.6a without the preprocessing step and in Figure 8.6b this compression ratio highly depends on the chosen k-gram size k and the possible alphabet size since the number of all possible k-grams is $|\Sigma|^k$. For $100,000$ files without preprocessing consisted of $255,322,941$ k-gram$s$ in average and the number of unique k-grams varied from $138,242$ for $k = 3$ up to $3,410,053$ for $k = 7$. With the character filtering step before the actual outsourcing, the overall number of k-gram$s$ is $216,817,129$ and the average number of unique k-grams varied from $50,315$ for $k = 3$ up to $10,313,490$ for $k = 7$.

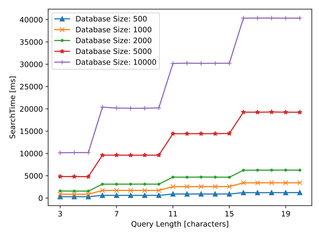
Substring Search Time

In this section we evaluate the three different filtering strategies presented in Section 8.3.3, that is, the straightforward position set reduction (Figure 8.7), the filtering on the server side based on deterministically encrypted position information (Figure 8.8) and the fragment search (Figure 8.9). All tests have been run on an unmodified MySQL database that has been accessed by the client via LAN interface and Java's JDBC driver. In order to evaluate the substring search in real-world scenarios, our measurements contain the complete query answering time including network latency and client postprocessing time. Particularly, the measured times include token generation, query transmission over the LAN interface, the MySQL database processing time and the network latency for result set transmission together with the client's intermediate or post-processing step.

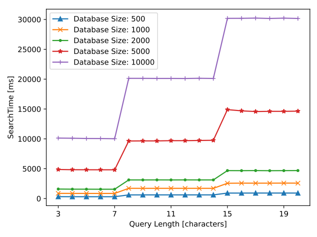
For each filtering strategy we have evaluated the substring search time for different k-gram sizes $3, 5, 7$, different query length starting with 3 up to 20 and a varying amount of indexed files out of the Enron dataset starting from 500 files up to $10,000$ files. In order to be comparable, each measurement is given for the same indexed files and the same sequence of substring queries. Furthermore, each plotted data point is the mean value of 100 values. The search times for the position set reduction as described in Section 8.3.3 are illustrated in Figure 8.7a for k-gram size 3, 8.7b for k-gram size 5, 8.7c for k-gram size 7.



(a) Indexed k-gram size 3.

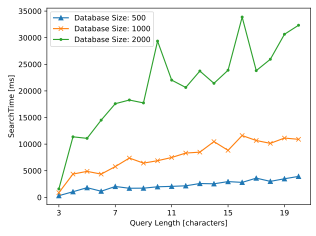


(b) Indexed k-gram size 5.

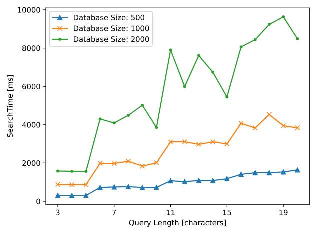


(c) Indexed k-gram size 7.

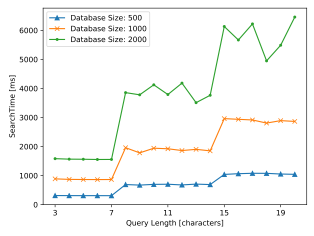
Figure 8.7: Search time for different k-gram sizes using the position set reduction filtering strategy.



(a) Indexed k-gram size 3.

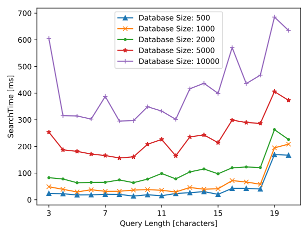


(b) Indexed k-gram size 5.

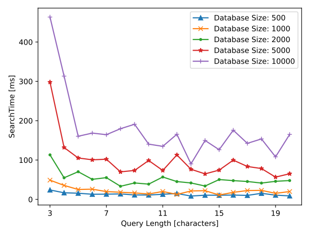


(c) Indexed k-gram size 7.

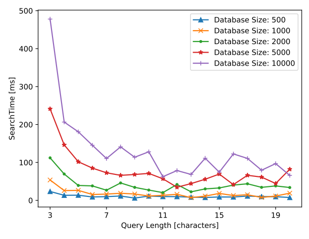
Figure 8.8: Search time for different k-gram sizes using the filtering on the server side.



(a) Indexed k-gram size 3.



(b) Indexed k-gram size 5.



(c) Indexed k-gram size 7.

Figure 8.9: Search time for different k-gram sizes using the fragment search filtering strategy.

As one can see, the search time grows linearly with increased databased size, for example, doubling the indexed database size from 5000 to 10000 indexed files increases the mean search time for query length 3 from 5 seconds to 10 seconds. This effect is independent of the used k-gram size. Nevertheless, the k-gram size influences the search time depending on the query length since the k-gram size determines the required query rounds. That is, a substring query of length l on database outsourced with k-gram size k requires $\lceil \frac{l}{k} \rceil$ FHOPE range queries and each query induces an additional scan of the complete database. Hence, the search time increases linearly with the number of required FHOPE range queries; a greater k-gram size supports longer substring queries with the same number of FHOPE (SQL) range queries but requires a larger client state as already discussed. Compared to the other filtering strategies, the processing time of this method is not affected by the result set size.

The search times for filtering strategy on the server side using deterministic encrypted position information as described in Section 13 for k-gram size $3, 5, 7$ are illustrated in Figures 8.8a, 8.8b, 8.8c. Here, both result set size and number of k-grams that form the substring query affect the search time. Recall that the position set for the reference k-gram $\mathtt{pos}_s[\mathtt{kg}_{\text{ref}}]$ is corrected *for each k-gram* using the corresponding offset value and needs to be re-encrypted and transferred again to the database system. Hence, the overhead for the encryption operations multiplied by the number of k-grams the substring query is transformed to, resulting in high search times especially for small k-gram sizes and big databases. Since the runtime results appear to be impractical for large databases we abandoned the evaluation time for databases containing a number of files greater than 2000.

The search times for the fragment search as described in Section 9 are illustrated in Figure 8.9a for k-gram size 3, 8.9b for k-gram size 5, and 8.9c for k-gram size 7. The fragment size has been set to 500 characters with 21 characters overlapping. Note that this filtering strategy can be performed with one single SQL query hence the round trip time is minimized. Further, the FHOPE-range queries are evaluated on fragment IDs instead of the complete position informations, decreasing the processing complexity (by approximately the fragment size). We identified two main parameters that affect the query time. First, the result set size has great impact, especially for short substring queries, since all (encrypted) matched fragments are transferred to the client for post processing. Second, the required number of JOIN operations that are evaluated on the database. Given a fixed k-gram size k, this correlates with the number of k-grams the substring query consists of, hence the length of substring query increases the processing time although the result set size decreases. Both effects can be observed in Figure 8.9.

8.5 Further Discussion

In this section we outline further extensions that support substring searches for dynamic databases. More particular, we discuss different approaches how to add strings to the outsourced database after the initial encryption process. Further, additional security measures are discussed in this section.

8.5.1 From Static to Dynamic Database

As discussed in Section 8.3.2, the initial preprocessing step – including encryption – is performed for the whole sensitive data collection once before the outsourcing process. Recall that the resulting output of the preprocessing step consists of the privacy-preserving search index γ and the secret state ST. This secret state ST can be exploited for adding data already available in ST while providing randomness for such added data. More precisely, we can hide the frequency information of a value x to be added by sampling a random ciphertext in the existing ciphertext range. For example, assume the client's state ST already holds five different ciphertexts for the encryption of k-gram x, that is, $\mathsf{Enc}^{\mathsf{FHOPE}}(x) \in \{a, a+1, a+2, a+3, a+4\}$. The client chooses one of these values randomly as ciphertext of value x. One the one hand, more frequent k-grams have a bigger ciphertext-domain from which the encryption value is sampled. On the other hand, less frequent k-grams have a smaller ciphertext-domain but a ciphertext is needed less frequent for these k-grams since they occur less frequent. In conclusion, this random sampling has the effect of histogram flattening for k-grams. A completely new k-gram $\mathtt{kg}_n$ induces the re-encryption of all k-grams that are greater than $\mathtt{kg}_n$,i.e. all k-grams $\mathtt{kg}_i$ with $\mathtt{kg}_i > \mathtt{kg}_n$ need to be re-encrypted. However, re-encryption is an easy task for a DBMS: lets assume a new k-gram $\mathtt{kg}$ is added, and its OPE encryption is $\mathsf{Enc}^{\mathsf{FHOPE}}(\mathtt{kg}) = x$.

So all values with greater ciphertexts need a re-encryption implemented by a simple SQL command, such as

```
UPDATE CIPHERS SET ENC = ENC + 1 WHERE ENC > x.
```

In order to minimize the necessity of this updating step, the client can reserve a bigger domain than needed for each value after indexing the initial database. For example, given a ciphertext domain for k-gram x as $\mathsf{Enc}^{\mathsf{FHOPE}}(x) \in \{a, a+1, a+2, a+3, a+4\}$ the client reserves an amount of b placeholding ciphertexts that are not used for the encryption of actual k-grams but added for later sampling. That is, the ciphertext-domain $\{(a+4)+1, \dots, (a+4)+b\}$ is added to the search index while the first ciphertext of the next real k-gram y is $(a+4)+b+1$. Since FHOPE encryption is applied to k-grams of a natural language, we can extract some statistics about x (or a prefix of x), e.g. in the case that k-gram x starts with the frequent letter ‘e’ we choose a bigger ciphertext gap b than in the case that x starts with the less frequent letter ‘q’.

Alternatively, it is always possible to create a separate search index for each subsequently indexed document collection. That is, a first document collection m_1 is indexed in a privacy-preserving index $ST_1, \gamma_1 \overset{\$}{\leftarrow} \mathsf{Enc}(K, m_1)$ and a second document collection is indexed afterwards in another privacy-preserving index $ST_2, \gamma_2 \overset{\$}{\leftarrow} \mathsf{Enc}(K, m_2)$. Now the client needs to query all different indexes separately, but we define a threshold t of different indexes. If t reached, all document collections $m_1, \dots, m_t$ are merged to $M = \bigcup_{i=1}^{t} m_i$. This merged document collection is then re-indexed to one fresh state and index $ST, \gamma \overset{\$}{\leftarrow} \mathsf{Enc}(K, M)$.

8.5.2 Increased Security

Although modular OPE as introduced by Boldyreva et al. [22] and further evaluated by Mavrofakis et al. [108] has been suggested for deterministic order-preserving encryption, the same intuition can be applied it to frequency-hiding order-preserving encryption. There are two different approaches: i) the ordering information over the alphabet are shifted with modular addition, e.g. the alphabet $\{a, \dots, z\}$ starts with $\{o, \dots, z, a, \dots, n\}$ or ii) the internal FHOPE range after building the index is shifted with a (secret) offset. This modular offset is then part of the secret state and increases the complexity of the bucketing attack described and evaluated in Section 8.4.2. While both approaches are viable in theory, the practical effect of the modular shift directly on the alphabet has a small security effect because there are only as much different shifts as the size of the alphabet.

As already noted in the introduction of this chapter, an alternative approach with an increased security level still enabling substring queries by our transformation from substrings to range queries can be achieved using functional encryption, i.e. privacy-preserving range queries such as range-predicate encryption proposed by Shen et al. [133]. Further improvements including search indexes as presented in previous Chapter 7 would also be applicable and accelerate the processing time compared to linear scans, however, still would be less efficient than the application of frequency-hiding order-preserving encryption. On the one hand, such constructions render the bucketing attack impossible, since no ordering information about the plaintext is leaked, but only the information if the plaintext falls within the queried range. As an additional advantage, such construction would allow the client to dispense with its state and hence render its deployment for dynamic databases more suitable. On the other hand, the integration overhead of such solutions increase tremendously because the database internals require modifications and well engineered indexing techniques. Further, since such constructions are founded on pairing-based cryptography, their computation overhead and hence processing time is much larger compared to the instantiation using frequency-hiding order-preserving encryption.

8.6 Summary

In this chapter we presented a novel approach for outsourcing encrypted data while providing substring search functionality with focus on the practical deployment. Our construction is based on k-gram indexing where each k-gram is encrypted using a static frequency-hiding order-preserving encryption scheme. We gave a theoretical security definition for this scheme and have evaluated the security of this privacy-preserving outsourcing techniques by practical means. That is, we attacked our construction with the strongest published attack on such encryption scheme [72] and report plaintext recovery rates between 1% and 15% based on the attacker's auxiliary knowledge about the indexed plaintext and the plaintext alphabet. Compared to previous schemes that allow privacy-preserving substring search, our scheme is easy to deploy into existing database systems without system-internal modifications. In combination with a substring search time of 98.3 ms over $10,000$ randomly chosen indexed e-mails of the Enron dataset our scheme can be deployed for practical use cases.

9 Conclusions

In the following chapter we conclude this dissertation. Section 9.1 contains a summary of this thesis. An outlook of further research directions related to query execution over encrypted databases is given in the final Section 9.2 of this dissertation.

9.1 Summary

We have presented novel constructions enabling database query execution over encrypted data. The presented constructions realize different query types, hence they are applicable for different use cases. On the one hand, we have focused on formal security properties provided by our constructions. Particularly for each query type, we have quantified an upper bound of the leakage induced by the initial encryption and the subsequent query execution in a simulation-based framework. On the other hand, we have demonstrated the practicability of such provably secure encryption schemes for database queries on encrypted data by means of practical evaluations founded on implementations. Our constructions for exact keyword search from Chapter 5, for range queries from Chapter 7 and secure database joins from Chapter 6 provide novel tradeoffs between security, performance for one specific query type.

Combining our novel encryption schemes with the existing approach introduced for CryptDB can be realized with small implementation effort. That is, our solutions can be integrated into solutions already realizing encrypted databases founded on adjustable encryption and the onion-encryption approach by supplementing novel onions encapsulating our proposed encryption schemes. These novel encryption onions are of special interest for columns with low plaintext entropy and thus would be protected insufficiently by property-preserving encryption. We have demonstrated the computational overhead of our constructions to be notably low and hence consider them to be suitable for practical adoption. Specifically, our constructions benefit from queries that repeatedly retrieve only a limited subset of the outsourced database. At the same time the security properties provided in these cases are enhanced compared to property-preserving encryption.

In addition to these novel tradeoffs between security and performance, novel functionality in form of substring search as presented in Chapter 8 further increases the use cases practically addressable by encrypted databases.

9.2 Outlook

The constructions presented in this thesis are a first stepping stone for encrypted database with advanced security properties that can refrain from property-preserving encryption. Supporting further query types and introducing privacy-preserving versions of more complex data structures remain as challenging research questions in the area of encrypted databases.

Specific functionality studied in this thesis, e.g. range queries realized with pure cryptographic pairing-based constructions still induce notable computational overhead. This overhead can be decreased drastically for implementations aided by trusted hardware such as Intel's Software Guard Extension (SGX), as demonstrated by Fuhry et al. [34]. General approaches based on trusted hardware are expected to result in huge performance increase, but require additional trust assumptions in the hardware vendor[1], e.g. Intel. A hybrid approach between cryptographic constructions and trusted hardware might decrease the required trust assumptions in the hardware vendor while decreasing the processing time compared solutions purely based on cryptographic assumptions.

Although the formal security framework adopted for searchable encryption quantifies an upper bound for the information leaked by each query execution, varying constructions state different leakage functions. Evaluating and comparing these different leakage functions is challenging even for experts. Further, the practical consequences of these individual information leakage functions remain as open question. As a result, we consider practical security evaluation for varying searchable encryption schemes to be vital as already started for substrings in Chapter 8 of this thesis. Forward and backward secrecy as introduced recently [31, 40] are novel security properties for searchable encryption, however, practical security consequences of these properties are still unclear and might be of further interest for such practical security evaluation. Finally, we believe that there is no universal construction fitting all scenarios but heavily depends on the scenario it is to be deployed and the characteristics of the data to be protected. Thus, an evaluation framework simulating different scenarios for encrypted databases might benchmark different constructions for encrypted databases and might support practitioners in the correct choice depending on the intended use case.

[1] One might argue that this trust is already required, namely, the user trusts the hardware vendor that the hardware follows the given specifications.

A Appendix

This appendix contains constructions that are used as tools for practical implementation. In order to achieve a self-contained document we briefly state these implementations enabling the interested reader to reconstruct the practical benchmarks that are a major contribution of this work. However, we refer to the corresponding references for a thorough presentation and a more detailed discussion.

A.1 Pairing Based Cryptography

All schemes presented in this appendix use pairing based cryptography. Abstractly, such constructions use three cyclic groups G_1, G_2, G_T of the same order n and a bilinear map $e : G_1 \times G_2 \to G_T$. This map e satisfies the following properties:

- The mapping is *computable*, that is, given $g_1 \in G_1$ and $g_2 \in G_2$ there is a PPT algorithm to compute $e(g_1, g_2) \in G_T$.
- The mapping is *bilinear*, that is, for any integers $x, y \in n$ we have $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$.
- The mapping is *non-degenerate*, that is, $\exists g_1 \in G_1, g_2 \in G_2$ such that $e(g_1, g_2)$ has order n in G_T.

We say that the bilinear groups are symmetric iff. $G_1 = G_2$ otherwise they are asymmetric.

A.2 Attribute Based Encryption

In the following we elaborate on the attribute-based encryption scheme used for our construction in Chapter 6 implementing secure joins.

Definition 29 (KP-ABE scheme proposed by Hohenberger and Waters [80])**.** *Let G and G_T be two multiplicative cyclic groups of prime order, $e : G \times G \to G_T$ be a bilinear map, and $H : \{0,1\}^* \to G$ be a hash function. Then a KP-ABE scheme can be implemented as follows:*

Setup(λ)*: choose a bilinear group G of prime order $p \in \Theta(2^\lambda)$. Select randomly $g \xleftarrow{\$} G$ and $\alpha \xleftarrow{\$} \mathbb{Z}_p$. Set*

$$PK = (G, p, g, e(g,g)^\alpha) \textit{ and } MK = (PK, \alpha).$$

Return (PK, MK).

Encrypt(PK, M, S)*: Let $M \in G_T$. Choose $s \xleftarrow{\$} \mathbb{Z}_p$ and compute the following:*

$$C = M \cdot e(g,g)^{\alpha s},\ \hat{C} = g^s,\ \{C_x = H(x)^s\}_{x \in S}.$$

Finally, return $CT = (C, \hat{C}, \{C_x\})$.

Note that since C_x are values derived from other column values in the same row as M, then they are assigned fixed positions within the ciphertext. In other words, all ciphertexts for a concrete table use the same internal index to store a value C_x derived from a specific column. Thus, we refer to them as C_i, where i is the aforementioned index assigned to the corresponding column.

KeyGen$(MK, \mathbf{a})$*: Let* $\mathbf{a} = \{a_{i_1}, ..., a_{i_l}\}$ *be the set of attribute restrictions we wish the key to describe, and* $i_1, ..., i_l \in \mathbb{N}$ *be the set of positions of these attributes (in our case, the positions in the ciphertext assigned to their corresponding columns). Let* $s_1, ..., s_l \xleftarrow{\$} \mathbb{N}$*. The algorithm generates* l *secret shares* $\lambda_1, ..., \lambda_l$ *from the master secret* α *using Shamir's l-from-l*[1] *linear secret-sharing scheme using the random evaluation points* $s_1, ..., s_l$*. Once these shares have been generated, the following will be executed for all* $i \in \{i_1, ..., i_l\}$*:*

$$r_i \xleftarrow{\$} \mathbb{Z}_p$$
$$D_i = g^{\lambda_i} \cdot \prod_{a \in \mathbf{a}} H(a)^{r_i}$$
$$R_i = g^{r_i}$$

The algorithm then returns $SK = ((D_{i_1}, R_{i_1}, s_1), ..., (D_{i_1}, R_{i_l}, s_l))$*. Moreover, we will assume that for every tuple* (D_i, R_i) *we can extract the actual index* $i \in i_1, ..., i_l$*.*

Decrypt(SK, CT)*: Using the indices* $s_1, ..., s_l$*, compute the coefficients* $\omega_1, ..., \omega_l$ *as described by Beimel in his thesis [13]. This construction allows us to reconstruct the secret shared across* $\lambda_1, ..., \lambda_l$*. Note that this method is executed by the untrusted party, and although she can reconstruct the* ω_i*, she does not have access to the actual values* λ_i*. Let* $\Delta = \{i_1, ..., i_l\}$ *be the set of indices referenced to by the secret key* SK*,* $h_i = H(x_i)$ *be the encoding of attribute* x_i *for column with index i in* CT*, and*

$$f(\Delta) = \prod_{i \in \Delta} h_i.$$

The algorithm computes

$$L = \prod_{i \in \Delta} C_i = \prod_{i \in \Delta} h_i^s = f(\Delta)^s.$$

The algorithm recovers the value $e(g, g)^{\alpha \cdot s}$ *by computing:*

$$\begin{aligned} e(\hat{C}, \prod_{i \in \Delta} D_i^{\omega_i}) / e(\prod_{i \in \Delta} R_i^{\omega_i}, L) &= \\ e(g^s, \prod_{i \in \Delta} g^{\lambda_i \omega_i} f(\Delta)^{r_i \omega_i}) / e(\prod_{i \in \Delta} g^{r_i \omega_i}, f(\Delta)^s) &= \\ e(g, g)^{\alpha s} \cdot e(g, f(\Delta))^{s \sum_{i \in \Delta} r_i \omega_i} / e(g, f(\Delta))^{s \sum_{i \in \Delta} r_i \omega_i} &= e(g, g)^{\alpha s} \end{aligned}$$

The decryption algorithm can then divide out this value from C *and obtain the message M.*

A.3 Range Predicate Encryption

In the following we review the construction for range predicate encryption (RPE) for domain $[0, d]$ as stated in Definition 21 based on inner product encryption (IPE). First, we give the abstract framework of inner-product encryption in the secret-key setting. Afterwards we sketch the general idea how to utilize IPE in order to implement RPE. In the last two subsection we summarize the constructions we use for our implementation supporting secure range queries with logarithmic search time.

Definition 30 (Inner product encryption (IPE))**.** *A symmetric key inner-product encryption scheme consists of the following four PPT algorithms.*

[1] Meaning that all shares are needed in order to reconstruct the master secret α.

$EK \leftarrow \mathsf{Setup}(\lambda)$ *is a probabilistic algorithm that takes the security parameter* λ *as input. It outputs a secret master key* MK.

$CT \leftarrow \mathsf{Encrypt}(MK, \vec{x})$ *is a probabilistic algorithm that takes the secret master key* MK *and a vector* $\vec{x}$ *of predefined dimension* n *as input. It outputs a ciphertext* CT.

$EK \leftarrow \mathsf{KeyGen}(MK, \vec{y})$ *is a probabilistic algorithm that takes the secret master key* MK *and a vector* $\vec{y}$ *of predefined dimension* n *as input. It outputs an evaluation key* EK.

0 *or* $1 \leftarrow \mathsf{Evaluate}(CT, EK)$ *is a deterministic algorithm taking a ciphertext* $CT \leftarrow \mathsf{Encrypt}(MK, \vec{x})$ *and an evaluation key* $EK \leftarrow \mathsf{KeyGen}(MK, \vec{y})$ *as input. It outputs* 1 *if* $\vec{x}$ *and* $\vec{y}$ *are orthogonal, that is,* $\langle \vec{x}, \vec{y} \rangle = 0$*; otherwise it outputs* 0.

It is simple to construct a RPE encryption scheme with domain $[0, d]$ given such IPE scheme supporting vectors of dimension d. Particularly, a value v can be encoded as vector $\vec{x}$ with $x_v = 1$ and $x_j = 0$ for all $j \in [1, d] \setminus \{v\}$. A range $[s, e]$ can then be encoded as vector $\vec{y}$ with $x_i = 0$ for $i \in [s, e]$ and $x_j = 1$ for $j \in [1, d] \setminus [s, e]$. It is straightforward to see that $v \in [s, e]$ if and only if $\langle \vec{x}, \vec{y} \rangle$ constructed in such way. However, such construction would require runtime that is linear in the domain size.

In the following we describe the construction decreasing this runtime to be logarithmic in the domain size as proposed by Lu [106] that we utilize in our implementation for range queries. In a first step, assume a binary tree T, where the leaves represent the domain $[0, d]$ to be encoded, particularly, each leaf represents one specific value as sketched in Figure A.1 for domain $[0, 7]$. Note that this tree has a hight that is logarithmic in the domain size, denoted as h. Further, each tree node n has a unique ID denoted as 'n'. We denote $\mathbb{P}(v)$ for a value $v \in [0, d]$ as the set of node IDs that are traversed on the path starting at the root and ending at the leaf representing v, e.g. $\mathbb{P}(4) = \{$'a', 'c', 'f', 'l'$\}$ as highlighted in Figure A.1 by circles. Further, we say a node ID covers a value v if this ID is contained in $\mathbb{P}(v)$. We denote $\mathbb{C}([s, e])$ as the minimal set of node IDs covering all values in $[s, e]$, e.g. $\mathbb{C}([0, 6]) = \{$'b', 'f', 'n' $\}$ as highlighted in Figure A.1 by squares. Given this notation, it is easy to see that $\mathbb{C}([s, e]) \cap \mathbb{P}(v) \neq \emptyset$ if and only if $v \in [s, e]$. Let denote $u(\cdot)$ as function mapping the node IDs to unique natural numbers, then we can represent the path $\mathbb{P}(v)$ by a polynomial $P_v(\cdot)$ with the node IDs' unique natural numbers as roots of this polynomial, that is, $P_v(X) = \prod_{n \in \mathbb{P}(v)} (X - u(n))$. Note that $\mathbb{P}(v)$ has h elements, thus $P(x)$ can be described as $P_v(X) = \sum_{i=0}^{h} \alpha_i X^i$; further, given $\mathbb{C}([s, e])$ with $v \in [s, e]$ then exists an $c \in \mathbb{C}([s, e])$ such that $P_v(u(c)) = 0$. Finally, this polynomial evaluation can be implemented using inner product encryption with vector $\vec{x} = (\alpha_0, \ldots, \alpha_h)$ representing $P_v(X)$ and vector $\vec{y} = \left(u(c)^0, \ldots, (c)^h\right)$ representing $c \in \mathbb{C}([s, e])$. Thus one can encode values by one vector with size that is logarithmic in the domain size, and a range can be represented by a set of vectors with size that is logarithmic in the domain size.

We refer to the work [106] published by Lu for construction details and a proof that this construction has runtime that is upper bounded to be logarithmic in the domain size and how to pad all range vector sets to prevent additional information leakage.

A.3.1 Secret Key Inner-Product Encryption

In this section we state two constructions implementing IPE, since the practical evaluation results given in Section 7.4.3 are founded on these constructions. However, we omit the correctness proof and the security proof of these constructions and refer to the corresponding publications [18, 133] for such details.

Inner-Product Encryption by Shen, Shi and Waters [133] uses symmetric bilinear groups of composite order; particularly, they use bilinear groups whose order is the product of four distinct primes. Let $\mathcal{G}$ denote a

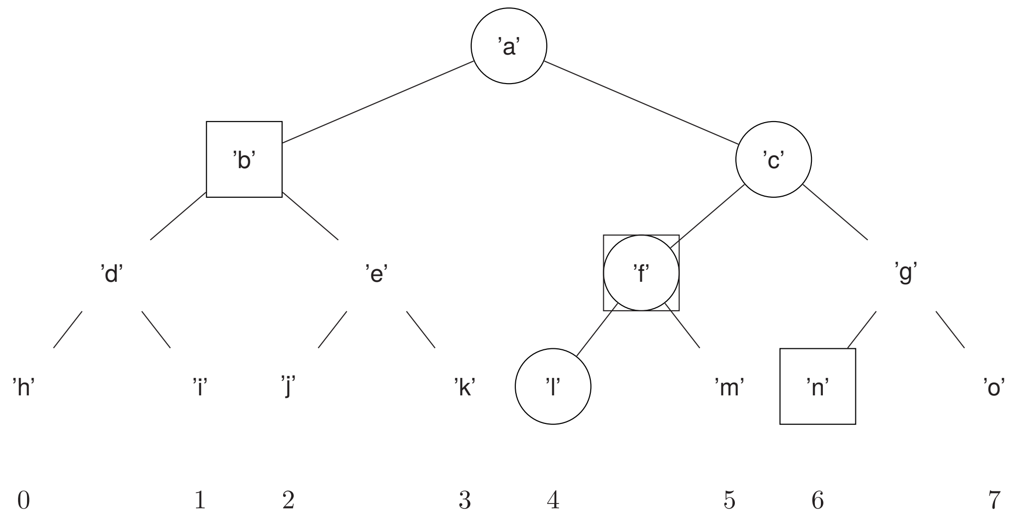


Figure A.1: Tree representing domain $[0, 7]$. Here, $\mathbb{P}(4)$ is denoted as circles and $\mathbb{C}\left([0, 6]\right)$ is denoted as square.

group generator algorithm that takes a security parameter λ as input and outputs a tuple (p, q, r, s, G, G_T, e) where p, q, r, s are distinct primes, G and G_T are two cyclic groups of order $N = pqrs$ and $e : G \times G \to G_T$ is a bilinear map as described in Section A.1. It is important, that the factorization of N is hard, that is, the primes p, q, r and s are chosen accordingly.

Now we describe the inner-product encryption proposed by Shen et al.

$MK \leftarrow \mathsf{Setup}(1^\lambda)$: on input of the security parameter λ the algorithm $(p, q, r, s, G, G_T, e) \leftarrow \mathcal{G}(1^\lambda)$ with $G = G_p \times G_q \times G_r \times G_s$. Next, it picks generators g_p, g_q, g_r, g_s of G_p, G_q, G_r, G_s, respectively. It chooses $h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i} \in G_p$ uniformly at random for $i \in [1, n]$ and outputs the secret master key $MK = (g_p, g_q, g_r, g_s, \{h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i}\}_{i=1}^n)$.

$CT \leftarrow \mathsf{Encrypt}(MK, \vec{x})$: on input of the master key $MK = (g_p, g_q, g_r, g_s, \{h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i}\}_{i=1}^n)$ and the vector $\vec{x} \in \mathbb{Z}_N^n$ this algorithm chooses independent and uniformly random elements $y, z, \alpha, \beta \in \mathbb{Z}_N$, random $S, S_0 \in G_s$ and random $R_{1,i}, R_{2,i} \in G_r$ for $i \in [1, n]$. It computes

$$C = S \cdot g_p^y, \qquad C_0 = S_0 \cdot g_p^z$$
$$\{C_{1,i} = h_{1,i}^y \cdot u_{1,i}^z \cdot g_q^{\alpha x_i} \cdot R_{1,i}, \qquad C_{2,i} = h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta x_i} \cdot R_{2,i}\}_{i=1}^n$$

and outputs the ciphertext $CT = \left(C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n\right)$.

$EK \leftarrow \mathsf{KeyGen}(MK, PP, \vec{y})$: on input of the master key $MK = (g_p, g_q, g_r, g_s, \{h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i}\}_{i=1}^n)$ and vector $\vec{y} \in \mathbb{Z}_N^n$ this algorithm chooses independent and uniformly random elements $f_1, f_2 \in \mathbb{Z}_N$ and random $r_{1,i}, r_{2,i} \in \mathbb{Z}_N$ for $i \in [1, n]$, random $R_0, R_1 \in G_r$ and random $S_{1,i}, S_{2,i} \in G_s$ for $i \in [1, n]$. It computes

$$K = R \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}, \, K_0 = R_0 \cdot \prod_{i=1}^{n} u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}}$$

$$\left\{K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_{1,v_i}} \cdot S_{1,i} \, , K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 v_i} \cdot S_{2,i}\right\}_{i=1}^n$$

and outputs the evaluation key $EK = \left(K, K_0 \, \{K_{1,i}, K_{2,i}\}_{i=1}^n\right)$.

0 or 1 $\leftarrow$ Evaluate(CT, EK): on input of the ciphertext $CT = \big(C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^{n}\big)$ and the evaluation key $EK = \big(K, K_0\, \{K_{1,i}, K_{2,i}\}_{i=1}^{n}\big)$ as computed above, this algorithm computes

$$E = e(C, K) \cdot e(C_0, K_0) \cdot \prod_{i=1}^{n} (C_{1,i}, K_{1,i}) \cdot e(C_{2,i}, K_{2,i}).$$

It outputs 1 if $E = 1$ and 0 otherwise.

Inner-Product Encryption by Bishop et al. [18] is based on asymmetric bilinear groups of prime order p. For $g \in G_i$ and $\vec{v} = (v_1, \ldots, v_n)\, \mathbb{Z}_p^n$ we abuse notation and write $g^{\vec{v}}$ denoting the n-tuple $(g^{v_1}, \ldots, g^{v_n})$ in the following description. Further, for $g_1 \in G_1, g_2 \in G_2$ and $\vec{v}, \vec{w} \in \mathbb{Z}_p^n$ we write:

$$e(g_1^{\vec{v}}, g_2^{\vec{w}}) = \prod_{i=1}^{n} e(g_1^{v_i}, g_2^{w_i}) = e(g_1, g_2)^{\langle \vec{v}, \vec{w} \rangle}$$

where the inner product is taken modulo p.

Bishop et al. utilize the concept of dual pairing vector spaces. Particularly, they propose to chose two random sets of vectors $B = \{\vec{b}_1, \ldots, \vec{b}_n\}$ and $B^* = \{\vec{b}_1^*, \ldots, \vec{b}_n^*\}$ with the constraint that they are "dual orthonormal"

$$\begin{aligned} \langle \vec{b}_i, \vec{b}_i^* \rangle &= 1 \pmod p && \text{for all } i \\ \langle \vec{b}_i, \vec{b}_j^* \rangle &= 0 \pmod p && \text{for all } i \neq j. \end{aligned}$$

We denote choosing such dual orthonormal sets randomly as $(B, B^*) \leftarrow Dual(\mathbb{Z}_p^n)$.

Now we are ready to describe the inner-product encryption proposed by Bishop et al.

$MK, PP \leftarrow$ Setup$(1^\lambda, n)$: on input of the security parameter λ and a positive integer n specifying the length of vectors this algorithm chooses asymmetric bilinear groups G_1, G_2, G_T all with prime order p. It samples generators g_1, g_2 of G_1, G_2 respectively. It samples dual orthonormal bases $B, B^* \leftarrow Dual(\mathbb{Z}_p^{2n})$ and dual orthonormal bases $D, D^* \leftarrow Dual(\mathbb{Z}_p^2)$. It outputs a secret master key $MK = B, B^* D, D^*$ and the groups G_1, G_2, G_T together with the generators g_1, g_2 and prime p as public parameters.

$CT \leftarrow$ Encrypt$(MK, PP, \vec{x})$: on input of the secret master key $MK = (B, B^*, D, D^*)$, the public parameters and a vector $\vec{x} \in \mathbb{Z}_p^n$, this algorithm chooses two independent and uniformly random elements $\alpha, \alpha' \in \mathbb{Z}_p$. It then computes

$$C_1 = g_1^{\alpha(x_1\vec{b}_1^* + \cdots + x_n\vec{b}_n^*) + \alpha'(x_1\vec{b}_{n+1}^* + \cdots + x_n\vec{b}_{2n}^*)}$$

and

$$C_2 = g_1^{\alpha\vec{d}_1^* + \alpha'\vec{d}_2^*}$$

and outputs the ciphertext $CT = (C_1, C_2)$.

$SK \leftarrow$ KeyGen$(MK, PP, \vec{y})$: on input of the secret master key $MK = (B, B^*, D, D^*)$, the public parameters and a vector $\vec{y} \in \mathbb{Z}_p^n$, this algorithm chooses to independent and uniformly random elements $\beta, \beta' \in \mathbb{Z}_p$. It then computes

$$K_1 = g_2^{\beta(y_1\vec{b}_1 + \cdots + y_n\vec{b}_n) + \beta'(y_1\vec{b}_{n+1} + \cdots + y_n\vec{b}_{2n})}$$

and

$$K_2 = g_2^{\beta \vec{d}_1 + \beta' \vec{d}_2}$$

and outputs the evaluation key $EK = (K_1, K_2)$.

m or $\perp \leftarrow \mathsf{Evaluate}(PP, CT, EK)$**:** on input of the public parameters, the ciphertext $CT = (C_1, C_2)$ and the evaluation key $EK = (K_1, K_2)$, this algorithm computers:

$$E_1 = e(C_1, K_1)$$

and

$$E_2 = e(C_2, K_2).$$

It then searches an m such that $E_2^m = E_1$ as elements of G_T and outputs m. Note that we can guarantee that the evaluation algorithm runs in polynomial time when we restrict to checking a fixed, polynomially size range of possible values for m and check $\perp$ otherwise. Particularly in our application, we only the functionality to evaluate if $\vec{x}$ and $\vec{y}$ are orthogonal, that is, if $m = \langle \vec{x}, \vec{y} \rangle = 0$ and output $\perp$ otherwise.

Bibliography

[1] Abraham, Ittai ; Fletcher, Christopher W. ; Nayak, Kartik ; Pinkas, Benny ; Ren, Ling: Asymptotically Tight Bounds for Composing ORAM with PIR. In: *IACR International Workshop on Public Key Cryptography* Springer, 2017

[2] Achenbach, Dirk ; Gabel, Matthias ; Huber, Matthias: MimoSecco: A Middleware for Secure Cloud Storage. In: *Improving Complex Systems Today*. 2011

[3] Aggarwal, Gagan ; Bawa, Mayank ; Ganesan, Prasanna ; Garcia-Molina, Hector ; Kenthapadi, Krishnaram ; Motwani, Rajeev ; Srivastava, Utkarsh ; Thomas, Dilys ; Xu, Ying: Two Can Keep a Secret: A Distributed Architecture for Secure Database Services. (2005)

[4] Agrawal, Rakesh ; Asonov, Dmitri ; Kantarcioglu, Murat ; Li, Yaping: Sovereign Joins. In: *Proceedings of the International Conference on Data Engineering*, 2006 (ICDE)

[5] Agrawal, Rakesh ; Kiernan, Jerry ; Srikant, Ramakrishnan ; Xu, Yirong: Order Preserving Encryption for Numeric Data. In: *Proceedings of the International Conference on Management of Data*, 2004 (SIGMOD)

[6] Agrawal, Shweta ; Freeman, David M. ; Vaikuntanathan, Vinod: Functional encryption for inner product predicates from learning with errors. In: *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 2011 (ASIACRYPT)

[7] Arasu, Arvind ; Blanas, Spyros ; Eguro, Ken ; Kaushik, Raghav ; Kossmann, Donald ; Ramamurthy, Ravishankar ; Venkatesan, Ramarathnam: Orthogonal Security with Cipherbase. In: *Proceedings of the Conference on Innovative Data Systems Research* (CIDR)

[8] Arasu, Arvind ; Kaushik, Raghav: Oblivious query processing. In: *arXiv preprint arXiv:1312.4012* (2013)

[9] Arnautov, Sergei ; Trach, Bohdan ; Gregor, Franz ; Knauth, Thomas ; Martin, Andre ; Priebe, Christian ; Lind, Joshua ; Muthukumaran, Divya ; O'keeffe, Dan ; Stillwell, Mark: SCONE: Secure Linux Containers with Intel SGX, 2016

[10] Attrapadung, Nuttapong ; Libert, Benoît ; De Panafieu, Elie: Expressive key-policy attribute-based encryption with constant-size ciphertexts. In: *International Workshop on Public Key Cryptography*, 2011 (WPKC)

[11] Bajaj, Sumeet ; Sion, Radu: TrustedDB: A Trusted Hardware based Database with Privacy and Data Confidentiality. In: *Proceedings of the International Conference on Management of Data*, 2011 (SIGMOD)

[12] Bater, Johes ; Elliott, Gregory ; Eggen, Craig ; Goel, Satyender ; Kho, Abel ; Rogers, Jennie: SMCQL: Secure Querying for Federated Databases. In: *Proceedings of the VLDB Endowment* (2017)

[13] Beimel, Amos: *Secure schemes for secret sharing and key distribution*. 1996

[14] Bellare, Mihir ; Boldyreva, Alexandra ; O'Neill, Adam: Deterministic and Efficiently Searchable Encryption. In: *Advances in Cryptology: Annual International Cryptology Conference*, 2007 (CRYPTO)

[15] Bellare, Mihir ; Fischlin, Marc ; O'Neill, Adam ; Ristenpart, Thomas: Deterministic Encryption: Definitional Equivalences and Constructions without Random Oracles. In: *Advances in Cryptology: Annual International Cryptology Conference*, 2008 (CRYPTO)

[16] Bellare, Mihir ; Rogaway, Phillip: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: *Proceedings of the Conference on Computer and Communications Security*, 1993 (CCS)

[17] Bellare, Mihir ; Rogaway, Phillip: Optimal Asymmetric Encryption. In: *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, 1994 (EUROCRYPT)

[18] Bishop, Allison ; Jain, Abhishek ; Kowalczyk, Lucas: Function-Hiding Inner Product Encryption. In: *Advances in Cryptology*, 2015 (ASIACRYPT)

[19] Blaze, Matt ; Bleumer, Gerrit ; Strauss, Martin: Divertible Protocols and Atomic Proxy Cryptography. (1998)

[20] Bloom, Burton H.: Space/Time Trade-offs in Hash Coding with Allowable Errors. In: *Communications of the ACM* 13 (1970), Nr. 7, S. 422–426

[21] Boldyreva, Alexandra ; Chenette, Nathan ; Lee, Younho ; O'Neill, Adam: Order-Preserving Symmetric Encryption. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2009 (EUROCRYPT)

[22] Boldyreva, Alexandra ; Chenette, Nathan ; O'Neill, Adam: Order-Preserving Encryption Revisited: Improved Security Analysis and Alternative Solutions. In: *Proceedings of the International Conference on Advances in Cryptology*, 2011 (CRYPTO)

[23] Boldyreva, Alexandra ; Fehr, Serge ; O'Neill, Adam: On Notions of Security for Deterministic Encryption, and Efficient Constructions Without Random Oracles. In: *Advances in Cryptology: Annual International Cryptology Conference*, 2008 (CRYPTO)

[24] Boneh, Dan ; Di Crescenzo, Giovanni ; Ostrovsky, Rafail ; Persiano, Giuseppe: Public key encryption with keyword search. In: *Proceedings of th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2004 (EUROCRYPT)

[25] Boneh, Dan ; Franklin, Matt: Identity-based encryption from the Weil pairing. In: *Annual international cryptology conference* Springer, 2001

[26] Boneh, Dan ; Goh, Eu-Jin ; Nissim, Kobbi: Evaluating 2-DNF formulas on ciphertexts. In: *Theory of Cryptography Conference*, 2005 (TCC)

[27] Boneh, Dan ; Lewi, Kevin ; Raykova, Mariana ; Sahai, Amit ; Zhandry, Mark ; Zimmerman, Joe: Semantically Secure Order-Revealing Encryption: Multi-Input Functional Encryption Without Obfuscation. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2015 (EUROCRYPT)

[28] Boneh, Dan ; Sahai, Amit ; Waters, Brent: Functional encryption: Definitions and challenges. In: *Proceedings of the Conference on Theory of Cryptography*, 2011 (TCC)

[29] Boneh, Dan ; Waters, Brent: Conjunctive, subset, and range queries on encrypted data. In: *Proceedings of the Theory of Cryptography Conference*, 2007 (TCC)

[30] Bösch, Christoph ; Hartel, Pieter ; Jonker, Willem ; Peter, Andreas: A survey of provably secure searchable encryption. In: *ACM Computing Surveys (CSUR)* 47 (2015), Nr. 2

[31] Bost, Raphael: Σ $o\varphi o\varsigma$: Forward Secure Searchable Encryption. In: *Proceedings of the Conference on Computer and Communications Security*, 2016 (CCS)

[32] Brakerski, Zvika ; Gentry, Craig ; Vaikuntanathan, Vinod: (Leveled) fully homomorphic encryption without bootstrapping. In: *ACM Transactions on Computation Theory (TOCT)* 6 (2014), Nr. 3

[33] Brakerski, Zvika ; Vaikuntanathan, Vinod: Efficient fully homomorphic encryption from (standard) LWE. In: *SIAM Journal on Computing* 43 (2014), Nr. 2

[34] Brasser, Ferdinand ; Hahn, Florian ; Kerschbaum, Florian ; Sadeghi, Ahmad-Reza ; Fuhry, Benny ; Bahmani, Raad: HardIDX: Practical and Secure Index with SGX. In: *Proceedings of the Conference on Data and Applications Security and Privacy*, 2017 (DBSec)

[35] Brenner, Stefan ; Wulf, Colin ; Goltzsche, David ; Weichbrodt, Nico ; Lorenz, Matthias ; Fetzer, Christof ; Pietzuch, Peter ; Kapitza, Rüdiger: SecureKeeper: Confidential ZooKeeper using Intel SGX. In: *Proceedings of the 17th International Middleware Conference*, 2016

[36] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: The Random Oracle Methodology, Revisited. In: *Journal of the ACM* (2004)

[37] Carbunar, Bogdan ; Sion, Radu: Toward Private Joins on Outsourced Data. In: *IEEE Transactions on Knowledge and Data Engineering* (2012)

[38] Cash, David ; Jaeger, Joseph ; Jarecki, Stanislaw ; Jutla, Charanjit ; Krawczyk, Hugo ; Rosu, Marcel ; Steiner, Michael: Dynamic Searchable Encryption in Very-Large Databases: Data Structures and Implementation. In: *Proceedings of the Network and Distributed System Security Symposium*, 2014 (NDSS)

[39] Cash, David ; Jarecki, Stanislaw ; Jutla, Charanjit ; Krawczyk, Hugo ; Rosu, Marcel-Catalin ; Steiner, Michael: Highly-Scalable Searchable Symmetric Encryption with Support for Boolean Queries. In: *Advances in Cryptology: Annual International Cryptology Conference*, 2013 (CRYPTO)

[40] Chamani, Javad G. ; Papadopoulos, Dimitrios ; Papamanthou, Charalampos ; Jalili, Rasool: New Constructions for Forward and Backward Private Symmetric Searchable Encryption. In: *Proceedings of the Conference on Computer and Communications Security*, 2018 (CCS)

[41] Chang, Yan-Cheng ; Mitzenmacher, Michael: Privacy Preserving Keyword Searches on Remote Encrypted Data. In: *Applied Cryptography and Network Security*, 2005 (ACNS)

[42] Chase, Melissa ; Kamara, Seny: Structured encryption and controlled disclosure. In: *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 2010 (ASIACRYPT)

[43] Chase, Melissa ; Shen, Emily: Substring-Searchable Symmetric Encryption. In: *Proceedings on Privacy Enhancing Technologies* (2015)

[44] Chenette, Nathan ; Lewi, Kevin ; Weis, Stephen A. ; Wu, David J.: Practical Order-Revealing Encryption with Limited Leakage. In: *Proceedings of International Conference on Fast Software Encryption*, 2016

[45] Ciriani, Valentina ; Di Vimercati, Sabrina De C. ; Foresti, Sara ; Jajodia, Sushil ; Paraboschi, Stefano ; Samarati, Pierangela: Fragmentation and Encryption to Enforce Privacy in Data Storage. In: *Proceedings of the European Symposium on Research in Computer Security*, 2007 (ESORICS)

[46] Codd, Edgar F.: A relational model of data for large shared data banks. In: *Communications of the ACM* 13 (1970), Nr. 6

[47] Cormen, T ; Leiserson, C ; Rivest, R ; Stein, C: *Introduction to Algorithms*. MIT press, 2009

[48] Costan, Victor ; Devadas, Srinivas: Intel SGX Explained. In: *IACR Cryptology ePrint Archive 2016/086* (2016)

[49] Curtmola, Reza ; Garay, Juan ; Kamara, Seny ; Ostrovsky, Rafail: Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions. In: *Proceedings of the Conference on Computer and Communications Security*, 2006 (CCS)

[50] De Cristofaro, Emiliano ; Kim, Jihye ; Tsudik, Gene: Linear-complexity private set intersection protocols secure in malicious model. In: *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, 2010 (ASIACRYPT)

[51] Demertzis, Ioannis ; Papadopoulos, Stavros ; Papapetrou, Odysseas ; Deligiannakis, Antonios ; Garofalakis, Minos: Practical Private Range Search Revisited. (2016)

[52] Dwork, Cynthia: Differential Privacy: A Survey of Results. In: *International Conference on Theory and Applications of Models of Computation*, 2008 (TAMC)

[53] Dwork, Cynthia ; Roth, Aaron: The Algorithmic Foundations of Differential Privacy. In: *Foundations and Trends in Theoretical Computer Science* (2014)

[54] El Gamal, Taher: A public key cryptosystem and a signature scheme based on discrete logarithms. In: *Advances in Cryptology*, 1985 (CRYPTO)

[55] Faber, Sky ; Jarecki, Stanislaw ; Krawczyk, Hugo ; Nguyen, Quan ; Rosu, Marcel ; Steiner, Michael: Rich Queries on Encrypted Data: Beyond Exact Matches. In: *"Proceedings of the European Symposium on Research in Computer Security"*, 2015 (ESORICS)

[56] Fagin, Ronald ; Naor, Moni ; Winkler, Peter: Comparing information without leaking it. In: *Communications of the ACM* 39 (1996), Nr. 5, S. 77–85

[57] Fisch, Ben ; Vinayagamurthy, Dhinakaran ; Boneh, Dan ; Gorbunov, Sergey: Iron: functional encryption using Intel SGX. In: *Proceedings of the Conference on Computer and Communications Security*, 2017 (CCS)

[58] Fox, Armando ; Griffith, Rean ; Joseph, Anthony ; Katz, Randy ; Konwinski, Andrew ; Lee, Gunho ; Patterson, David ; Rabkin, Ariel ; Stoica, Ion: Above the Clouds: A Berkeley View of Cloud Computing. In: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28 (2009), Nr. 13

[59] Freedman, Michael ; Nissim, Kobbi ; Pinkas, Benny: Efficient Private Matching and Set Intersection. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2004 (EUROCRYPT)

[60] Garg, Sanjam ; Gentry, Craig ; Halevi, Shai ; Raykova, Mariana ; Sahai, Amit ; Waters, Brent: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: *SIAM Journal on Computing* 45 (2016), Nr. 3

[61] Gentry, Craig u. a.: Fully homomorphic encryption using ideal lattices. In: *STOC* Bd. 9, 2009, S. 169–178

[62] Gentry, Craig ; Halevi, Shai ; Smart, Nigel P.: Homomorphic Evaluation of the AES Circuit. In: *Advances in Cryptology: Annual International Cryptology Conference*. 2012 (CRYPTO)

[63] Gentry, Craig ; Halevi, Shai ; Smart, Nigel P.: *Homomorphic Evaluation of the AES Circuit*. IACR Cryptology ePrint Archive, Report 2012/099, 2012

[64] Goh, Eu-Jin: Secure Indexes / IACR Cryptology ePrint Archive 2003/216. 2003. – Forschungsbericht

[65] Goldreich, Oded ; Ostrovsky, Rafail: Software Protection and Simulation on Oblivious RAMs. In: *Journal of the ACM* 43 (1996), Nr. 3

[66] Goldwasser, Shafi ; Kalai, Yael T. ; Popa, Raluca A. ; Vaikuntanathan, Vinod ; Zeldovich, Nickolai: How to run turing machines on encrypted data. In: *Advances in Cryptology: Annual International Cryptology Conference*, 2013 (CRYPTO)

[67] Goldwasser, Shafi ; Micali, Silvio: Probabilistic Encryption and How to play Mental Poker Keeping Secret All Partial Information. In: *Proceedings of the Symposium on Theory of Computing*, 1982

[68] Goldwasser, Shafi ; Micali, Silvio ; Rackoff, Charles: The Knowledge Complexity of Interactive Proof Systems. In: *SIAM Journal on Computing* 18 (1989), Nr. 1

[69] Golle, Philippe ; Staddon, Jessica ; Waters, Brent: Secure conjunctive keyword search over encrypted data. In: *Proccedings of the International Conference on Applied Cryptography and Network Security*, 2004 (ACNS)

[70] Goyal, Vipul ; Pandey, Omkant ; Sahai, Amit ; Waters, Brent: Attribute-based encryption for fine-grained access control of encrypted data. In: *Proceedings of the Conference on Computer and Communications Security*, 2006 (CCS)

[71] Grofig, Patrick ; Haerterich, Martin ; Hang, Isabelle ; Kerschbaum, Florian ; Kohler, Mathias ; Schaad, Andreas ; Schroepfer, Axel ; Tighzert, Walter: Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data. In: *Sicherheit*, 2014

[72] Grubbs, Paul ; Sekniqi, Kevin ; Bindschaedler, Vincent ; Naveed, Muhammad ; Ristenpart, Thomas: Leakage-abuse attacks against order-revealing encryption. (2017)

[73] Guttman, Antonin: R-trees: a dynamic index structure for spatial searching. In: *Proceedings of the International Conference on Management of Data*, 1984 (SIGMOD)

[74] Hacigümüs, Hakan ; Iyer, Balakrishna R. ; Li, Chen ; Mehrotra, Sharad: Executing SQL over Encrypted Data in the Database-Service-Provider Mode. In: *Proceedings of the International Conference on Management of Data*, 2002 (SIGMOD)

[75] Hacigümüs, Hakan ; Mehrotra, Sharad ; Iyer, Balakrishna R.: Providing Database as a Service. In: *Proceedings of the International Conference on Data Engineering*, 2002 (ICDE)

[76] Hahn, Florian ; Kerschbaum, Florian: Searchable Encryption with Secure and Efficient Updates. In: *Proceedings of the Conference on Computer and Communications Security*, 2014 (CCS)

[77] Hahn, Florian ; Kerschbaum, Florian: Poly-Logarithmic Range Queries on Encrypted Data with Small Leakage. In: *Proceedings of the ACM on Cloud Computing Security Workshop*, 2016 (CCSW)

[78] Hahn, Florian ; Loza, Nicolas ; Kerschbaum, Florian: Encrypted Database Joins with Fine Granular Security, ???? (In Preparation)

[79] Hahn, Florian ; Loza, Nicolas ; Kerschbaum, Florian: Practical and Secure Substring Search. In: *Proceedings of the International Conference on Management of Data*, 2018 (SIGMOD)

[80] Hohenberger, Susan ; Waters, Brent: Attribute-based encryption with fast decryption. In: *Public-Key Cryptography*. 2013 (PKC)

[81] Hore, Bijit ; Mehrotra, Sharad ; Tsudik, Gene: A Privacy-Preserving Index for Range Queries. In: *Proceedings of the 30th International Conference on Very Large Data Bases*, 2004 (VLDB)

[82] Islam, Mohammad ; Kuzu, Mehmet ; Kantarcioglu, Murat: Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In: *Proceedings of the Network and Distributed System Security Symposium*, 2012 (NDSS)

[83] Kamara, Seny ; Papamanthou, Charalampos: Parallel and Dynamic Searchable Symmetric Encryption. In: *Financial Cryptography and Data Security*. 2013

[84] Kamara, Seny ; Papamanthou, Charalampos ; Roeder, Tom: Dynamic Searchable Symmetric Encryption. In: *Proceedings of the Conference on Computer and Communications Security*, 2012 (CCS)

[85] Katz, Jonathan ; Lindell, Yehuda: *Introduction to Modern Cryptography*. CRC press, 2014

[86] Katz, Jonathan ; Sahai, Amit ; Waters, Brent: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*

[87] Kerschbaum, Florian: Collusion-resistant outsourcing of private set intersection. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012 (SAC)

[88] Kerschbaum, Florian: Outsourced Private Set Intersection Using Homomorphic Encryption. In: *Proceedings of the Symposium on Information, Computer and Communications Security*, 2012 (ASIACCS)

[89] Kerschbaum, Florian: Frequency-hiding order-preserving encryption. In: *Proceedings of the Conference on Computer and Communications Security*, 2015 (CCS)

[90] Kerschbaum, Florian ; Härterich, Martin ; Grofig, Patrick ; Kohler, Mathias ; Schaad, Andreas ; Schröpfer, Axel ; Tighzert, Walter: Optimal Re-Encryption Strategy for Joins in Encrypted Databases. In: *Proceedings of the Conference on Data and Applications Security and Privacy*, 2013 (DBSec)

[91] Kerschbaum, Florian ; Schröpfer, Axel: Optimal average-complexity ideal-security order-preserving encryption. In: *Proceedings of the Conference on Computer and Communications Security*, 2014 (CCS)

[92] Kiayias, Aggelos ; Papadopoulos, Stavros ; Triandopoulos, Nikos ; Zacharias, Thomas: Delegatable pseudorandom functions and applications. In: *Proceedings of the Conference on Computer and Communications Security*, 2013 (CCS)

[93] Koblitz, Neal ; Menezes, Alfred J.: The random oracle model: a twenty-year retrospective. In: *Designs, Codes and Cryptography* (2015)

[94] Kocher, Paul ; Genkin, Daniel ; Gruss, Daniel ; Haas, Werner ; Hamburg, Mike ; Lipp, Moritz ; Mangard, Stefan ; Prescher, Thomas ; Schwarz, Michael ; Yarom, Yuval: Spectre attacks: Exploiting speculative execution. In: *arXiv preprint arXiv:1801.01203* (2018)

[95] Kushilevitz, Eyal ; Ostrovsky, Rafail: Replication is not needed: Single database, computationally-private information retrieval. In: *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on* IEEE, 1997, S. 364–373

[96] Lacharité, Marie-Sarah ; Minaud, Brice ; Paterson, Kenneth G.: Improved reconstruction attacks on encrypted data using range query leakage. In: *Proceedings of the Symposium on Security and Privacy*, 2018 (S&P)

[97] Lewi, Kevin ; Wu, David J.: Order-Revealing Encryption: New Constructions, Applications, and Lower Bounds. In: *Proceedings of the Conference on Computer and Communications Security*, 2016 (CCS)

[98] Lewko, Allison ; Okamoto, Tatsuaki ; Sahai, Amit ; Takashima, Katsuyuki ; Waters, Brent: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010 (EUROCRYPT)

[99] Lewko, Allison ; Waters, Brent: Decentralizing attribute-based encryption. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011 (EUROCRYPT)

[100] Lewko, Allison ; Waters, Brent: Unbounded HIBE and attribute-based encryption. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2011 (EUROCRYPT)

[101] Li, Ninghui ; Li, Tiancheng ; Venkatasubramanian, Suresh: t-closeness: Privacy beyond k-anonymity and l-diversity. In: *Proceedings of the International Conference on Data Engineering*, 2007 (ICDE)

[102] Li, Yaping ; Chen, Minghua: Privacy Preserving Joins. In: *Proceedings of the International Conference on Data Engineering*, 2008 (ICDE)

[103] Lindell, Yehuda: How To Simulate It - A Tutorial on the Simulation Proof Technique. Version: 2016. `https://eprint.iacr.org/2016/046`. 2016 (046). – Forschungsbericht

[104] Lipp, Moritz ; Schwarz, Michael ; Gruss, Daniel ; Prescher, Thomas ; Haas, Werner ; Mangard, Stefan ; Kocher, Paul ; Genkin, Daniel ; Yarom, Yuval ; Hamburg, Mike: Meltdown. In: *arXiv preprint arXiv:1801.01207* (2018)

[105] Loza, Nicolas: *Implementing Secure Join Operations over Encrypted Databases with Low Information Leakage*, Karlsruher Institute für Technologie (KIT), Master's Thesis, 2017

[106] Lu, Yanbin: Privacy-preserving logarithmic-time search on encrypted data in cloud. In: *Proceedings of the Network and Distributed System Security Symposium*, 2012 (NDSS)

[107] Machanavajjhala, Ashwin ; Gehrke, Johannes ; Kifer, Daniel ; Venkitasubramaniam, Muthuramakrishnan: l-Diversity: Privacy Beyond k-Anonymity. In: *Proceedings of the International Conference on Data Engineering*, 2006 (ICDE)

[108] Mavroforakis, Charalampos ; Chenette, Nathan ; O'Neill, Adam ; Kollios, George ; Canetti, Ran: Modular Order-Preserving Encryption, Revisited. In: *Proceedings of the International Conference on Management of Data*, 2015 (SIGMOD)

[109] Mayberry, Travis ; Blass, Erik-Oliver ; Chan, Agnes H.: Efficient Private File Retrieval by Combining ORAM and PIR. In: *Proceedings of the Network and Distributed System Security Symposium*, 2014 (NDSS)

[110] McKeen, Frank ; Alexandrovich, Ilya ; Berenzon, Alex ; Rozas, Carlos V. ; Shafi, Hisham ; Shanbhogue, Vedvyas ; Savagaonkar, Uday R.: Innovative Instructions and Software Model for Isolated Execution. In: *HASP ISCA* 10 (2013)

[111] McSherry, Frank ; Talwar, Kunal: Mechanism Design via Differential Privacy. In: *Symposium on Foundations of Computer Science*, 2007 (FOCS)

[112] Micali, Silvio: Computationally Sound Proofs. In: *SIAM Journal on Computing* 30 (2000), Nr. 4

[113] Narayanan, Arvind ; Shmatikov, Vitaly: Robust De-anonymization of Large Sparse Datasets. In: *Proceedings of the Symposium on Security and Privacy*, 2008 (S&P)

[114] Naveed, Muhammad: The Fallacy of Composition of Oblivious RAM and Searchable Encryption. In: *IACR Cryptology ePrint Archive 2015/668* (2015)

[115] Naveed, Muhammad ; Kamara, Seny ; Wright, Charles V.: Inference Attacks on Property-Preserving Encrypted Databases. In: *Proceedings of the Conference on Computer and Communications Security*, 2015 (CCS)

[116] Naveed, Muhammad ; Prabhakaran, Manoj ; Gunter, Carl A.: Dynamic Searchable Encryption via Blind Storage. In: *Proceedings of the Symposium on Security and Privacy*, 2014 (S&P)

[117] Okamoto, Tatsuaki: Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In: *Advances in Cryptology: Annual International Cryptology Conference*, 1992 (CRYPTO)

[118] Paillier, Pascal: Public-key cryptosystems based on composite degree residuosity classes. In: *International Conference on the Theory and Applications of Cryptographic Techniques*, 1999 (EUROCRYPT)

[119] Pandey, Omkant ; Rouselakis, Yannis: Property Preserving Symmetric Encryption. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2012 (EUROCRYPT)

[120] Pang, Hweehwa ; Ding, Xuhua: Privacy-Preserving Ad-Hoc Equi-Join on Outsourced Data. In: *ACM Transactions on Database Systems* (2014)

[121] Pinkas, Benny ; Schneider, Thomas ; Zohner, Michael: *Scalable Private Set Intersection Based on OT Extension*. IACR Cryptology ePrint Archive, Report 2016/930, 2016

[122] Popa, Raluca A. ; Li, Frank H. ; Zeldovich, Nickolai: An Ideal-Security Protocol for Order-Preserving Encoding. In: *Proceedings of the Symposium on Security and Privacy*, 2013 (S&P)

[123] Popa, Raluca A. ; Redfield, Catherine ; Zeldovich, Nickolai ; Balakrishnan, Hari: CryptDB: Protecting Confidentiality with Encrypted Query Processing. In: *Proceedings of the Symposium on Operating Systems Principles*, 2011 (OSP)

[124] Popa, Raluca A. ; Stark, Emily ; Helfer, Jonas ; Valdez, Steven ; Zeldovich, Nickolai ; Kaashoek, Frans ; Balakrishnan, Hari: Building Web Applications on Top of Encrypted Data Using Mylar. In: *Proceedings of the USENIX Symposium of Networked Systems Design and Implementation*, 2014 (NSDI)

[125] Popa, Raluca A. ; Zeldovich, Nickolai: Cryptographic treatment of CryptDB's adjustable join. (2012). `https://css.csail.mit.edu/cryptdb/`

[126] Ren, Ling ; Fletcher, Christopher W. ; Kwon, Albert ; Stefanov, Emil ; Shi, Elaine ; Van Dijk, Marten ; Devadas, Srinivas: Constants Count: Practical Improvements to Oblivious RAM. In: *USENIX Security Symposium*, 2015

[127] Rivest, Ronald L. ; Adleman, Len ; Dertouzos, Michael L.: On data banks and privacy homomorphisms. In: *Foundations of secure computation* 4 (1978), Nr. 11

[128] Roche, Daniel S. ; Apon, Daniel ; Choi, Seung G. ; Yerukhimovich, Arkady: POPE: Partial Order Preserving Encoding. In: *Proceedings of the Conference on Computer and Communications Security*, 2016 (CCS)

[129] Sahai, Amit ; Waters, Brent: Fuzzy identity-based encryption. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2005 (EUROCRYPT)

[130] Samarati, Pierangela ; Sweeney, Latanya: Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression / SRI International. 1998. – Forschungsbericht

[131] Sasy, Sajin ; Gorbunov, Sergey ; Fletcher, Christopher W.: ZeroTrace: Oblivious Memory Primitives from Intel SGX. In: *Proceedings of Symposium on Network and Distributed System Security*, 2017 (NDSS)

[132] Schuster, Felix ; Costa, Manuel ; Fournet, Cédric ; Gkantsidis, Christos ; Peinado, Marcus ; Mainar-Ruiz, Gloria ; Russinovich, Mark: VC3: Trustworthy Data Analytics in the Cloud Using SGX. In: *Proceedings of the Symposium on Security and Privacy*, 2015 (S&P)

[133] Shen, Emily ; Shi, Elaine ; Waters, Brent: Predicate Privacy in Encryption Systems. In: *Theory of Cryptography Conference*, 2009 (TCC)

[134] Shi, Elaine ; Bethencourt, John ; Chan, Hubert T.-H. ; Song, Dawn X. ; Perrig, Adrian: Multi-Dimensional Range Query over Encrypted Data. In: *Proceedings of the Symposium on Security and Privacy*, 2007 (S&P)

[135] Smart, Nigel P. ; Vercauteren, Frederik: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: *International Workshop on Public Key Cryptography*, 2010 (WPKC)

[136] Smith, Sean W.: Secure coprocessing applications and research issues. 1996. – Forschungsbericht

[137] Song, Dawn X. ; Wagner, David ; Perrig, Adrian: Practical techniques for searches on encrypted data. In: *Proceedings of the Symposium on Security and Privacy*, 2000 (S&P)

[138] Stefanov, Emil ; Dijk, Marten van ; Shi, Elaine ; Fletcher, Christopher ; Ren, Ling ; Yu, Xiangyao ; Devadas, Srinivas: Path ORAM: An Extremely Simple Oblivious RAM Protocol. In: *Proceedings of the Conference on Computer and Communications Security*, 2013 (CCS)

[139] Stefanov, Emil ; Papamanthou, Charalampos ; Shi, Elaine: Practical Dynamic Searchable Encryption with Small Leakage. In: *Proceedings of the Network and Distributed System Security Symposium*, 2014 (NDSS)

[140] Sweeney, Latanya: Simple Demographics Often Identify People Uniquely. In: *Health (San Francisco)* (2000)

[141] Tarjan, Robert E.: Amortized Computational Complexity. In: *SIAM Journal on Algebraic Discrete Methods* 6 (1985), Nr. 2

[142] Van Dijk, Marten ; Gentry, Craig ; Halevi, Shai ; Vaikuntanathan, Vinod: Fully homomorphic encryption over the integers. In: *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2010 (EUROCRYPT)

[143] Wang, Boyang ; Hou, Yantian ; Li, Ming ; Wang, Haitao ; Li, Hui: Maple: Scalable Multi-dimensional Range Search over Encrypted Cloud Data with Tree-based Index. In: *Proceedings of the Symposium on Information, Computer and Communications Security*, 2014 (ASIACCS)

[144] Wang, Peng ; Ravishankar, Chinya: Secure and Efficient Range Queries on Outsourced Databases Using R-trees. In: *Proceedings of the International Conference on Data Engineering*, 2013 (ICDE)

[145] Wang, Yujue ; Pang, HweeHwa: Probabilistic Public Key Encryption for Controlled Equijoin in Relational Databases. In: *The Computer Journal* (2016)

[146] Xiao, Yuan ; Zhang, Xiaokuan ; Zhang, Yinqian ; Teodorescu, Radu: One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation. In: *USENIX Security Symposium*, 2016

[147] Yao, Andrew C.: Protocols for Secure Computations. In: *Proceedings of the Annual Symposium on Foundations of Computer Science* (FOCS)

[148] Yao, Andrew C.: How to Generate and Exchange Secrets. In: *Proceedings of the Annual Symposium on Foundations of Computer Science*, 1986 (FOCS)

[149] Yung, Moti: From mental poker to core business: why and how to deploy secure computation protocols? In: *Proceedings of the Conference on Computer and Communications Security*, 2015 (CCS)

[150] Zhang, Yupeng ; Katz, Jonathan ; Papamanthou, Charalampos: All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption. In: *USENIX Security Symposium*, 2016

[151] Zipf, George K.: The psycho-biology of language. (1935)