Differentially Private Searchable Symmetric Encryption Scheme with Configurable Pattern Leakage

by

Zhiwei Shang

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science (Co-operative Program)

Waterloo, Ontario, Canada, 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Searchable symmetric encryption (SSE) allows a data owner to outsource its data to a cloud server while maintaining the ability to search over it. Most existing SSE schemes leak access-pattern leakage, and thus are vulnerable to attacks like the IKK attack. Oblivious RAM and PIR can be used to construct SSE schemes that fully hide access patterns. However, such schemes suffer from heavy communication overhead or computation overhead making them impractical. Chen et al. proposed an obfuscation mechanism to protect existing SSE schemes against access-pattern leakage. This mechanism can produce differentially private access patterns per keyword. However, it cannot hide whether or not the same keyword is being searched multiple times or, in other words, the search patterns, making this mechanism vulnerable to search-pattern attacks.

In this thesis, we propose a stronger security definition for differentially private searchable symmetric encryption schemes and present a real construction, DP-SSE, fulfilling it. On the one hand, DP-SSE is adaptively semantically secure and provides differential privacy for both keywords and documents implying search-pattern hiding and access-pattern hiding, respectively. On the other hand, DP-SSE has communication overhead as small as $O(\log \log n)$ and computation complexity of $O(n \cdot \log \log n)$ when querying relatively frequent keyword w. When assuming queries follow Zipfian distribution, the amortized communication overhead would be $O(\log n \cdot \log \log n)$. By replicating the IKK attack, we show that DP-SSE can actually hide access patterns and make it difficult to extract useful information from differentially private access-pattern leakage. Finally, we perform KMeans clustering, we were able to show that inferring search patterns from differentially private access-pattern leakage is difficult, namely search patterns are hidden.

Acknowledgements

I would like to thank all the little people who made this thesis possible.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Searchable symmetric encryption (SSE) allows a client to outsource its database to another party (server) in a private way, while enabling the database to be searched by the client without revealing the content of queries or documents. In a typical SSE scenario, the client first locally produces an encrypted version of its database under one key, together with an encrypted index with another key, to be outsourced to the server. Later, the client can issue queries to search the outsourced encrypted index, and retrieve matched encrypted results to be decrypted locally.

During the interaction, the server will learn which documents are accessed. This type of leakage is known as access-pattern leakage. Most existing SSE schemes [8, 29, 17, 16, 3, 15, 2] allow such leakage for performance considerations. However, recent studies [14, 4, 32] demonstrated that with some prior knowledge of the outsourced database, an honest-but-curious server is able to recover the underlying keywords of queries with high accuracy.

One solution to hide access patterns is oblivious RAM (ORAM). An ORAM algorithm allows a client to access remote documents without revealing which documents are accessed. Such property is achieved by requiring clients to continuously shuffle and re-encrypt data as they are accessed which involves high communication overhead. In particular, to obliviously access one of n documents, at least $O(\log n)$ documents need to be accessed [13]. As pointed by [25], in some cases, such overhead is larger than downloading the entire database.

Chen et al. [5] proposed an obfuscation framework to protect SSE schemes from access-pattern leakage. The framework obfuscates the index of a database in a differentially private way before applying any SSE scheme. Later, client and server engage in an SSE scheme as usual. Since the index used to build an SSE scheme is obfuscated, the server only learns obfuscated access patterns making it more difficult to extract useful information from such leakage. Although the framework has much smaller communication overhead compared with ORAM, it cannot hide *search patterns*, namely whether a keyword has been searched multiple times, due to the fact that the access pattern for each keyword is deterministic after outsourcing. Moreover, it might happen that a *true positive* document is never returned due to the obfuscation, i.e. the *permanent false negative problem*.

Motivated by the above differentially private obfuscation framework, we propose a differentially private searchable symmetric encryption scheme (DP-SSE) based on inner product predicate encryption to mitigate access-pattern and search-pattern leakage. In DP-SSE, both access-pattern leakage and search-pattern leakage are differentially private and the level of such leakage can be configured per query. In particular, during one query for keyword w, a document D will be returned with probability p if $w \in D$; otherwise, it can also be returned with probability q, where p, q are determined per query. In this way, DP-SSE can overcome the permanent false negative problem of [5], hide both access patterns and search patterns providing stronger privacy guarantees than in [5] while still maintaining lower communication overhead than in ORAM ($O(\log \log n)$ v.s. $O(\log n)$), where n is the number of documents.

The paper is structured as follows. In chapter 2 we review related work, before we introduce preliminaries in chapter 3. We define a stronger privacy definition for differentially private SSE in chapter 4 and describe scheme construction in chapter 5. We analyze the security, differential privacy, and complexity of the scheme in chapters 6, 7, 8, respectively. In chapter 9 we summarize the results of evaluation and present our conclusions in chapter 10.

Chapter 2

Related Work

2.1 Searchable symmetric encryption

SSE is an encryption scheme which allows search. It enables a data owner to outsource its encrypted database to an untrusted server while still preserving the search functionality. SSE was put forward by Song et al. [29] which suggested several practical constructions whose search complexity were linear in the size of database and secure under Chosen Plaintext Attack (CPA). Goh et al. [12] pointed out CPA was not adequate for SSE schemes. Formal notions of security and functionality for SSE were provided in [8], as well as the first constructions satisfying them with search complexity linear in the number of results (sublinear in the size of the database). From then on, a long line of research has investigated SSE with various security features, efficiency properties, and flexible functionalities [17, 16, 3, 26, 2, 15]. However, all the above SSE schemes reveal access patterns to gain efficiency, where access pattern means which documents are accessed and returned in each query. This access-pattern leakage is proved to have opened the door for powerful query recovery attacks [14, 32, 4, 22].

2.2 Query Recovery Attacks

In 2012, Islam, Kuzu and Kantarcioglu [14] demonstrated that when knowing some statistics about a database and the content of a small fraction of queries, a semi-honest server could recover the contents of all queries with more than 90% accuracy. This is the first powerful attack (known as IKK attack) utilizing access-pattern leakage. An improved IKK attack is provided in [4] if additional information, i.e. the size of query results, is given. An attack targeting dynamic SSE called file-injection attack appeared in [32]. The authors of [32] showed that an active adversary could inject only a small number of carefully designed files in order to recover the content of queries by observing the access patterns of the injected files. Liu et al. [22] introduced another attack utilizing search-pattern leakage, which could be revealed through access patterns or query tokens. They showed that an adversary who had some prior knowledge about the client's search habits could uncover the underlying keywords of the client's queries.

2.3 Oblivious RAM

ORAM, first introduced by Goldreich and Ostrovsky [13], was designed to hide memory access patterns of CPU. ORAM is formalized in a game between CPU and memory where the CPU holds a sequence of memory block locations, say $\{l_1, l_2, ..., l_m\}$, to be accessed, and the memory stores the database, say $\{d_1, d_2, ..., d_n\}$. In this game, the CPU wants to blind memory about which (true) blocks have been accessed for how many times. The authors of [13] showed that a client could hide entirely the access patterns by continuous shuffling and re-encrypting data as they accessed with a $poly(\log n)$ communication overhead and $O(\log n)$ client-side storage. Since its proposal, there has been a fruitful line of research on further reducing this overhead [30, 1, 24, 31, 9]. Path ORAM [30], popular for its simplicity and efficiency, achieved $O(\log n)$ communication overhead with $O(\log n)$ client storage but required block size to be $\Omega(\log n)$. Apon et al. [1] showed that one can construct an ORAM scheme with constant communication overhead by fully homomorphic encryption. Inspired by the idea of utilizing homomorphic encryption(HE) in ORAM, Circuit ORAM [31], Onion ORAM [9], and optimized Onion ORAM [24] were proposed to further reduce the communication overhead to a smaller constant. However, those constants are still too large (say 100) to be practical with a reasonably small block size. On the other hand, homomorphic encryption operations induce very expensive computation which makes them far from being practical. It is still open whether or not a non-HE-based ORAM construction can break the communication overhead lower bound of $O(\log n)$ given by [13, 21]. Garg et al. [11] proposed an ORAM-based SSE scheme to hide access pattern with a communication overhead $O(\log n \cdot \log\log n)$. However, this $O(\log n \cdot \log\log n)$ communication overhead seems to be too large, since it might lead to, when searching for frequent keywords, a communication volume larger than directly downloading the entire database [25].

2.4 Private Information Retrieval

PIR is used to hide access patterns to public databases. It enables a client holding an index i to retrieve the i^{th} item d_i from a server holding a public database $\{d_1, d_2, ..., d_n\}$ without revealing the index i to the server. PIR was put forward by Chor et al. [6] in 1995 in the setting where there are many non-cooperating copies of the same databases. They also showed that single-database PIR, in the information-theoretic sense, does not exist. Later in 1997, Kushilevitz and Ostrovsky [20] demonstrated a method to construct a computationally secure single-database PIR based on *Quadratic Residuosity Problem* assuming a computationally bounded server. Unlike in ORAM, single-database PIR may induce a lower communication overhead. However, it naturally requires to touch every bit in the database per access in order to hide access patterns, namely a $O(n)$ computation overhead. Therefore, PIR either requires multiple non-cooperating servers to work or have a computation overhead linear in size of the database. Moreover, PIR considers a different threat model where the content of the database is public.

2.5 Differentially Private Access Patterns

Chen et al.[5] proposed to use a differentially private obfuscation framework to mitigate access-pattern leakage. The framework is compatible with existing SSE schemes with only a slight change when building the index. Instead of building an SSE upon the true index $IND = \{IND[i,j], i \in [|\Delta|], j \in [n]\}$ (where $IND[i,j] \in \{0,1\}, [|\Delta|] = [1,2,...,|\Delta|]$ similar to $[n]$, and $|\Delta|$ and n are the number of keywords and documents, respectively) of a database $\mathcal{D}$, the mechanism first obfuscates IND to IND' in a way that

$$IND'[i,j] = \begin{cases} \textit{if } IND[i,j] = 1 \begin{cases} \text{1 with probability p;} \\ \text{0 otherwise;} \end{cases} \\ \textit{if } IND[i,j] = 0 \begin{cases} \text{1 with probability q;} \\ \text{0 with probability 1 - q.} \end{cases} \end{cases}$$

As a consequence of this obfuscation, the server only learns the obfuscated version of the access patterns from which it is much harder to derive useful information. However, the obfuscation is fixed after outsourcing. If a client searches for a keyword w multiple times, the same obfuscated access pattern repeats. This repetition actually reveals search patterns which can be used to perform a query recovery attack with high accuracy [22]. Chen et al. suggested to use the *grouping-based construction*(GBC) proposed in [22] to mitigate such attack. In a high level, when a client wants to search for a keyword w, GBC suggested to search for addtional t keywords, say $\{w_1, w_2, ..., w_t\}$ besides w in order to make the query frequency of keywords uniform. However, GBC itself suffers from the following drawbacks:

- The communication overhead grows linearly with the privacy level of GBC. With GBC, a client needs to search for t redundant keywords per query leading to a $O(t)$ communication overhead. On the other hand, t defines the privacy level of GBC, the larger the better since what GBC can provide is a $(t+1)$-anonymity notion. [22] empirically suggested t to be larger than 2.
- GBC leaks group search patterns. Although the server cannot tell which exact keyword is queried due to GBC redundancy, it can tell that the keyword must be in the

set $\{w, w_1, ..., w_t\}$. As pointed in the same paper as GBC, it is not guaranteed that leaking group search patterns is secure enough. One example is, if redundancy group $\{w_{i,0}, w_{i,1}, ..., w_{i,t}\}$ is leaked to server, then the next time for searching $w_{i,j}, i \in [t]$, the server has a better chance to guess the underlying keyword, which is $1/(t+1)$. With some prior knowledge, the chance of a successful guess could be higher.

Chapter 3

Preliminaries

Let $\Delta = \{w_{(1)}, w_{(2)}, ..., w_{(|\Delta|)}\}$ be the keyword universe of size $|\Delta|$, and 2^{Δ} be the power set of Δ which denotes the set of all possible documents. Let $id(D)$ be the identifier of document D, $\mathcal{D} \subset 2^{\Delta}$ be an ordered (based on their *id*s) list of n documents $\mathcal{D} = \big[\mathcal{D}[1], \mathcal{D}[2], ..., \mathcal{D}[n]\big]$, and $2^{2^{\Delta}}$ be the set of all possible document collections. Let $id(\mathcal{D})$ be an ordered list of identifiers of all documents in $\mathcal{D}$. Without loss of generality, we set $id(\mathcal{D}[i]) = i$. Let $\mathcal{D}(w)$ denote the ordered (based on their *id*s) list of all documents containing keyword w.

3.1 Searchable Symmetric Encryption

Definition 3.1.1. (Index). The index *IND* of $\mathcal{D}$ over Δ is a binary matrix of size $|\Delta| \times n$ such that

$$\mathit{IND}[i,j] = \begin{cases} 1 \text{ if } w_i \in \mathcal{D}[j]; \\ 0 \text{ otherwise.} \end{cases}$$

Definition 3.1.2. (Searchable Symmetric Encryption Scheme) (SSE) An SSE scheme is a collection of four polynomial-time algorithms (Keygen, BuildIndex,Trapdoor, Search) such that:

- $\mathsf{Keygen}(1^\lambda)$ is a probabilistic key generation algorithm that is run by the client to setup the scheme. It takes a security parameter λ and returns a secret key sk such that the length of sk is polynomially bounded in λ.

- $\mathsf{BuildIndex}(sk, \mathcal{D})$ is a (possibly probabilistic) algorithm run by the client to generate an index. It takes a secret key sk and document collection $\mathcal{D}$ as inputs, and returns an encrypted index I such that the length of I is polynomially bounded in λ.

- $\mathsf{Trapdoor}(sk, w)$ is run by the client to generate a trapdoor τ_w for a given keyword w. It takes a secret key sk and a word w as inputs, and returns a trapdoor τ_w.

- $\mathsf{Search}(I,\tau_w)$ is run by the server in order to search for $\mathcal{D}(w)$. It takes an encrypted index I for a collection $\mathcal{D}$ and a trapdoor τ_w for keyword w as inputs, and returns $\mathrm{id}(\mathcal{D}(w))$, the set of identifiers of documents containing w.

We borrow definitions for *history, view* and *trace* from [8] to ease the later algorithm description and security analysis.

Definition 3.1.3. (History). A t-query history over $\mathcal{D}$ is a tuple $H_t = (\mathcal{D}, \vec{w})$ (sometimes denoted by $H_{\vec{w}}$) where $\vec{w} = [\vec{w}[1], \vec{w}[2], ..., \vec{w}[t]]$ is the vector of underlying keywords of the t queries.

An initiation of such a history is called an *interaction*. A partial history of H_t, say H_t^s, is a tuple $(\mathcal{D}, \vec{w}')$ where $\vec{w}' = [\vec{w}[1], \vec{w}[2], ..., \vec{w}[s]]$ and $0 \leq s \leq t$. A history models the sensitive information that the client wants to keep private. During an interaction of a history H, what the server can "see" is the *view*.

Definition 3.1.4. (View). A view of a history H_t under a secret key sk is defined as $V_{sk}(H_t) = (id(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), , ..., \mathcal{E}(\mathcal{D}[n]), I, \tau_1, ..., \tau_t)$ where τ_i is the query token for the i^{th} query. The partial view $V_{sk}^s(H_t)$ of a history H_t under secret key sk is the tuple $V_{sk}^s(H_t) = (id(\mathcal{D}), \mathcal{E}(\mathcal{D}[1]), ..., \mathcal{E}(\mathcal{D}[n]), I, \tau_1, ..., \tau_s)$.

A view includes the encrypted search index of $\mathcal{D}$, search tokens as well as some additional common information, like the number of documents and the size of encrypted documents. It should be noted that a partial view also includes the entire encrypted search index I. However, a view should not reveal any sensitive information about a history. This leads to the notion of *trace* of a history. The trace models the leakage allowed by an SSE scheme. We will omit the subscript sk when sk does not need to be addressed. Before defining trace, we first define formally the access pattern and the search pattern.

Definition 3.1.5. (Access Pattern) The access pattern $\Pi_{\vec{w}}$ over an t-query history $H_t = (\mathcal{D}, \vec{w})$ is a binary matrix of size $t \times n$ such that

$$\Pi_{\vec{w}}[i,j] = \begin{cases} 1 \text{ if } \vec{w}[i] \in \mathcal{D}[j]; \\ 0 \text{ otherwise.} \end{cases}$$

Definition 3.1.6. (Search Pattern) The search pattern $\Phi_{\vec{w}}$ over a t-query history $H_t = (\mathcal{D}, \vec{w})$ is a symmetric binary matrix of size $t \times t$ such that

$$\Phi_{\vec{w}}[i,j] = \begin{cases} 1 \text{ if } \vec{w}[i] = \vec{w}[j]; \\ 0 \text{ otherwise.} \end{cases}$$

Definition 3.1.7. (Trace). The trace of $H_t = (\mathcal{D}, \vec{w})$ is the sequence $T(H_t) = (id(\mathcal{D})$, $|\mathcal{D}[1]|$, ..., $|\mathcal{D}[n]|$, $\Pi_{\vec{w}}$, $\Phi_{\vec{w}})$ where $\Pi_{\vec{w}}$ and $\Phi_{\vec{w}}$ are the access patterns and search patterns of history H_t , respectively.

It should be noted that the search patterns $\Phi_{\vec{w}}$ can be revealed in two possible ways. If the query token is deterministic for each keyword, then $(\tau_1, ..., \tau_t)$ directly reveals $\Phi_{\vec{w}}$. $\Pi_{\vec{w}}$ can also reveal $\Phi_{\vec{w}}$ since two identical access patterns of two queries are expected to have the same underlying keyword.

Definition 3.1.8. (Adaptive Semantic Security for SSE[8]) An SSE scheme is adaptively semantically secure if for all $t \in \mathbb{N}$ and for all (non-uniform) probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a (non-uniform) probabilistic polynomial-time algorithm (the simulator) $\mathcal{S}$ such that for all traces T_t of length t, all polynomially samplable distributions $\mathcal{H}_t$ over $\{H_t : T(H_t) = T_t\}$, i.e. the set of histories with trace T_t, all functions $f : \{0,1\}^{|H_t|} \rightarrow \{0,1\}^{poly(|H_t|)}$, all $0 \leq s \leq t$ and all polynomials P and sufficiently large k:

$$\left| Pr[\mathcal{A}(V^s_{sk}(H_t)) = f(H^s_t)] - Pr[\mathcal{S}(T(H^s_t)) = f(H^s_t)] \right| < \frac{1}{P(k)}$$

where $H_t \leftarrow \mathcal{H}_t, sk \leftarrow \mathsf{Keygen}(1^k)$, and probabilities are taken over $\mathcal{H}_t$ and the internal coins of $\mathcal{A}, \mathcal{S}$ and the underlying Keygen, BuildIndex, Trapdoor, Search algorithms.

A proof for the equivalence of adaptive semantic security and adaptive indistinguishability for SSE is provided in the same paper [8]. We will show, in later chapter, our SSE construction provides adaptively semantic security.

3.2 Cryptographic Tools

Let Σ denote a finite set of plaintexts, and let $\mathcal{F}$ denote a finite set of predicates $f : \Sigma \rightarrow \{0, 1\}$. We say that $x \in \Sigma$ satisfies a predicate f if $f(x) = 1$.

Definition 3.2.1. (Symmetric-Key Inner Product Predicate Encryption) A symmetric-key Inner Product predicate encryption scheme for the class of predicates $\mathcal{F}$ over the set of attributes Σ consists of the following probabilistic polynomial-time algorithms.

- $\mathsf{Setup}(1^\lambda)$ takes as input a security parameter 1^λ and outputs a secret key skp.
- $\mathsf{Encrypt}(skp, x)$ takes as input a secret key skp and a plaintext $x \in \Sigma$ and outputs a ciphertext ct_x.
- $\mathsf{GenToken}(skp, f)$ takes as input a secret key skp and a description of a predicate $f \in \mathcal{F}$ and outputs a search token st_f.
- $\mathsf{Query}(st_f, ct_x)$ takes as input a token st_f for a predicate f and a ciphertext ct_x for plaintext x. It outputs either 0 or 1, indicating the value of $f(x)$.

Correctness. For correctness, we require the following conditions. For all λ, all $x \in \Sigma$, and all $f \in \mathcal{F}$, letting $skp \leftarrow \mathsf{Setup}(1^\lambda)$, $st_f \leftarrow \mathsf{GenToken}(skp, st_f)$, and $ct_x \leftarrow \mathsf{Encrypt}(skp, x)$,

1. If $\langle x, f \rangle = 0$, then $\mathsf{Query}(st_f, ct_x) = 1$.
2. If $\langle x, f \rangle \neq 0$, then $\Pr[\mathsf{Query}(st_f, ct_x) = 0] > 1 - \delta(\lambda)$ where δ is a negligible function.

Inner product predicate encryption scheme can be used to build predicate encryption schemes for classes of predicates corresponding to polynomial evaluation [18]. In general, a symmetric-key inner-product predicate encryption (IPPE) scheme of dimension $d+1$ can be used to construct a predicate encryption scheme for the family of polynomials of degress at most d, i.e. $\Psi_{\leq d}^{poly} = \{f_P, |P \in \mathbb{Z}_N[x], deg(P) \leq d\}$ where:

$$f_P(x) = \begin{cases} 1 & \text{if } P(x) = 0 \text{ mod N} \\ 0 & \text{otherwise.} \end{cases}$$

as follows:
Given a polynomial of degree d, i.e. $P(x) = \sum_{i=0}^{d} a_i \cdot x^i = \langle \vec{\alpha}, \vec{\beta} \rangle$ where $\vec{\alpha} = (a_0, a_1, ..., a_d)$, $\vec{\beta} = (x^0, x^1, ..., x^d)$.

- The IPPE.Setup algorithm keeps unchanged.
- To encrypt an attribute $\vec{\alpha} \in \mathbb{Z}_N^{d+1}$, the ciphertext will be $ct_{\vec{\alpha}} \leftarrow$ IPPE.Encrypt$(skp, \vec{\alpha})$.
- To generate a token corresponding to a variable x, the token will be $st_\beta \leftarrow$ IPPE.GenToken $(skp,\ \vec{\beta})$.

Then, whenever a variable x satisfies a polynomial(P(x) = 0), the token generated from it will also satisfy the encrypted attribute corresponding to the same polynomial.

Full Security of a symmetric-key inner product predicate encryption scheme is defined by the following game G between an adversary $\mathcal{A}$ and a challenger holding IPPE.

Setup: The challenger runs IPPE.Setup(1^λ) and keep sk to itself. The challenger picks a random bit b.

Queries: $\mathcal{A}$ adaptively issues queries, where each query is one of two types:

- Ciphertext query. On the j^{th} ciphertext query, $\mathcal{A}$ outputs two plaintexts $x_{j,0}, x_{j,1} \in \Sigma$. The challenger responds with IPPE.Encrypt$(sk, x_{j,b})$
- Token query. On the i^{th} token query $\mathcal{A}$ outputs descriptions of two predicates $f_{i,0}, f_{i,1} \in \mathcal{F}$. The challenger responds with IPPE.GenToken$(sk, f_{i,b})$.
  $\mathcal{A}$'s queries are subject to the restriction that, for all ciphertext queries $(x_{j,0}, x_{j,1})$ and all predicate queries $(f_{i,0}, f_{i,1})$, $f_{i,0}(x_{j,0}) = f_{i,1}(x_{j,1})$.

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

The advantage of $\mathcal{A}$ is defined as $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$.

Definition 3.2.2. (Full Security for IPPE). A symmetric-key inner product predicate encryption scheme is fully secure if, for any probabilistic polynomial adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in λ.

Roughly speaking, full security guarantees that given a set of tokens of predicates $f_1, ..., f_k$ and a set of encryptions of plaintexts $x_1, ..., x_t$, no adversary can gain any information about any predicate or any plaintext other than the value of each predicate evaluated on each of the plaintexts. The notion of predicate privacy is inherently impossible in the public-key setting, which is the reason why our idea of construction works only in the private-key setting.

There is another stronger security notion in the context of IPPE which is simulation-based security (SIM-security). SIM-security requires that every efficient adversary $\mathcal{A}$ that interacts with the real IPPE can be simulated given only oracle access to the inner products between each pair of vectors that $\mathcal{A}$ submits to the real IPPE. It should be noted that SIM-security implies full security.

Definition 3.2.3. (SIM-Security for IPPE). Let IPPE = ($\mathsf{IPPE.Setup}$, $\mathsf{IPPE.Encrypt}$, $\mathsf{IPPE.GenToken}$, $\mathsf{IPPE.Query}$) be an inner product predicate encryption scheme. Then IPPE is SIM-secure if IPPE is fully secure and for any efficient $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ such that the following two games are computationally indistinguishable:

$\mathsf{Real}_{\mathcal{A}}(1^\lambda)$:	$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$:
1. $sk \leftarrow \mathsf{IPPE.Setup}(1^\lambda)$	1. $sk' \leftarrow \mathcal{S}.\mathsf{Setup}(1^\lambda)$
2. $b' \leftarrow G_{\mathcal{A},\mathsf{IPPE}}(1^\lambda)$	2. $b'' \leftarrow G_{\mathcal{A},\mathcal{S}}(1^\lambda)$
3. output b'.	3. output b''.

where game G represents the game defined in full security, $G_{\mathcal{A},\mathsf{IPPE}}$ is such a game between $\mathcal{A}$ and IPPE, and $G_{\mathcal{A},\mathcal{S}}$ is such a game between $\mathcal{A}$ and $\mathcal{S}$.

Chapter 4

Definitions

As mentioned in Chapter 1, leaking access patterns or search patterns in SSE might lead to leaking sensitive information, while hiding them entirely like in ORAM or PIR has large computation or communication overhead (or both). Motivated by this, we explore a middle-ground solution that consists in hiding access patterns and search patterns in SSE in a differentially private way. There are two possible ways to define a differentially private SSE: one is based on differential privacy for keywords (similar to [5]) and the other one is based on differential privacy for documents that we define later. In the following parts of this chapter, we will show implications of the two definitions, demonstrate the relation between them, and finally give our definition of a differentially private SSE.

4.1 Differential Privacy For Keywords

Definition 4.1.1. (Differential Privacy for Keywords) A searchable encryption scheme $\mathcal{SE} : (2^{2^{\Delta}}, \Delta^{|\vec{w}|}) \rightarrow T(H_{\vec{w}})$ gives ϵ-differential privacy for *keywords*, iff for any database $\mathcal{D} \in 2^{2^{\Delta}}$ and for any pair of neighboring keyword lists $\vec{w}, \vec{w}' \in \Delta^{|\vec{w}|}$, namely $\vec{w}$ and $\vec{w}'$ differ in only one element, say $\vec{w}[i] \neq \vec{w}'[i]$,

$$Pr[\mathcal{SE}(\mathcal{D}, \vec{w}) \in S] \leq e^{d \times \epsilon} Pr[\mathcal{SE}(\mathcal{D}, \vec{w}') \in S]$$

where $S \subset \mathcal{T}(\mathcal{SE})$, $\mathcal{T}(\mathcal{SE})$ represents the set of possible traces of $\mathcal{SE}$, and d is the number of different documents between $\mathcal{D}(\vec{w}[i])$ and $\mathcal{D}(\vec{w}'[i])$.

Intuitively, this definition implies hiding search patterns since it guarantees that no one can determine, through observing the trace (allowed leakage) of the SE, whether a client is searching one keyword list or the other. However, there is no implication for hiding access patterns in the above definition. Based on the definition of access pattern, hiding it means no one can determine whether a document contains a keyword or not. This leads to our definition of differential privacy for documents.

4.2 Differential Privacy For Documents

Definition 4.2.1. (Differential Privacy for Documents) A searchable encryption scheme $\mathcal{SE}(2^{2^{\Delta}}, \Delta^{|\vec{w}|}) \rightarrow T(H_{\vec{w}})$ gives ϵ-differential privacy for *documents*, iff for any keyword list $\vec{w} \in \Delta^{|\vec{w}|}$ and for any pair of neighboring databases $\mathcal{D}, \mathcal{D}' \in 2^{2^{\Delta}}$, namely there exists only one position i and exactly one keyword w, such that w is in either $\mathcal{D}[i]$ or $\mathcal{D}'[i]$ but not both,

$$Pr[\mathcal{SE}(\mathcal{D}, \vec{w}) \in S] \leq e^{|\vec{w}| \times \epsilon} Pr[\mathcal{SE}(\mathcal{D}', \vec{w}) \in S]$$

where $S \subset \mathcal{T}(\mathcal{SE})$ and $\mathcal{T}(\mathcal{SE})$ represents the set of possible traces of $\mathcal{SE}$.

Intuitively, satisfying the above definition gives the guarantee that no one can determine whether a document contains a keyword or not based only on the trace of the SE. Recalling the definition of access pattern which is just the reflection of which document contains which keyword, an SE providing differential privacy for documents automatically hide access patterns. However, the above definition is not directly related to hiding search patterns.

4.3 Relations

By intuition, differential privacy for keywords implies search-pattern hiding while differential privacy for documents implies access-pattern hiding. If one can imply the other, we can just focus on one notion. However, neither of the above two definitions can imply the other one.

Theorem 4.3.1. Neither differential privacy for documents nor differential privacy for keywords imply the other one.

Proof. Refer to appendix A.1. □

4.4 Utility

An SE scheme with high utility should satisfy the following two requirements when searching keyword w:

1. Almost all documents in $\mathcal{D}(w)$ are returned. In other words, the true positive rate is high.
2. Not many documents in $\mathcal{D} \setminus \mathcal{D}(w)$ are returned. In other words, the false positive rate is low.

Based on this, we define the utility of an SE scheme as follows.

Definition 4.4.1. (**Utility**) The utility of an SE scheme is $p(1-q)$, where p and q are true positive rate and false positive rate, respectively.

It is easy to see that, when searching for w, an SE which always returns $\mathcal{D}(w)$ has utility of 1, while an SE which always returns the entire database $\mathcal{D}$ has utility $\dfrac{|\mathcal{D}(w)|}{|\mathcal{D}|} \approx 0$.

4.5 Differentially Private SSE

We define $(u, \epsilon_1, \epsilon_2)$-differentially private SSE as follows.

Definition 4.5.1. ((u, ϵ_1, ϵ_2)-Differentially Private Searchable Symmetric Encryption Scheme) A $(u, \epsilon_1, \epsilon_2)$-differentially private SSE is an SSE of utility u which provides ϵ_1-differential privacy for keywords and ϵ_2-differential privacy for documents.

Chapter 5

Algorithm Description

In this chapter, we first describe in a very high level how our proposed scheme operates, then we construct a naive SSE scheme (denoted by SSE-Naive) by function-hiding inner product predicate encryption scheme. Later we modify SSE-Naive to construct our differentially private SSE denoted by DP-SSE, and finally we introduce 3 variants of DP-SSE: denoted by DP-SSE-1, DP-SSE-2 and DP-SSE-3. Those 3 variants provide different performance and privacy guarantees. For reference, table 5.1 contains the main notation used in this section.

5.1 Construction Overview

We construct an SSE scheme from a functional encryption scheme, in particular a function-hiding inner product predicate encryption (FHIPPE) scheme. An FHIPPE is an IPPE scheme which provides SIM-security. FHIPPE is first probabilistic (encrypting the same message several times usually yields different ciphertexts) which prevents inferring search patterns from search tokens. Second, FHIPPE's function-hiding property prevents additional leakage of search process besides access patterns. To hide real access patterns, we dedicate a search token generation function to only leak a differentially private version of real access patterns. The function also makes it hard to infer search patterns from access

Notation	Description
D	a document.
$id(D)$	an identifier of a document D.
C_{max}	the maximum keyword frequency. Note that a document can contribute at most 1 in one keyword's.
C_w	frequency of keyword w.
S_{max}	the maximum number of distinct keywords that a single document can have.
Δ	the keyword universe of the entire database.
$\|\cdot\|$	the size of $\cdot$.
$a\|\|b$	a concatenates b. a and b are 0-padded to fixed length.
γ	an integer distinct from any keyword in Δ.
k_{-1}	an integer distinct from any keyword in Δ and γ.
$[t]$	$\{1, 2, 3, ..., t\}$.
n	number of documents.
$\mathcal{D}$	a list of all documents ordered by their ids.
$\mathcal{D}(w)$	a list of all documents that contain w ordered by their ids.
Φ	search pattern.
Π	access pattern.
H	history.
V	view.
T	trace.
Γ	search token list.
τ	a search token.

Table 5.1: Notations

patterns. The reason for this is that the access-pattern leakage is obfuscated in a differentially private way per query, namely the access patterns for two queries searching for the same keyword are expected to be different with high probability. To summarize, besides leakage from initialization our scheme would only leak a differentially private version of access patterns.

To better understand such access-pattern leakage, we visualize it in Figure 5.1. The real access pattern of a search for w, denoted by $\hat{\Pi}_w$, is a binary vector of length n, i.e. $(a_1, a_2, ..., a_n)$ where $a_i \in \{0, 1\}$ representing whether $\mathcal{D}[i]$ is matched $(a_i = 1)$ or not $(a_i = 0)$. In Figure 5.1, when searching for w, $a_1 = 0$ and $a_2 = 1$ mean that $\mathcal{D}[1]$ is not matched while $\mathcal{D}[2]$ is matched. The access pattern leaked in our scheme while searching for w, denoted by Π_w, is no longer a binary vector of length n but a multinary vector of length $n + |h|$, i.e. $(b_1, b_2, ..., b_n, b_{n+1}, b_{n+|h|})$ where $b_i \in \{0, 1, 2, ...\}$, h is a label distribution function which will be discussed later and $|h|$ represents the size of the range of h which is quite smaller than n. For $i \leq n$, b_i represents the times that $\mathcal{D}[i]$ is matched. The reason why b_i is multinary is because multiple tokens would be issued per query and one document might be matched by more than one token. For $i \leq |h|$, b_{n+i} represents the number of non-match tokens that have label i. The reason for these additional dimensions is because there are tokens that match no documents. Here, h is a function that assigns a label to each document. Each query token will also have a label assigned by query generation function to ensure that a query token can only match documents with the same label. It should be noted that the labels of documents and tokens are known to the server who utilizes them to reduce computation complexity as will be discussed in 8.2.

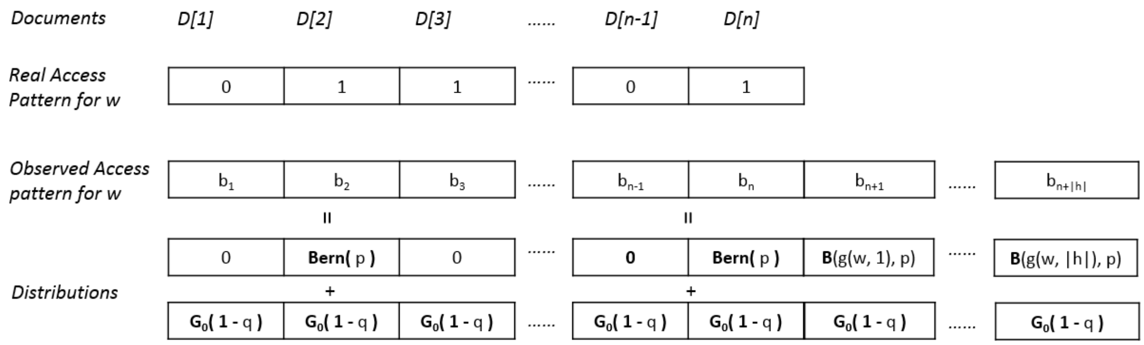


Figure 5.1: Access Pattern Leakage

The differential privacy guarantee for such access-pattern leakage comes from the fact that each b_i satisfies some special distribution. In particular, if $a_i = 1$, then b_i can be viewed as sum of two random variables u_i and v_i following a Bernoulli distribution and a geometric distribution, respectively, i.e. $u_i \sim \mathbf{Bern}(p)$ and $v_i \sim \mathbf{G}_0(1-q)$ where p and q are two probabilities; while if $a_i = 0$, then b_i follows geometric distribution, i.e. $b_i \sim \mathbf{G}_0(1-q)$. Similarly, for $i \leq |h|$, b_{n+i} can be viewed as sum of two random variables, i.e. $b_{n+i} = u_{n+i} + v_{n+i}$ where u_{n+i} and v_{n+i} follow a binomial distribution and a geometric distribution, respectively, i.e. $u_{n+i} \sim \mathbf{B}(g(w,i),p)$ and $v_{n+i} \sim \mathbf{G}_0(1-q)$. Here, $g(\cdot,i)$ is a database-specific function which will be discussed later. As we will see in 5.3.2, g has a close relation to h. It should be noted that all the above random variables u_i and v_i are independent. Also, for $i \leq |h|+n, j \in \mathbb{N}$, $Pr[b_i = j] > 0$, in other words, the access pattern for any keyword could be any value in $\mathbb{N}^{n+|h|}$ no matter what the content of each document is. As a consequence, we can make the following claim.

Claim 5.1.1. The access-pattern leakage of our scheme is differentially private.

5.2 SSE-Naive: A Naive SSE Scheme from FHIPPE

In this section, we introduce a naive SSE scheme that use a function-hiding inner product predicate encryption scheme (FHIPPE). For this naive SSE scheme (denoted by SSE-Naive), we use traditional symmetric encryption denoted by $\mathcal{E}$ to encrypt the content of each document D, and append to the resulting ciphertext denoted by $\mathcal{E}(D)$ a *search key* generated by an FHIPPE. In particular, SSE-Naive has the following four polynomial-time algorithms (Keygen, BuildIndex, Trapdoor, Search) such that:

- Keygen(1^λ): takes as input a security parameter 1^λ and returns FHIPPE.Setup(1^λ), which outputs a secret key sk.
- BuildIndex$(sk, \mathcal{D})$: takse as input the secret key sk and the document collection $\mathcal{D}$. For every document $\mathcal{D}[i]$, it calls FHIPPE.Encrypt$(sk, \mathsf{genVector}(\mathcal{D}[i], i))$ where genVector is a function that takes as input a document's content and its id i, and outputs a *vector* $v_i \in \Sigma$. The output of FHIPPE.Encrypt(sk, v_i) denoted by $I[i]$ will be appended to $\mathcal{E}(\mathcal{D}[i])$ as *search key*. We call the collection of all *search keys* the *search index* of $\mathcal{D}$ denoted by I which is the output of the function BuildIndex.

- Trapdoor$(sk,\ w)$: takes as input the secret key sk and a keyword w, and calls genPredicate(w) to get a predicate set $F_w \subset \mathcal{F}$. For every predicate $f \in F_w$, this function calls FHIPPE.GenToken(sk, f) and outputs a query token τ_f. Therefore, the output of Trapdoor(sk, w) is a set of query tokens Γ_{F_w}.

- Search(I, Γ_{F_w}): takes as input the search index I and query token set Γ_{F_w}, then for every search key $I[i] \in I$ and every query token $\tau_f \in \Gamma_{F_w}$, it calls FHIPPE.Query$(I[i], \tau_f)$ and, if the output is 0, it returns the corresponding document identifier i. Therefore, the output of Search(I, Γ_{F_w}) is a set of document identifiers $id(\bar{\mathcal{D}}(w))$. Note that those identifiers are readable to the server, namely the server can respond to the client with the corresponding encrypted documents $\{\mathcal{E}(\mathcal{D}[i]) \mid i \in id(\bar{\mathcal{D}}(w))\}$ in one round.

The function Keygen, BuildIndex, and Trapdoor are executed in the client side and the output of BuildIndex denoted by I will be outsourced to the server together with the encrypted documents $\mathcal{E}(\mathcal{D})$. The function Search is executed in server side during keyword search.

SSE-Naive achieve a 1 true positive rate and an approximate 0 false positive rate, so its utility is approximate 1 according to Definition 4.4.1.

5.2.1 genVector and genPredicate

Function genVector takes as input a document $\mathcal{D}[i] = \{w_1, ..., w_{|\mathcal{D}[i]|}\}$[1] and its document id i, and outputs an attribute vector v_i used later for encryption. Basically, the attribute vector v_i should be satisfied by any token generated from $w_j \in \mathcal{D}[i]$. A straightforward idea is first derive a polynomial $P(x)$ such that for every $w_j \in \mathcal{D}[i]$, $P(w_j) = 0$, and then obtain the attribute vector from the coefficients of $P(x)$. The following polynomial satisfies such requirements:

$$P(x) = \prod_{j=1}^{|\mathcal{D}[i]|} (x - w_j) = \sum_{j=0}^{|\mathcal{D}[i]|} a_j \cdot x^j$$

[1] We abuse the notation $\mathcal{D}[i]$ to represent both the document $\mathcal{D}[i]$ and its keyword list.

Due to the fact that FHIPPE scheme requires every attribute (predicate) to be of the same length, the polynomial should have the ability to encode the largest document. Thus, the following polynomial is actually used:

$$P(x) = \prod_{j=1}^{|\mathcal{D}[i]|} (x - w_j) \cdot \prod_{j=|\mathcal{D}[i]|+1}^{S_{max}} (x - \gamma) = \sum_{j=0}^{S_{max}} a_j \cdot x^j$$

where $w_j \in \mathcal{D}[i]$ for $j \in \left[|\mathcal{D}[i]|\right]$, and $\gamma \notin \Delta$. (Note that all terms with γ are acting as placeholders, and γ will never be used to generate predicates.)

Directly, we can obtain $v_i = (a_0, a_1, ..., a_{S_{max}})$.

The token generation function is much more straight forward given the polynomial $P(x)$. Let f_i be the output of $\mathsf{genPredicate}(w_i)$, then we can simply assign:

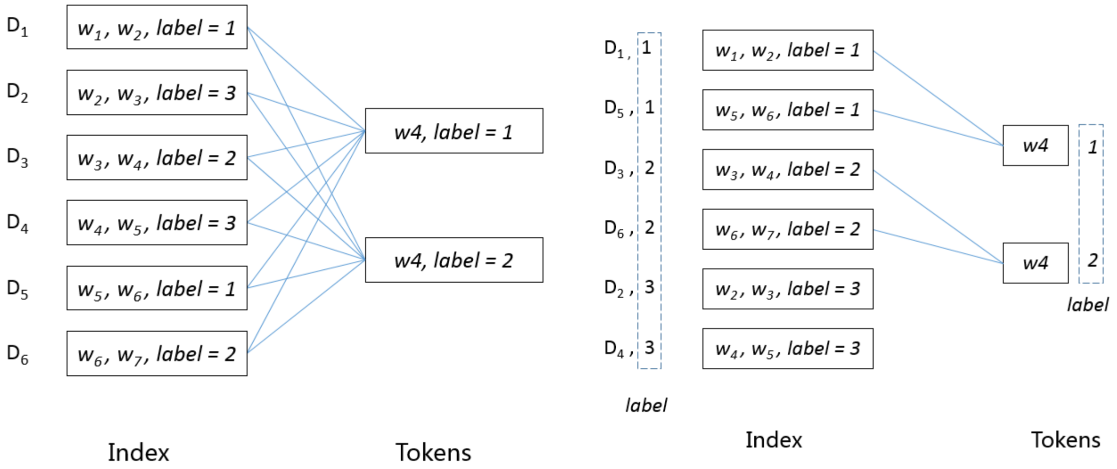
$$f_i = (w_i^0, w_i^1, ..., w_i^{S_{max}})$$

Now, genVector and genPredicate are both well-defined. It can be easily verified that the above implementations of genVector and genPredicate satisfy the correctness requirements and give utility of 1.

5.3 DP-SSE: A Differentially Private SSE from FHIPPE

In this section, we detail our construction of differentially private SSE (denoted by DP-SSE) which achieves what has been described in 5.1 by modifying SSE-Naive. Specifically, DP-SSE is a $(u, \epsilon_1^h, \epsilon_1^h)$-differentially private SSE where $u = (p + q - pq)(1 - p)$ and the values of ϵ_1^h and ϵ_2^h depend on how h is constructed. For example, if h is chosen as a hash function of range $[C_{max}]$, then $\epsilon_1^h = \epsilon_2^h = \ln(1 + p/(q(1 - p)))$. DP-SSE can be viewed as the *meta*-scheme of all the other 3 variants. It captures (almost) all the details needed for their security proofs and privacy analyses.

DP-SSE utilizes vector and token generation functions different from those utilized by SSE-Naive. Let genVectorRd and genPredicateRd be the vector and token generation functions in DP-SSE, respectively. DP-SSE reveals one more thing, a label distribution function h, to server, which is used in both genVectorRd and genPredicateRd.



(a) Without Revealing Label Distribution

(b) Revealing Label Distribution

Figure 5.2: Search Complexity in DP-SSE.

Another difference in DP-SSE is the Search function. By revealing h and the label of each query token, the server can simplify the search process by evaluating FHIPPE.Query only on the search key and query token pairs which share the same label (refer to Figure 5.2).

- DP-SSE.Search(I, Γ_{F_w}): takes as input the search index I and a query token set Γ_{F_w}. Then for every query token $(\tau_f, label) \in \Gamma_{F_w}$, it calls FHIPPE.Query$(I[i], \tau_f), \forall i \in \{j | \mathcal{D}[j].label = label\}$; if the output is 0, then it returns the corresponding document identifier i. Therefore, the output of Search(I, Γ_{F_w}) is a set of document identifiers $\vec{id}$. Note that those identifiers are readable to the server, namely the server can still respond to the client with the corresponding encrypted documents $\{\mathcal{E}(\mathcal{D}[i]) \mid i \in \vec{id}\}$ in one round.

It should be noted that, except for what is mentioned above, all the other parts of DP-SSE are the same as in SSE-Naive.

5.3.1 genVectorRd

Function $\mathsf{genVectorRd}$ takes as input a document $\mathcal{D}[i] = \{w_1, w_2, ..., w_{|\mathcal{D}[i]|}\}$ and its document id i, and outputs an attribute vector v_i used later for encryption. Like $\mathsf{genVector}$, $\mathsf{genVectorRd}$ should also be able to be satisfied by any token generated from $w_j \in \mathcal{D}[i]$. The difference is $\mathsf{genVectorRd}$ should also have the ability to be satisfied by a manually created query token in order to generate false positive results. Following the same idea as in $\mathsf{genVector}$, we define the following polynomial (given document $\mathcal{D}[i]$):

$$P(x) = \prod_{j=1}^{|\mathcal{D}[i]|} \left(x - w_j \| h(i) \| counter_{w_j,i}\right) \cdot \prod_{j=|\mathcal{D}[i]|+1}^{S_{max}} (x - \gamma \| 0 \| 0) \cdot (x - i \| 0 \| \text{-}1) = \sum_{j=0}^{S_{max}+1} a_j \cdot x^j$$

where $w_j \in \mathcal{D}[i]$, for $j \in \left[|\mathcal{D}[i]|\right]$,$\gamma \notin \Delta$, $h(\cdot) \in [|h|]$ is a label distribution function, and $counter_{w_j,i}$ is a global variable indicating how many times $w_j \| h(i) \| \cdot$ have been previously generated. It should be noted that those three parts of $(\cdot \| \cdot \| \cdot)$ are allocated fixed length of bits each meaning that a difference in any part will lead to a totally different concatenation. Thus, $(\gamma \| 0 \| 0), (id_i \| 0 \| \text{-}1)$ and $(w \| h \| counter)$ are different from each other. Another thing to mention is that h takes as input explicitly the document identifier and implicitly all global variables, then outputs a label.

Similarly, all terms with γ are acting as placeholders and will never be used to generate predicates. Terms like $(i \| 0 \| \text{-}1)$ will be used when document $\mathcal{D}[i]$ is going to be returned as a false positive result.

Directly, we can assign $v_i = (a_0, a_1, ..., a_{S_{max}+1})$ which can be satisfied by any token generated by predicate f produced by $\mathsf{genPredicate}(w_j \| h(i) \| counter_{w_j,i})$ where $w_j \in \mathcal{D}[i]$ to generate true positive results. In the meantime, such assignment of v_i also allows satisfaction by a token generated by $\mathsf{genPredicateRd}(i \| 0 \| \text{-}1)$ to produce false positive result.

The reason for using a *counter* in $\mathsf{genVectorRd}$ function is that we want every token to at most match one document due to the fact that a false positive token (generated from $id \| 0 \| \text{-}1$) can only match at most one document (since every document has a unique *id*). Otherwise, if a token could match multiple documents, the server could determine that this token must be a true positive token (a token that only matches a true positive document) and the matched document is a true positive. Such token matching multiple documents could actually exist if we did not use *counter*. Suppose that two documents $\mathcal{D}[i]$ and $\mathcal{D}[j]$ such that there exists a keyword $w \in \mathcal{D}[i]$, $w \in \mathcal{D}[j]$ and $h(i) = h(j)$ (it will normally happen since $|h|$ is quite smaller than n). Then a token generated by f produced by $\mathsf{genPredicate}(w \| h(i))$ would match both $\mathcal{D}[i]$ and $\mathcal{D}[j]$. With the help of *counter*, we solve this issue.

The candidates of h is discussed in section 5.4.

5.3.2 genPredicateRd

Function genPredicateRd takes as input a keyword w, and outputs, instead of a single predicate as in genPredicate a predicate, a set of predicates F_w, each of which can only satisfy at most one document (a predicate f satisfies one document D means that the query token generated from f satisfies the search key generated from attribute v derived from D). The algorithm is as follows.

In a high level, algorithm 1 flips a coin over every possible *true positive predicate* to realize that a true positive document will be returned when selected with probability p and will not be returned when not selected with probability $1 - p$. A true positive predicate when searching for w is a predicate that will match a document which contains w. Note that the set of all true positive predicates can be predetermined due to the construction of search index I. The algorithm does the same thing to true negative documents to give them a chance of q to be matched.

The algorithm first initializes two empty sets `FalsePosSet` and `TruePosSet`. As indicated by their names, they are to carry possible candidates for generating false positive predicates and true positive predicates, respectively. A false positive predicate when searching for w is a predicate that will match a document which does not contain w. However, their names only reflect our intention. In fact, candidates in `FalsePosSet` might also match true positive documents and candidates in `TruePosSet` might match no documents. Each candidate in either set shares the same structure, say $X\|Y\|Z$. A candidate is called *true positive candidate* if it can be used to generate a true positive predicate. Given a keyword

Algorithm 1 Generating Predicates When Searching Keyword w

```
procedure genPredicateRd($w$)
    FalsePosSet $\leftarrow \emptyset$
    TruePosSet $\leftarrow \emptyset$
    for $label$ in $[1, |h|]$ do
        $counter \leftarrow 0$
        for $counter < counter_{max}$ do
            with probability p, TruePosSet.$add\big([w \| label \| counter, label]\big)$
            counter ++
    for $id$ in $ID_{db}$ do
        with probability q, FalsePosSet.$add\big([id \| 0 \| \text{-1}, h(id)]\big)$
    TValSet $\leftarrow$ FalsePosSet $\cup$ TruePosSet
    PredicateSet $\leftarrow \emptyset$
    for $tval, label \in$ TValSet do
        PredicateSet.$add\big([\text{genPredicate}(tval), label]\big)$
    return PredicateSet
```

w, all positive documents share the property that they can be matched by some predicate generated from some candidates in which $X = w$. Thus, by enumerating all possible value of Y and Z, the algorithm can simulate the coin flipping process over all true positive documents. As for true negative documents, the algorithm has not too much information other than they must be in the n documents. Recall that the search key of each document has one more term which enables to generate a false positive match. This special term shares the same structure $X\|0\|$-1 where $X \in [n]$. Therefore, the algorithm can realize the coin flipping process for true negative documents by enumerating all possible values of X. Finally, the algorithm calls $\mathsf{genPredicate}$ to convert all selected candidates into predicates.

Algorithm 1 guarantees that one predicate can only satisfy at most one document. However, it might happen that a document is satisfied by multiple predicates when searching for keyword w, i.e.

$$\exists D \in \mathcal{D}, w \in D, s.t.\ f \text{ and } f' \text{ are both in } \texttt{PredicateSet}.$$

where $f = \mathsf{genPredicate}\big(w\|h(id(D)\|counter_{w,id(D)})\big)$, and $f' = \mathsf{genPredicate}(id(D)\|0\|\text{-}1)$.

It can be shown that both f and f' satisfy document D, and they can be generated together by $\mathsf{genPredicateRd}(w)$ with some probability. The reason why this issue should be avoided is that it reveals that D must have keyword w since a document can only be matched by at most one false positive predicate, if D is matched by two predicates, the other one must be a true positive predicate revealing the document is a true positive.

In order to resolve the issue, an idea is to make the matching count for a document differentially private, i.e. only by the number of time that a document is matched, one can not tell whether a document is a true positive or a false positive.

Algorithm 2 adds another process when generating false positive candidates. It *repeat*s the coin flipping process on *potential* false positive candidates until the candidate set becomes empty. In this way, an *id* can be added to `FalsePosSet` multiple times (note that `FalsePosSet` is a multiset which allows duplicates) meaning that for a document, it is possible to be matched by multiple false positive predicates. It also means if there is a document which is matched multiple times, one cannot determine whether it is a true positive or a false positive. It should be noted that the number of times for a single false

Algorithm 2 Generating Predicates When Searching Keyword w

```
procedure genPredicateRd($w$)
    FalsePosSet $\leftarrow \emptyset$
    TruePosSet $\leftarrow \emptyset$
    for $label$ in $[1, |h|]$ do
        $counter \leftarrow 0$
        for $counter < counter_{max}$ do
            with probability p, TruePosSet.$add\big([w \| label \| counter, label]\big)$
            counter ++
    tmpSet $\leftarrow \{1, 2, ..., n\}$
    while tmpSet $\neq \emptyset$ do
        for $id$ in tmpSet do
            with probability q, FalsePosSet.$add\big([id \| 0 \| \text{-1}, h(id)]\big)$    ▷ FalsePosSet is a multiset, it allows duplicates
            else  tmpSet.$remove(id)$
    TValSet $\leftarrow$ FalsePosSet $\cup$ TruePosSet
    PredicateSet $\leftarrow \emptyset$
    for  $tval, label \in$ TValSet do
        PredicateSet.$add\big([\text{genPredicate}(tval), label]\big)$
    return PredicateSet
```

positive candidate to be added to `FalsePosSet` follows a geometric distribution, which makes the process of generating false positive tokens converges quickly.

Another issue with Algorithm 2 is that a *non-match* token, i.e. a token that matches no documents, must be from `TruePosSet`. Since there is an upper bound of the size of `TruePosSet`, i.e. $ST_{max} = counter_{max} \cdot |h|$, if there are ST_{max} non-matches, it reveals that all matched documents are from `FalsePosSet` which violates the differential privacy guarantee. To avoid this, we use a similar trick as above to make the number of *non-match*es differentially private.

In Algorithm 3 (the actual algorithm used in DP-SSE), some manually made *non-match* candidates are added according to a geometric distribution. The geometric distribution guarantees that the predicate generation process will quickly converge and the number of manually made *non-match* candidates will be small. It should be noted that utilizing algorithm 3 will result in a true positive rate $p+q-pq$ and a false positive rate q. In other words, algorithm 3 leads to a differentially private SSE of utility $(p+q-pq)(1-q)$.

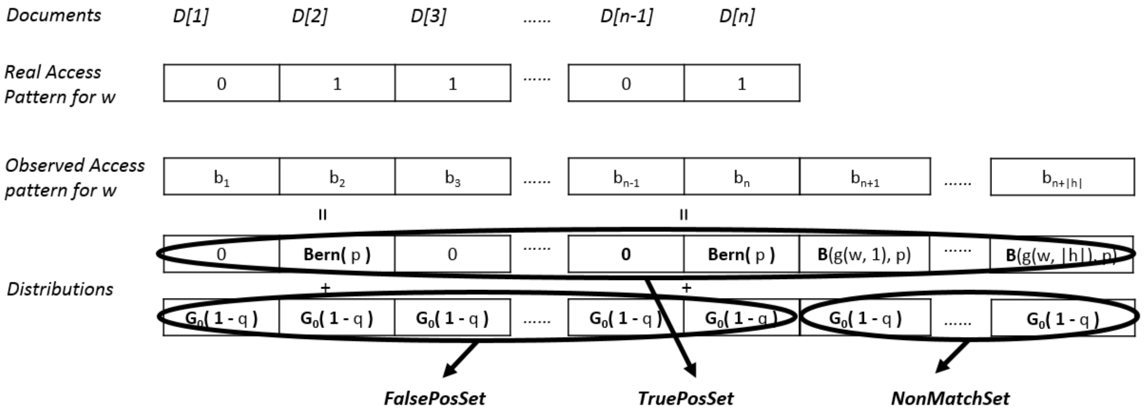


Figure 5.3: Access Pattern Leakage in DP-SSE

In the rest of this section, we show how the properties introduced in section 5.1 are achieved in algorithm 3. In a high level, as displayed by Figure 5.3, the process of generating `TruePosSet` contributes all the Bernoulli distributions and binomial distributions; the process of generating `FalsePosSet` contributes all the first n geometric distributions; the

Algorithm 3 Generating Predicates When Searching Keyword w

```
procedure GENPREDICATERD($w$)
    FalsePosSet $\leftarrow \emptyset$
    TruePosSet $\leftarrow \emptyset$
    for $label$ in $[1, |h|]$ do
        $counter \leftarrow 0$
        for $counter < counter_{max}$ do
            with probability p, TruePosSet.$add\big([w \| label \| counter, label]\big)$
            counter ++
    tmpSet $\leftarrow \{1, 2, ..., n\}$
    while tmpSet $\neq \emptyset$ do
        for $id$ in tmpSet do
            with probability q, FalsePosSet.$add\big([id \| 0 \| -1, h(id)]\big)$    ▷ FalsePosSet is a multiset, it allows duplicates
            else  tmpSet.$remove(id \| 0 \| -1))$
    NonMatchSet $\leftarrow \emptyset$
    for $label$ in $[1, |h|]$ do
        mark $\leftarrow True$
        while mark do
            with probability q, NonMatchSet.$add\big([w_{-1} \|$-1$\| 0, label]\big)$
            else mark $\leftarrow False$
    TValSet $\leftarrow$ FalsePosSet $\cup$ TruePosSet $\cup$ NonMatchSet
    PredicateSet $\leftarrow \emptyset$
    for  $tval, label \in$ TValSet do
        PredicateSet.$add\big([\text{genPredicate}(tval), label]\big)$
    return PredicateSet
```

process of generating `NonMatchSet` contributes the remaining geometric distributions. Now we discuss them in detail. Recall the real access pattern and access pattern in DP-SSE are denoted by $(a_1, a_2, ..., a_n)$ and $(b_1, b_2, ..., b_n, b_{n+1}, ..., b_{n+|h|})$, respectively. Recall that for $i \leq n$, $b_i = u_i + v_i$ if $a_i = 1$ where $u_i \sim \mathbf{Bern}(p)$ and $v_i \sim \mathbf{G}_0(1-q)$. This is because a true positive document indicated by $a_i = 1$ can be matched by a true positive token originated from `TruePosSet` which is generated from a Bernoulli distribution and a false positive token originated from `FalsePosSet` which is generated from a geometric distribution. For $i \leq n$ and if $a_i = 0$, the corresponding document can only be matched by false positive tokens which are generated from a geometric distribution, in other words, $b_i \sim \mathbf{G}_0(1-q)$. For $i \leq |h|$, b_{n+i} represents the number of non-match tokens of label i. The non-match tokens comes from two sources: $\texttt{TruePosSet}_i$ and $\texttt{NonMatchSet}_i$ where $\texttt{TruePosSet}_i$ and $\texttt{NonMatchSet}_i$ represent the subset of `TruePosSet` and `NonMatchSet` with label i, respectively. $\texttt{NonMatchSet}_i$ is generated from a geometric distribution. While for $\texttt{TruePosSet}_i$, each element in it is generated from a Bernoulli distribution and there are $\mathit{counter}_{max} - |\{\mathcal{D}[j] \mid w \in \mathcal{D}[j], h(j) = i\}|$ candidates for non-match tokens, therefore this contributes a Binomial distribution to b_{n+i}. In the meantime, we obtain the real value of $g(w, i)$, i.e. $g(w, i) = \mathit{counter}_{max} - |\{\mathcal{D}[j] \mid w \in \mathcal{D}[j], h(j) = i\}|$.

5.4 Label Distribution Function: h and $\mathit{counter}_{max}$

Label distribution function h works in $\mathsf{genVectorRd}$ and affects the construction of $\mathsf{genPredicateRd}$ (since they are highly correlated) and the value of $\mathit{counter}_{max}$ (since it reflects how uniform h is). Recall the polynomial $P(x)$ used to define $\mathsf{genVectorRd}(\mathcal{D}[i], i)$.

$$P(x) = \prod_{j=1}^{|\mathcal{D}[i]|} \left(x - w_j \| h(i) \| \mathit{counter}_{w_j,i}\right) \times \prod_{j=|\mathcal{D}[i]|+1}^{S_{max}} (x - \gamma \| 0 \| 0) \times (x - i \| 0 \| \text{-}1)) = \sum_{j=0}^{S_{max}+1} a_j \cdot x^j$$

In $P(x)$, every keyword $w \in \mathcal{D}[i]$ is concatenated with the label of $\mathcal{D}[i]$ denoted by $h(i)$ and a counter $\mathit{counter}_{w,i}$ to make such concatenation $w \| h(i) \| \mathit{counter}_{w,i}$ different for every document which has w, i.e. $w \| h(i) \| \mathit{counter}_{w,i} \neq w \| h(j) \| \mathit{counter}_{w,j}$ for all $i \neq j$ where $w \in \mathcal{D}[i]$ and $w \in \mathcal{D}[j]$. To achieve this, h is first used to label every document which can be viewed as a way of grouping documents by label; then for each small group with the same label, $counter_{w,\cdot}$ is used to distinguish documents containing w by assigning different values (choose from $\mathbb{N}$ one by one) to $counter_{w,i}$ and $counter_{w,j}$ for every i and j where $h(i) = h(j)$, $w \in \mathcal{D}[i]$ and $w \in \mathcal{D}[j]$. $counter_{max}$ models the number of distinct values $counter$ could be.

Scheme	h	$counter_{max}$
DP-SSE-1	constant function	C_{max}
DP-SSE-2	one hash function	$O\left(\frac{\log n}{\log\log n}\right)$
DP-SSE-3	two hash functions	$O(\log\log n)$

Table 5.2: $counter_{max}$

In this section, we introduce 3 candidates of h, compute the corresponding $counter_{max}$ (Table 5.2 summarize the results) and apply each of them to DP-SSE to get 3 differentially private SSE, denoted by DP-SSE-1, DP-SSE-2 and DP-SSE-3, respectively. As will be discussed in section 8.3, given $|h|$, the smaller $counter_{max}$ is, the smaller the communication complexity and the computation complexity will be. There are some other options of h. However, we will show in section 8.3 that it is NP-hard to find an h with $|h| \geq 3$ such that $counter_{max}$ is minimized.

5.4.1 h As A Constant Function

The most naive way to choose h is to make it a constant function. In order to distinguish from concatenations for other purposes (false positive or non-match), we can choose h to be equal to any positive integer. Without loss of generality, we choose $h \equiv 1$. Then $|h| = 1$. By replacing $h(\cdot)$ with 1 and $|h|$ with 1 in DP-SSE, we obtain DP-SSE-1. It is easy to check that $counter_{max}$ in this case is C_{max}.

One might be surprised that such causal choice of h will result in a differentially private SSE which has the least communication complexity. However, the computation complexity of DP-SSE-1 is huge. Both communication complexity and computation complexity will be discussed in section 8.

5.4.2 h As a Hash Function

Another straightforward way is to choose h as a hash function. By defining h as a hash function of range $[C_{max}]$, i.e. $h : [n] \rightarrow [C_{max}]$ and applying such h directly to DP-SSE, we obtain DP-SSE-2.

To calculate $counter_{max}$ in DP-SSE-2, we formulate the following equivalent *Balls into Bins* problem: sequentially throw k balls into k bins by placing each ball into a bin chosen independently and uniformly at random; what is the maximum number of balls in any bin? The only difference is that we need to distribute $|\Delta|$ keywords which is analogous to playing the *Balls into Bins* game $|\Delta|$ times. By doing so, we implicitly assume that keywords are distributed independently.

We first show a Lemma in *Balls into Bins* problems and then apply it to approximate $counter_{max}$ with the help of the union bound.

Lemma 5.4.1. Sequentially throw k balls into k bins by placing each ball into a bin chosen independently and uniformly at random, the maximum number of balls in any bin is $\dfrac{c \ln k}{\ln \ln k}$ where $c \geq 3$ with probability at least $1 - \dfrac{1}{k^{c-1}}$.

Proof. Refer to Appendix A.2 □

Theorem 5.4.2. If h is chosen as a hash function of range $[C_{max}]$, i.e. $h : [n] \rightarrow [C_{max}]$, $counter_{max} = O\left(\dfrac{\log C_{max}}{\log \log C_{max}}\right) = O\left(\dfrac{\log n}{\log \log n}\right)$ with probability at least $1 - \dfrac{1}{n}$ assuming $|\Delta| = O(n)$ and $C_{max} = O(n)$.

Proof. Refer to Appendix A.4 □

DP-SSE-2 has larger communication complexity but much smaller computation complexity than DP-SSE. The detailed analysis of complexity is in section 8.

5.4.3 h As the Better Choice of Two Hash Functions

In the *Balls into Bins* problems, there is a *2-choice* solution in which each ball is put into the bin with less balls out of two uniformly at random chosen bins. In this way, the maximum number of balls in any bin is $O(\log\log k)$ when throwing k balls into k bins.

Inspired by this greedy solution, we can define the output of h as the better output (the output which makes *counter* smaller) of two hash functions $h_1, h_2 : [n] \to [C_{max}]$, which will also make $counter_{max}$ smaller. As we will see later, smaller $counter_{max}$ makes a smaller communication complexity and a smaller computation complexity.

Applying such h to DP-SSE involves modifications of genVectorRd, genPredicateRd and Search. The reason of modifying Search is as follows. Two choices of h means that there are two label candidates for every document. With a high probability, both two label candidates, say l_1 and l_2, for a document D would be used to generate concatenations of some keywords w_i and $w_j \in D$, i.e. $w_i \| l_1 \| counter_{w_i,id(D)}$ and $w_j \| l_2 \| counter_{w_j,id(D)}$. Therefore, we cannot assign just one label to D, instead we need to preserve both labels for each document and evaluate FHIPPE.Query on document (search key) and token pairs as long as the token's label matches either label of the document.

- DP-SSE-3.Search(I, Γ_{F_w}): takes as input the search index I and query token set Γ_{F_w}, for every query token $(\tau_f, label) \in \Gamma_{F_w}$, it calls FHIPPE.Query$(I[i], \tau_f), \forall i \in \{j | label \in \mathcal{D}[j].labels\}$. If the output is 0, then it returns the corresponding document identifier i. Therefore, the output of Search(I, Γ_{F_w}) is a set of document identifiers $\vec{id}$. Note that those identifiers are readable to the server, namely the server can respond to the client with the corresponding encrypted documents $\{\mathcal{E}(\mathcal{D}[i]) \mid i \in \vec{id}\}$ in one round.

The new DP-SSE-3.Search function is actually a composition of two DP-SSE-2.Search functions in which one is using h_1 to label documents and the other using h_2 to do so. The result of DP-SSE-3.Search is the union of the results of the two. Therefore, the access pattern observed by the server, which is originally a vector of length $n + |h|$, changes to two vectors of length $n + |h_1|$ and $n + |h_2|$, respectively. Since $|h_1|$ and $|h_2|$ are chosen the same as $|h| = C_{max}$, the new access pattern becomes two vectors of length $n + |h|$ where $|h| = C_{max}$. Let $(\Pi_w^{(1)}, \Pi_w^{(2)})$ be the access pattern of DP-SSE-3 when searching for w where $\Pi_w^{(1)} = (b_1^{(1)}, ..., b_n^{(1)}, b_{n+1}^{(1)}, ..., b_{n+|h|}^{(1)})$ and $\Pi_w^{(2)} = (b_1^{(2)}, ..., b_n^{(2)}, b_{n+1}^{(2)}, ..., b_{n+|h|}^{(2)})$. For $i \leq n$, $b_i^{(t)}$ represents the number of times that $\mathcal{D}[i]$ is matched when it is only labeled by h_t for $t \in \{1, 2\}$. For $i \leq |h|$, $b_{n+i}^{(t)}$ represents the number of non-match tokens to have label i when all documents are labeled by h_t for $t \in \{1, 2\}$. In order to preserve the properties introduced in section 5.1, genVectorRd and genPredicateRd need to be changed. Recall the polynomial $P(x)$ used to define genVectorRd$(\mathcal{D}[i], i)$.

$$P(x) = \prod_{j=1}^{|\mathcal{D}[i]|} \left(x - w_j \| h(i) \| counter_{w_j,i}\right) \cdot \prod_{j=|\mathcal{D}[i]|+1}^{S_{max}} (x - \gamma \| 0 \| 0) \cdot (x - i \| 0 \| \text{-}1) = \sum_{j=0}^{S_{max}+1} a_j \cdot x^j$$

genVectorRd. For each document $\mathcal{D}[i]$ and for each keyword w in $\mathcal{D}[i]$, choose $h(i)$ as $h_b(i)$ where $b \in \{0, 1\}$ and the number of times that $w \| h_b(i)$ is chosen as $h(i)$ is less than $w \| h_{1-b}(i)$. In other words, $h(i)$ is chosen as the value that makes $counter_{w,i}$ smaller. After $h(i)$ is determined, $counter_{w,i}$ can be determined. In order to add geometrically distributed variables to both of the two access patterns $\Pi_w^{(1)}$ and $\Pi_w^{(2)}$ to hide whether a document is matched by a true positive token or not, we need to modify $P(x)$ in such a way that it supports adding false positives for both cases while using 2 hash functions. The following polynomial achieves such property.

$$\begin{aligned} P'(x) &= \prod_{j=1}^{|\mathcal{D}[i]|} \left(x - w_j \| h(i) \| counter_{w_j,i}\right) \\ &\cdot \prod_{j=|\mathcal{D}[i]|+1}^{S_{max}} (x - \gamma \| 0 \| 0) \\ &\cdot (x - i \| h_1(i) \| \text{-}1)) \cdot (x - i \| h_2(i) \| \text{-}1) \\ &= \sum_{j=0}^{S_{max}+2} a_j \cdot x^j \end{aligned}$$

It should be noted that $i \| h_t(i) \|$-1 will never be equal to some $w \| h \| counter$ or $\gamma \| 0 \| 0$, meaning that $i \| h_t(i) \|$-1 can only be used to generate false positive for $t \in \{1, 2\}$. With the new polynomial $P(x)$ and the new way to assign values to $h(i)$ and $counter_{w,i}$, the new genVectorRd is well-defined by preserving all the other parts of the old genVectorRd.

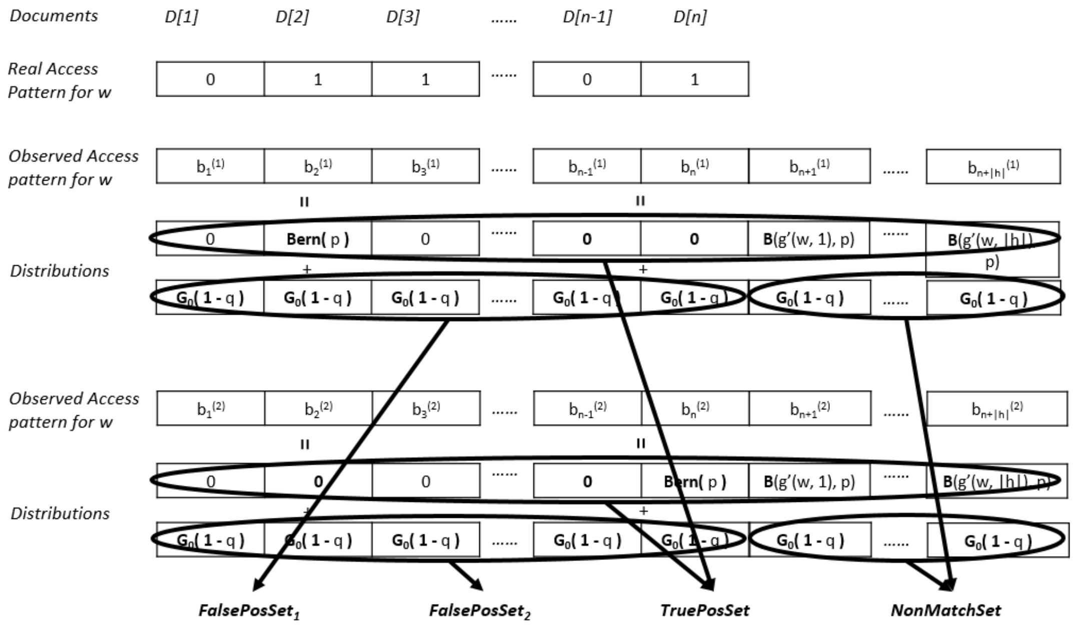


Figure 5.4: Access Pattern Leakage in DP-SSE-3

genPredicateRd. The new construction of genPredicateRd is shown in Algorithm 4. Figure 5.4 demonstrates how this algorithm can achieve properties that are similar to the old algorithm 3. It should be noted that, when searching for w, `TruePosSet` can only add one Bernoulli distribution to either $\Pi_w^{(1)}$ or $\Pi_w^{(2)}$ in every position i where $a_i = 1, i \leq n$ unless $h_1(i) = h_2(i)$ in which case the Bernoulli distribution will be added to both. For example in Figure 5.4, for $i = 2$, the Bernoulli distribution is added to $\Pi_w^{(1)}$; while for $i = n$, the Bernoulli distribution is added to $\Pi_w^{(2)}$. The existence of those geometric distributions acts as an important role to provide differential privacy.

With the above modifications, we obtain DP-SSE-3 by preserving all the other parts of DP-SSE. It should be noted that the true positive rate and false positive rate of DP-SSE-3 become $1-(1-p)(1-q)^2$ and $1-(1-q)^2$, respectively, namely the utility of DP-SSE-3 is $(1-(1-p)(1-q)^2)(1-(1-q)^2)$.

Algorithm 4 Generating Predicates When Searching Keyword w

```
procedure genPredicateRd($w$)
    $\texttt{FalsePosSet}_1, \texttt{FalsePosSet}_2, \texttt{TruePosSet}, \texttt{NonMatchSet} \leftarrow \emptyset$
    for $label$ in $[1, |h|]$ do
        $counter \leftarrow 0$
        for $counter < counter_{max}$ do
            with probability p, $\texttt{TruePosSet}.add\big([w \| label \| counter, label]\big)$
            counter ++
    $\texttt{tmpSet} \leftarrow \{1, 2, ..., n\}$
    while $\texttt{tmpSet} \neq \emptyset$ do
        for $id$ in tmpSet do
            with probability q, $\texttt{FalsePosSet}_1.add\big([id \| h_1(id) \| -1, h_1(id)]\big)$
            else  $\texttt{tmpSet}.remove(id)$  ▷ $\text{FalsePosSet}_1$ is a multiset, it allows duplicates
    $\texttt{tmpSet} \leftarrow \{1, 2, ..., n\}$
    while $\texttt{tmpSet} \neq \emptyset$ do
        for $id$ in tmpSet do
            with probability q, $\texttt{FalsePosSet}_2.add\big([id \| h_2(id) \| -1, h_2(id)]\big)$
            else  $\texttt{tmpSet}.remove(id)$  ▷ $\text{FalsePosSet}_2$ is a multiset, it allows duplicates
    for $label$ in $[1, |h|]$ do
        mark $\leftarrow True$
        while mark do
            with probability q, $\texttt{NonMatchSet}.add\big([w_{-1} \| \text{-1} \| 0, label]\big)$
            else mark $\leftarrow False$
    $\texttt{TValSet} \leftarrow \texttt{FalsePosSet}_1 \cup \texttt{FalsePosSet}_2 \cup \texttt{TruePosSet} \cup \texttt{NonMatchSet}$
    $\texttt{PredicateSet} \leftarrow \emptyset$
    for  $tval, label \in \texttt{TValSet}$ do
        $\texttt{PredicateSet}.add\big([\text{genPredicate}(tval), label]\big)$
    return PredicateSet
```

We can use a similar *Balls into Bins* problem as in section 5.4.2 to show that the $counter_{max}$ of DP-SSE-3 is $O(\log\log n)$.

Lemma 5.4.3. Sequentially throw k balls into k bins by placing each ball into the bin that has less balls out of two independently and uniformly at random chosen bins. The maximum number of balls in any bin is $O\left(\log\log k\right)$ with probability at least $1 - \dfrac{1}{k^3}$.

Proof. Refer to Appendix A.3 □

Theorem 5.4.4. In DP-SSE-3, $counter_{max} = O\left(\log\log C_{max}\right) = O\left(\log\log n\right)$ with probability at least $1 - \dfrac{1}{n}$ assuming $|\Delta| = O(n)$ and $C_{max} = O(n)$.

Proof. Refer to Appendix A.5 □

DP-SSE-3 has exponentially smaller $counter_{max}$ than DP-SSE-2 which makes communication complexity and computation complexity smaller. However, we shall see later in section 7, the privacy level of DP-SSE-3 is weaker while providing the same utility compared to others.

Chapter 6

Security

In this chapter, we first show the leakage of DP-SSE. The leakage includes not only (differentially private) access patterns like in other schemes but also some meta data specific to DP-SSE. Then, we prove that with such leakage a simulator can simulate the view of an adversary who is allowed to adaptively issue queries, namely DP-SSE is adaptively semantically secure in the condition that the underlying FHIPPE scheme is SIM-secure. It should be noted that the following arguments is based on an abstract h. With minor changes, one could prove that DP-SSE-1, DP-SSE-2 and DP-SSE-3 are all adaptively semantically secure.

6.1 Leakage

The leakage here refers to the information that the server can learn about the database $\mathcal{D}$ and the underlying keyword list $\vec{w}$ of queries when hosting the searchable encryption scheme. In other words, the leakage is the trace of a history. It can be divided into two parts by the time point when the client begins issuing query tokens, namely $L = \{L_{init}, L_{search}\}$.

Before the client begins the search, the server can learn $|\mathcal{D}[i]|$, S_{max}, and C_{max}, as well as the id of each document. Let L_{init} be such leakage during initialization, i.e. $L_{init} = \big\{\{id(\mathcal{D}[i])\},\ \{|\mathcal{D}[i])|\},\ S_{max},\ C_{max}\big\}$. It should be noted that $id(\mathcal{D}[i]) = i$.

After the client begins the search, the server will learn (differentially private) access patterns $\Pi_{\vec{w}}$ for a list of keywords $\vec{w}$ (allowing repeated keywords) that the client has searched. Note that $\Pi_{\vec{w}}$ is a differentially private version of the real access patterns $\hat{\Pi}_{\vec{w}}$. As discussed in chapter 8, the client reveals the *label assignments* to server due to performance considerations. In that case, label distribution function h would be learned by the server in addition. We argue that leaking h is of no harm to the scheme since h is independent of all sensitive information, say the history. Let $L^{\vec{w}}_{search}$ be such leakage after searching, i.e. $L^{\vec{w}}_{search} = \{\Pi_{\vec{w}}, h\}$.

After analyzing the leakage of the scheme, we reach a stage to rewrite the definition of trace which defines the leakage of an SSE scheme (definitions of history and view remain unchanged).

Definition 6.1.1. (New Trace) The trace of a history $H_{\vec{w}}$ for searching for a list of keywords $\vec{w}$ in DP-SSE is defined as $T_{\vec{w}} = \{L_{init}, L^{\vec{w}}_{search}\}$.

Compared with the old definition of trace, the access patterns are replaced with their differentially private versions and the search patterns are removed. Some additional information is included in the new trace, namely label distribution function h, C_{max} and S_{max}. We will show in the next section that DP-SSE is adaptively semantically secure under the new definition of trace.

6.2 Security

Theorem 6.2.1. DP-SSE is an adaptively semantically secure SSE scheme.

Proof. We are going to construct a simulator $\mathcal{S}$ that can simulate the partial view of an adversary given only a trace of a partial history. To be precise, given $T(H^s_t)$, $\mathcal{S}$ can generate a view $(V^s_t)^*$ such that $(V^s_t)^*$ is indistinguishable from $V^s_{sk}(H_t)$ for all $0 \leq s \leq t$, all polynomial-bounded function f_p, all probabilistic polynomial-time adversary $\mathcal{A}$, all distribution $\mathcal{H}_t$, except with a negligible probability, where $q \in \mathbb{N}, H_t \overset{R}{\leftarrow} \mathcal{H}_t$ and $sk \leftarrow \mathsf{Keygen}(1^k)$.

Before describing the details about $\mathcal{S}$, let's recall the notions of view and trace when searching for $\vec{w} = [\vec{w}[1], \vec{w}[2], ..., \vec{w}[t]]$ in the context of DP-SSE:

- $V_{sk}(H_t) = (1, ..., n, \mathcal{E}(\mathcal{D}[1]), ..., \mathcal{E}(\mathcal{D}[n]), I, \Gamma_1, ..., \Gamma_t)$;
- $V_{sk}^s(H_t) = (1, ..., n, \mathcal{E}(\mathcal{D}[1]), ..., \mathcal{E}(\mathcal{D}[n]), I, \Gamma_1, ..., \Gamma_s)$, where $0 \leq s \leq t$;
- $T(H_t) = (1, ..., n, |\mathcal{D}[1]|, ..., |\mathcal{D}[n]|, h, S_{max}, C_{max}, \Pi_{\vec{w}[1]}, ..., \Pi_{\vec{w}[t]})$;
- $T(H_t^s) = (1, ..., n, |\mathcal{D}[1]|, ..., |\mathcal{D}[n]|, h, S_{max}, C_{max}, \Pi_{\vec{w}[1]}, ..., \Pi_{\vec{w}[s]})$, where $0 \leq s \leq t$.

There are three types of data to simulate the view of an adversary $\mathcal{A}$: document encryption $\mathcal{E}(\mathcal{D}[i])$, search key $I[i]$, and search token list Γ.

Simulate $\mathcal{E}(\mathcal{D}[i])$:

$\mathcal{E}(\mathcal{D}[i])$ is indistinguishable from a random string $e_i^* \xleftarrow{R} \{0,1\}^{|\mathcal{D}[i]|}$ since $\mathcal{E}$ is semantically secure.

Γ_i cannot be simulated directly as $\mathcal{E}(\mathcal{D}[i])$ since it should embed some functionalities in order to enable search. When it comes to the simulation of I, things become more complex.

In order to provide adaptively security, $\mathcal{S}$ must commit to an index I^* before any queries are issued. In other words, $\mathcal{S}$ must generate, at time $s = 0$, an index I^* which will be included in all partial views $(V_t^s)^*$ used to simulate $\mathcal{A}$ for any $0 \leq s \leq t$. I^* should satisfy two requirements: first, it should be indistinguishable from I; second, it should be able to simulate all $\mathcal{A}$'s views, namely answer future unknown queries. Recall that I is composed of n search keys, say $I[1], I[2], ..., I[n]$, generated from documents, say $\mathcal{D}[1], \mathcal{D}[2], ..., \mathcal{D}[n]$, respectively. $\mathcal{S}$ first runs $\mathsf{Keygen}(1^\lambda)$ to get a secret key sk^*; then $\mathcal{S}$ simulates each $I[i]$ and each Γ as follows.

Simulate $I[i]$:

At time $s = 0$, $\mathcal{S}$ only holds $T(H_t^0) = (1, ..., n, |\mathcal{D}[1]|, ..., |\mathcal{D}[n]|, h, S_{max}, C_{max})$ by which $\mathcal{S}$ generates $I[i]$ as follows: first, $\mathcal{S}$ uniformly at random samples $S_{max} + 1$ random numbers, say U, from a range disjoint with $[n]$, e.g., $[5n, 5n+1, ..., 6n]$; then, $\mathcal{S}$ calls FHIPPE.$\mathsf{Encrypt}(sk^*, \mathsf{genVectorRd}(U, i))$ to compute $I^*[i]$. I^* can be otained by combining all $I^*[i]$ together. We claim that I^* is indistinguishable from I which will be proved later in correctness part.

Simulate $\Gamma[i]$:

For $1 \leq s \leq t$, $\mathcal{S}$ simulates $\mathcal{E}(\mathcal{D}[i])$ and I as above. To construct Γ_s^* such that Γ_s^* is indistinguishable from Γ_s, $\mathcal{S}$ does the following (Γ_i^* for $i < s$ can be constructed similarly): first, $\mathcal{S}$ defines a set $X = \bar{\mathcal{D}}(\vec{w}[s])$ which is the *id* set of all returned documents and label each element $j \in X$ by $h(j)$; second, $\mathcal{S}$ extends X into a multiset X^* where each element in X is duplicated according to Π_{w_s}, i.e., if $\mathcal{D}[i]$ is matched by Γ_s for $\Pi_{\vec{w}[s]}[i]$ times, there would be $\Pi_{\vec{w}[s]}[i]$ elements of value i in X^*; third, $\mathcal{S}$ adds $\Pi_{\vec{w}[s]}[n+j]$ *non-match* predicates with label j to X^* for $j \in [|h|]$; finally, set $\mathsf{TValSet} = X^*$ and run all the remaining parts (lines after 19) of algorithm 3 (together with sk^*) to get Γ_s^*. We claim Γ_s^* is indistinguishable from Γ_s which will be proved later in correctness part.

Correctness:
We prove I^* is indistinguishable from I by contradiction. If there exists an adversary $\mathcal{A}$ which can distinguish I^* from I, we show that $\mathcal{A}$ breaks SIM-security.

Assume $\mathcal{A}$ can distinguish $I^*[i]$ from $I[i]$. Consider a simulator $\mathcal{S}'$ which simulates everything like $\mathcal{S}$ but with secret key sk. Let $I'[i]$ and Γ_i' be the simulation of I and Γ_i in $\mathcal{S}'$, respectively. Since FHIPPE is SIM-secure, it must be also fully secure. Therefore, no adversary including $\mathcal{A}$ can distinguish $I'[i]$ from $I[i]$. Since $\mathcal{A}$ can distinguish $I^*[i]$ from $I[i]$, $\mathcal{A}$ must be able to distinguish $I'[i]$ from $I^*[i]$.

On the other hand, since FHIPPE is SIM-secure, there exists a simulator $\mathcal{S}_{\text{FHIPPE}}$ such that it can simulate all $\mathcal{A}$'s views about $I'[i]$ by some $I'_{\text{FHIPPE}}[i]$. Since $\mathcal{A}$ can distinguish $I'[i]$ from $I^*[i]$, it must also be able to distinguish $I'_{\text{FHIPPE}}[i]$ from $I^*[i]$. However, $I'_{\text{FHIPPE}}[i]$ should be an indistinguishable simulation of $I^*[i]$ due to the fact the underlying secrets of $I'[i]$ and $I^*[i]$ (plus Γ_i' and Γ_i^*) are identical (or in other words, the information that $\mathcal{S}_{\text{FHIPPE}}$ is able to use to simulate $I'[i]$ and $I^*[i]$ is identical). Thus, if $\mathcal{A}$ can distinguish $I'[i]$ from $I^*[i]$, $\mathcal{A}$ can break SIM-security contradicting the condition that FHIPPE is SIM-secure.

Using the same proof strategy, we can prove that Γ_i^* is indistinguishable from Γ_i. □

Chapter 7

Differential Privacy

In this chapter, we will show that the meta-scheme DP-SSE is a differentially private SSE. Then we will show using a similar analysis strategy that $\text{DP-SSE-1}, \text{DP-SSE-2}$ and DP-SSE-3 are all differentially private SSEs.

Recall that a differentially private SSE is an SSE which provides differential privacy for both documents and keywords.

Differential privacy for documents requires an SSE to satisfy that for any pair of neighbouring databases $\mathcal{D}, \mathcal{D}' \in 2^{2^{\Delta}}$, and for any keyword list $\vec{w} \in \Delta^{|\vec{w}|}$,

$$Pr[\mathcal{SE}(\mathcal{D}, \vec{w}) \in S] \leq e^{|\vec{w}| \cdot \epsilon} Pr[\mathcal{SE}(\mathcal{D}', \vec{w}) \in S]$$

where $S \subset \mathcal{T}(\mathcal{SE})$.

Differential privacy for keywords requires an SSE to satisfy that for any database $\mathcal{D} \in 2^{2^{\Delta}}$, and for any pair of neighbouring keyword lists $\vec{w}, \vec{w}' \in \Delta^{|\vec{w}|}$ where $\vec{w}[i] \neq \vec{w}'[i]$,

$$Pr[\mathcal{SE}(\mathcal{D}, \vec{w}) \in S] \leq e^{d \cdot \epsilon} Pr[\mathcal{SE}(\mathcal{D}, \vec{w}') \in S]$$

where $S \subset \mathcal{T}(\mathcal{SE})$ and $d = d(\mathcal{D}(\vec{w}[i]), \mathcal{D}(\vec{w}'[i]))$.

In the above two definitions, the output of $\mathcal{SE}$ is defined as the trace over some history. Recall the trace $T(H_{\vec{w}})$ of history $H_{\vec{w}} = (\mathcal{D}, \vec{w})$ is defined as

$$(1, ..., n, |\mathcal{D}[1]|, ..., |\mathcal{D}[n]|, h, S_{max}, C_{max}, \Pi_{\vec{w}[1]}, ..., \Pi_{\vec{w}[t]}).$$

One might have noticed that except for the access patterns in the trace, all the other parts are identical among $T(\mathcal{D}, \vec{w}), T(\mathcal{D}, \vec{w}')$ and $T(\mathcal{D}', \vec{w})$. Therefore, we only need to show that the access-pattern leakage is differentially private in DP-SSE.

Theorem 7.0.1. DP-SSE is a (u, ϵ, ϵ)-differentially private SSE where $u = (p+q-pq)(1-q)$ and $\epsilon = \ln\left(1 + \dfrac{p}{q(1-p)}\right)$ if the output of h only depends on document ids.

Proof. Let $\mathcal{M}$ denote the mechanism in DP-SSE and let the output of $\mathcal{M}$ be the access pattern. Since the output of h only depends on document ids, this means that $h(i)$ is fixed no matter what the content of $\mathcal{D}[i]$ is.

Differential Privacy for Documents. Consider a pair of neighbouring databases $\mathcal{D}$ and $\mathcal{D}'$ where $\mathcal{D}[i] \neq \mathcal{D}'[i]$ and $|\mathcal{D}[i]| = |\mathcal{D}'[i]|$, and a list of query keywords $\vec{w} \in \Delta^{|\vec{w}|}$. Let w be the keyword such that w is only in one of $\mathcal{D}$ and $\mathcal{D}'$.

If $w \notin \vec{w}$, then $Pr[\mathcal{M}(\mathcal{D}, \vec{w}) \in S] = Pr[\mathcal{M}(\mathcal{D}', \vec{w}) \in S]$.

If $w \in \vec{w}$, let $\Lambda = \{j \mid \vec{w}[j] = w\}$ and $\vec{w}_\Lambda = (\vec{w}[\Lambda[1]], \vec{w}[\Lambda[2]], ..., \vec{w}[\Lambda[|\Lambda|]])$ where $\Lambda[j]$ represents the j^{th} smallest element in Λ.

$$
\begin{aligned}
&\frac{Pr[\mathcal{M}(\mathcal{D}, \vec{w}) \in S]}{Pr[\mathcal{M}(\mathcal{D}', \vec{w}) \in S]} = \frac{Pr[\mathcal{M}(\mathcal{D}, \vec{w}_\Lambda) \in S]}{Pr[\mathcal{M}(\mathcal{D}', \vec{w}_\Lambda) \in S]} = \left(\frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]}\right)^{|\Lambda|} \\
&\frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]} \leq \max_{\Pi \in S} \left\{ \frac{Pr[\mathcal{M}(\mathcal{D}, w) = \Pi]}{Pr[\mathcal{M}(\mathcal{D}', w) = \Pi]} \right\} \\
&\qquad \leq \begin{cases} \max_{\alpha,\beta} \dfrac{Pr[u_i + v_i = \alpha, b_{n+h(i)} = \beta]}{Pr[v'_i = \alpha, b'_{n+h(i)} = \beta]}, \text{ if } w \in \mathcal{D}[i], \text{ but } w \notin \mathcal{D}'[i], \\ \max_{\alpha,\beta} \dfrac{Pr[v_i = \alpha, b_{n+h(i)} = \beta]}{Pr[u'_i + v'_i = \alpha, b'_{n+h(i)} = \beta]}, \text{ if } w \in \mathcal{D}'[i], \text{ but } w \notin \mathcal{D}[i] \end{cases} \\
&\qquad \text{where } \alpha = \Pi[i], \beta = \Pi[n + h(i)]
\end{aligned}
$$

Recall that for $i \leq n$, $u_i \sim \mathbf{Bern}(p), v_i \sim \mathbf{G}_0(1-q)$; for $i \leq |h|, u_{n+i} \sim \mathbf{B}(g(w, i), p), v_{n+i} \sim \mathbf{G}_0(1-q)$.

When $w \in \mathcal{D}[i]$, but $w \notin \mathcal{D}'[i]$, let $g(w, h(i)) = G$ where $G = counter_{max} - |\{j|h(j) = h(i), w \in \mathcal{D}[j]\}|$,

$$\frac{Pr[u_i + v_i = \alpha, b_{n+h(i)} = \beta]}{Pr[v'_i = \alpha, b'_{n+h(i)} = \beta} = \frac{Pr[u_i + v_i = \alpha]}{Pr[v'_i = \alpha]} \cdot \frac{Pr[b_{n+h(i)} = \beta]}{Pr[b'_{n+h(i)} = \beta]}$$

$$\frac{Pr[u_i + v_i = \alpha]}{Pr[v'_i = \alpha]} = \begin{cases} \dfrac{(1-p)(1-q)}{1-q} = 1-p, \text{ when } \alpha = 0; \\ \dfrac{pq^{\alpha-1}(1-q) + (1-p)q^{\alpha}(1-q)}{q^{\alpha}(1-q)} = \dfrac{p + (1-p)q}{q}, \text{ when } \alpha \geq 1. \end{cases} \tag{7.1}$$

Let $A_{s,t} = \sum\limits_{k=0}^{t} \binom{s}{k} p^k (1-p)^{s-k} q^{\beta-k}(1-q)$,

$$\frac{Pr[b_{n+h(i)} = \beta]}{Pr[b'_{n+h(i)} = \beta]} = \frac{\sum\limits_{k=0}^{min(G,\beta)} Pr[u_{n+h(i)} = k, v_{n+h(i)} = \beta - k]}{\sum\limits_{k=0}^{min(G+1,\beta)} Pr[u'_{n+h(i)} = k, v'_{n+h(i)} = \beta - k]} = \frac{A_{G,min(G,\beta)}}{A_{G+1,min(G+1,\beta)}}$$

$$= \frac{1}{1-p} \cdot \frac{\sum\limits_{k=0}^{min(G,\beta)} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{min(G+1,\beta)} \binom{G+1}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \leq \frac{1}{1-p} \tag{7.2}$$

since that $\dbinom{G}{k} \leq \dbinom{G+1}{k}$.

As a consequence,

$$\frac{Pr[u_i + v_i = \alpha, b_{n+h(i)} = \beta]}{Pr[v'_i = \alpha, b'_{n+h(i)} = \beta} \leq \frac{1}{1-p} \cdot \frac{p + (1-p)q}{q} \leq 1 + \frac{p}{q(1-p)}.$$

When $w \in \mathcal{D}'[i]$, but $w \notin \mathcal{D}[i]$, let $g'(w, h(i)) = G$ where $G = counter_{max} - |\{j|h(j) =$

$h(i), w \in \mathcal{D}[j]\}|$,

$$\frac{Pr[v_i = \alpha, b_{n+h(i)} = \beta]}{Pr[u_i' + v_i' = \alpha, b'_{n+h(i)} = \beta]} = \frac{Pr[v_i = \alpha]}{Pr[u_i' + v_i' = \alpha]} \cdot \frac{Pr[b_{n+h(i)} = \beta]}{Pr[b'_{n+h(i)} = \beta]}$$

$$\frac{Pr[v_i = \alpha]}{Pr[u_i' + v_i' = \alpha]} = \begin{cases} \dfrac{1-q}{(1-p)(1-q)} = \dfrac{1}{1-p}, \text{ when } \alpha = 0; \\ \dfrac{q^{\alpha}(1-q)}{pq^{\alpha-1}(1-q) + (1-p)q^{\alpha}(1-q)} = \dfrac{q}{p+(1-p)q}, \text{ when } \alpha \geq 1. \end{cases} \tag{7.3}$$

$$\frac{Pr[b_{n+h(i)} = \beta]}{Pr[b'_{n+h(i)} = \beta]} = \frac{A_{G+1,min(G+1,\beta)}}{A_{G,min(G,\beta)}}$$

When $\beta \leq G$,

$$\begin{aligned} \frac{A_{G+1,min(G+1,\beta)}}{A_{G,min(G,\beta)}} &= \frac{\sum\limits_{k=0}^{\beta} \binom{G+1}{k} p^k (1-p)^{G+1-k} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{\beta} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\ &= \frac{\sum\limits_{k=0}^{\beta} \left(\binom{G}{k} + \binom{G}{k-1}\right) p^k (1-p)^{G+1-k} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{\beta} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\ &= (1-p) + \frac{\sum\limits_{k=1}^{\beta} \binom{G}{k-1} p^k (1-p)^{G-(k-1)} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{\beta} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\ &= (1-p) + \frac{\sum\limits_{k=0}^{\beta-1} \binom{G}{k} p^{k+1} (1-p)^{G-k} q^{\beta-(k+1)}(1-q)}{\sum\limits_{k=0}^{\beta} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\ &< 1 - p + \frac{p}{q}; \end{aligned} \tag{7.4}$$

when $\beta \geq G+1$,

$$\begin{aligned}
\frac{A_{G+1,min(G+1,\beta)}}{A_{G,min(G,\beta)}} &= \frac{\sum\limits_{k=0}^{G+1} \binom{G+1}{k} p^k (1-p)^{G+1-k} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{G} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\
&= \frac{\sum\limits_{k=0}^{G+1} \left(\binom{G}{k} + \binom{G}{k-1}\right) p^k (1-p)^{G+1-k} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{G} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\
&= 1-p+\frac{\sum\limits_{k=1}^{G+1} \binom{G}{k-1} p^k (1-p)^{G-(k-1)} q^{\beta-k}(1-q)}{\sum\limits_{k=0}^{G} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\
&= 1-p+\frac{\sum\limits_{k=0}^{G} \binom{G}{k} p^{k+1} (1-p)^{G-k} q^{\beta-(k+1)}(1-q)}{\sum\limits_{k=0}^{G} \binom{G}{k} p^k (1-p)^{G-k} q^{\beta-k}(1-q)} \\
&= 1-p+\frac{p}{q}; \qquad (7.5)
\end{aligned}$$

As a consequence,

$$\frac{Pr[v_i = \alpha, b_{n+h(i)} = \beta]}{Pr[u'_i + v'_i = \alpha, b'_{n+h(i)} = \beta} \leq \frac{1}{1-p} \cdot \left(1-p+\frac{p}{q}\right) = 1 + \frac{p}{q(1-p)}.$$

Therefore, we obtain

$$\frac{Pr[\mathcal{M}(\mathcal{D}, \vec{w}) \in S]}{Pr[\mathcal{M}(\mathcal{D}', \vec{w}) \in S]} \leq \left(1+\frac{p}{q(1-p)}\right)^{|\Lambda|} \leq \left(1+\frac{p}{q(1-p)}\right)^{|\vec{w}|}.$$

Differential Privacy for Keywords. Consider a database $\mathcal{D}$, and a pair of neighbouring keyword lists $\vec{w}, \vec{w}' \in \Delta^{|\vec{w}|}$ where $\vec{w}[i] \neq \vec{w}'[i]$. Let w be $\vec{w}[i]$, w' be $\vec{w}'[i]$ and $\Lambda = \{j \,|\, \hat{\Pi}_w[j] \neq \hat{\Pi}_{w'}[j]\}$ where $\hat{\Pi}_w$ is the real access pattern while searching for w. Define $h(\Lambda) = \{n +$

$h(j)|j \in \Lambda\}$. Let U be a list and define $U_\Lambda = (U_{\Lambda[1]}, U_{\Lambda[2]}, ..., U_{\Lambda[|\Lambda|]})$ where $\Lambda[j]$ represents for the j^{th} smallest element in Λ.

$$\begin{aligned}\frac{Pr[\mathcal{M}(\mathcal{D},\vec{w}) \in S]}{Pr[\mathcal{M}(\mathcal{D},\vec{w}') \in S]} = \frac{Pr[\mathcal{M}(\mathcal{D},\vec{w}[i]) \in S]}{Pr[\mathcal{M}(\mathcal{D},\vec{w}'[i]) \in S]} &= \frac{Pr[\mathcal{M}(\mathcal{D},\vec{w}[i]) = \Pi]}{Pr[\mathcal{M}(\mathcal{D},\vec{w}'[i]) = \Pi]} \\ &= \frac{Pr[\mathcal{M}(\mathcal{D},\vec{w}[i])_{\Lambda\cup h(\Lambda)} = \Pi_{\Lambda\cup h(\Lambda)}]}{Pr[\mathcal{M}(\mathcal{D},\vec{w}'[i])_{\Lambda\cup h(\Lambda)} = \Pi_{\Lambda\cup h(\Lambda)})]} \\ &= \prod_{j\in\Lambda\cup h(\Lambda)} \frac{Pr[\Pi_w[j] = \Pi[j]]}{Pr[\Pi_{w'}[j] = \Pi[j]]}\end{aligned}$$

Let $\eta = |\{j \,|\, 1 = \hat{\Pi}_w[j] > \hat{\Pi}_{w'}[j], j \le n\}|$. In other words, η is the number of documents which contain w but not w'. Then the number of documents which contain w' but not w is $\rho = |\Lambda| - \eta$. Let $\eta_t = |\{j \mid 1 = \hat{\Pi}_w[j] > \hat{\Pi}_{w'}[j]$ and $h(j) = t, j \le n\}|$ and $\rho_t = |\{j \mid 1 = \hat{\Pi}'_w[j] > \hat{\Pi}_w[j]$ and $h(j) = t, j \le n\}|$, namely η_t and ρ_t are the number of documents which contain w but not w' and the number of documents which contain w' but not w, respectively. Therefore, $\eta = \sum_{j\in h(\Lambda)} \eta_j$ and $\rho = \sum_{j\in h(\Lambda)} \rho_j$.

When $j \in \Lambda$, based on what have been proved in equation (7.1), (7.3),

$$\frac{Pr[\Pi_w[j] = \Pi[j]]}{Pr[\Pi_{w'}[j] = \Pi[j]]} = \begin{cases} \dfrac{Pr[u_i + v_i = \Pi[j]]}{Pr[v_i = \Pi[j]]} \le \dfrac{p + (1-p)q}{q}, \text{ if } w \in \mathcal{D}[j] \text{ but } w' \notin \mathcal{D}[j]; \\ \\ \dfrac{Pr[v_i = \Pi[j]]}{Pr[u_i + v_i = \Pi[j]]} \le \dfrac{1}{1-p}, \text{ if } w \notin \mathcal{D}[j] \text{ but } w' \in \mathcal{D}[j]; \end{cases}$$

Therefore,

$$\prod_{j\in\Lambda} \frac{Pr[\Pi_w[j] = \Pi[j]]}{Pr[\Pi_{w'}[j] = \Pi[j]]} = \left(\frac{p + q(1-p)}{q}\right)^\eta \cdot \left(\frac{1}{1-p}\right)^\rho \tag{7.6}$$

When $j \in h(\Lambda)$, let $\beta = \Pi[n+j]$ and $A_{s,t} = \sum_{k=0}^{t} \binom{s}{k} p^k (1-p)^{s-k} q^{\beta-k}(1-q)$. Let $g(w,j) = G$ where $G = counter_{max} - |\{t|h(t) = j, w \in \mathcal{D}[t]\}|$ and $g(w',j) = G + \eta_j - \rho_j$.

$$\frac{Pr[\Pi_w[n+j] = \Pi[n+j]]}{Pr[\Pi_{w'}[n+j] = \Pi[n+j]]} = \frac{Pr[b_{n+j} = \beta]}{Pr[b'_{n+j} = \beta]} = \frac{A_{G,min(G,\beta)}}{A_{G+\eta-\rho,min(G+\eta_j-\rho_j,\beta)}}$$

When $\eta_j \geq \rho_j$, then $G \leq G + \eta_j - \rho_j$.

$$\frac{A_{G,min(G,\beta)}}{A_{G+\eta_j-\rho_j,min(G+\eta_j-\rho_j,\beta)}} = \prod_{j=1}^{\eta_j-\rho_j} \frac{A_{G+j-1,min(G+j-1,\beta)}}{A_{G+j,min(G+j,\beta)}} \leq \left(\frac{1}{1-p}\right)^{\eta_j-\rho_j}. \tag{7.7}$$

When $\eta_j < \rho_j$, then $G \geq G + \eta_j - \rho_j$.

$$\frac{A_{G,min(G,\beta)}}{A_{G+\eta_j-\rho_j,min(G+\eta_j-\rho_j,\beta)}} = \prod_{j=1}^{\rho_j-\eta_j} \frac{A_{G-j+1,min(G-j+1,\beta)}}{A_{G-j,min(G-j,\beta)}} \leq \left(1-p+\frac{p}{q}\right)^{\rho_j-\eta_j}. \tag{7.8}$$

The last inequation holds is because of what has been proved in equation (7.4), (7.5) in the part of proving differential privacy for documents. Therefore,

$$\prod_{j\in h(\Lambda)} \frac{Pr[\Pi_w[n+j] = \Pi[n+j]]}{Pr[\Pi_{w'}[n+j] = \Pi[n+j]]} = \left(\frac{1}{1-p}\right)^{\hat{\eta}-\hat{\rho}} \cdot \left(1-p+\frac{p}{q}\right)^{\rho-\hat{\rho}-(\eta-\hat{\eta})} \tag{7.9}$$

where $\hat{\eta} = \sum\limits_{j\in\Lambda,\eta_j\geq\rho_j} \eta_j$ and $\hat{\rho} = \sum\limits_{j\in\Lambda,\eta_j\geq\rho_j} \rho_j$.

Combining equation (7.6), (7.9), we can obtain that

$$\begin{aligned}
\prod_{j\in\Lambda} \frac{Pr[\Pi_w[j] = \Pi[j]]}{Pr[\Pi_{w'}[j] = \Pi[j]]} &\leq \left(\frac{p+(1-p)q}{q}\right)^{\eta} \cdot \left(\frac{1}{1-p}\right)^{\rho} \cdot \left(\frac{1}{1-p}\right)^{\hat{\eta}-\hat{\rho}} \cdot \left(1-p+\frac{p}{q}\right)^{\rho-\hat{\rho}-(\eta-\hat{\eta})} \\
&= \left(1-p+\frac{p}{q}\right)^{\rho-\hat{\rho}+\hat{\eta}} \cdot \left(\frac{1}{1-p}\right)^{\hat{\eta}-\hat{\rho}+\rho} = \left(1+\frac{p}{q(1-p)}\right)^{\rho-\hat{\rho}+\hat{\eta}} \\
&\leq \left(1+\frac{p}{q(1-p)}\right)^{\rho+\eta} = \left(1+\frac{p}{q(1-p)}\right)^{d(\mathcal{D}(\vec{w}),\mathcal{D}(\vec{w}'))}.
\end{aligned}$$

Therefore,

$$\frac{Pr[\mathcal{M}(\mathcal{D},\vec{w}) \in S]}{Pr[\mathcal{M}(\mathcal{D},\vec{w}') \in S]} = \prod_{j\in\Lambda} \frac{Pr[\Pi_w[j] = \Pi[j]]}{Pr[\Pi_{w'}[j] = \Pi[j]]} \leq \left(1+\frac{p}{q(1-p)}\right)^{d(\mathcal{D}(\vec{w}),\mathcal{D}(\vec{w}'))}.$$

where $d(\mathcal{D}(\vec{w}),\mathcal{D}(\vec{w}')) = |\{j \mid \hat{\Pi}_w[j] \neq \hat{\Pi}_{w'}[j]\}| = \eta + \rho$.

Utility. The true positive rate and false positive rate of DP-SSE are $(p+q-pq)$ and q, respectively. Therefore, the utility of DP-SSE is $(p+1-pq)(1-q)$. □

Using the same proof strategy as above, we can prove the follow theorems.

Theorem 7.0.2. DP-SSE-1 is a (u, ϵ, ϵ)-differentially private SSE where $u = (p + q - pq)(1 - q)$ and $\epsilon = \ln\left(1 + \frac{p}{q(1-p)}\right)$.

Theorem 7.0.3. DP-SSE-2 is a (u, ϵ, ϵ)-differentially private SSE where $u = (p + q - pq)(1 - q)$ and $\epsilon = \ln\left(1 + \frac{p}{q(1-p)}\right)$.

However, DP-SSE-3 is slightly different from the other two schemes since the output of the label distribution function depends on the content of the document. This does not affect the analysis in differential privacy for keywords. However, in the analysis of differential privacy for documents, removing or adding a single keyword w to one document $\mathcal{D}[i]$ will affect at most C_{max} search keys. Therefore, we have the following theorem.

Theorem 7.0.4. DP-SSE-3 is a $(u, \epsilon_1, \epsilon_2)$-differentially private SSE where $u = (1 - (1 - p)(1 - q)^2)(1 - (1 - q)^2)$, $\epsilon_1 = \ln\left(1 + \frac{p}{q(1-p)}\right)$, $\epsilon_2 = 2C_{max} \cdot \ln\left(1 + \frac{p}{q(1-p)}\right)$.

Proof. Refer to Appendix A.6. □

Chapter 8

Complexity

In this chapter, we will discuss the complexity of DP-SSE, namely communication complexity and computation complexity. Note that the initialization is not discussed here, since it can be done efficiently in practice. After the analysis of the meta-scheme DP-SSE, the communication complexity and computation complexity of DP-SSE-1, DP-SSE-2 and DP-SSE-3 can be obtained by replacing the *hyper parameters*(like $counter_{max}$ and $|h|$) with corresponding values.

Communication complexity refers to the total network traffic between the client and the server during the query process for one keyword. It has two parts: the network traffic from client to server, i.e. all tokens, and the network traffic from server to client, i.e. all matched documents. Computation complexity refers to the complexity of the function `Trapdoor` and the function `Search` which consume the most computation resources.

8.1 Communication Complexity

When searching for keyword w, the client will generate a list of tokens instead of one and send them to the server. These tokens are originated from three sources: `TruePosSet`, `FalsePosSet`, and `NonMatchSet`. The expected size of them is $|h| \cdot counter_{max} \cdot p$, $n \cdot \sum_{i=0}^{\infty} iq^i(1-q) = \frac{nq}{1-q}$, and $|h| \cdot \frac{q}{1-q}$, respectively. Therefore, the expected number of tokens sent from client to server, denoted by E^{client}, is

$$\begin{aligned} E^{client} &= |h| \cdot counter_{max} \cdot p + \frac{nq}{1-q} + |h| \cdot \frac{q}{1-q} \\ &= O(|h| \cdot counter_{max}), \text{ assuming } nq = O(C_{max}). \end{aligned} \tag{8.1}$$

It should be noted that $|h| \cdot counter_{max} \geq C_{max}$. It should also be noted that the number of issued tokens is independent from the keyword being searched.

The other part of *communication complexity* comes from network traffic sent from server to client, i.e. the matched documents. When searching for keyword w, there would be two types of documents returned (a document would be returned if it is matched at least once), i.e. true positives and false positives. The expected numbers of them are $|\mathcal{D}(w)| \cdot (p+q-pq)$ and $(n - |\mathcal{D}(w)|)q$, respectively. Therefore, the expected number of document returned in total when searching for w, denoted by E_w^{server}, would be

$$\begin{aligned} E_w^{server} &= |\mathcal{D}(w)| \cdot (p+q-pq) + (n - |\mathcal{D}(w)|)q \\ &\leq C_{max} + C_{max} = O(C_{max}) \end{aligned} \tag{8.2}$$

It should be noted that the above bound for E_w^{server} also holds in big-O notation for DP-SSE-3 even the true positive rate and false positive rate change in DP-SSE-3.

Note that the size of each document is not necessarily equal. To simplify the analysis, we assume all documents are of the same size S_{max}. Therefore, the expected traffic volume when searching for w is

$$V_{DPSSE} = (E_w^{server} + E^{client}) \cdot S_{max} = O(|h| \cdot counter_{max} \cdot S_{max}). \tag{8.3}$$

Note that V_{DPSSE} is independent of the queried w in big-O notation.

In order to compute the communication overhead of DP-SSE, we need to make assumptions on the distributions of keywords and queries. Here we give our analysis assuming keywords follow uniform distribution and *Zipfian* distribution, respectively.

As for a traditional searchable encryption scheme, it should be able to return all true positive documents in order to provide high utility, so there is a lower bound of communication complexity in SSE.

8.1.1 Keywords Following Uniform Distribution

If keywords are uniformly distributed in the database, i.e. $|\mathcal{D}(w_i)| = |\mathcal{D}(w_j)| = C_{max}$, the communication volume in an SSE when searching for keyword w is

$$V_{SSE}^{u} = |\mathcal{D}(w)| \cdot S_{max} = C_{max} \cdot S_{max} \tag{8.4}$$

Note that V_{SSE}^{u} is independent of w. Therefore, we can compute the communication overhead of DP-SSE.

Theorem 8.1.1. DP-SSE has $O\left(\frac{|h| \cdot counter_{max}}{C_{max}}\right)$ (amortized) communication overhead when keywords follow the uniform distribution.

Proof.

$$\frac{V_{DPSSE}}{V_{SSE}^{u}} = \frac{O(|h| \cdot counter_{max} \cdot S_{max})}{C_{max} \cdot S_{max}} = O\left(\frac{|h| \cdot counter_{max}}{C_{max}}\right) \tag{8.5}$$

□

It should be noted that the exact value of $counter_{max}$ and $|h|$ both depend on the label distribution function, i.e. h.

8.1.2 Keywords Following Zipfian Distribution

Zipf's law states that the frequency of an individual word in a corpus of natural language utterances is inversely proportional to its rank (the position of it in a sorted list in decreasing order of frequency) [33]. Let $w(i)$ be the i^{th} most frequent keyword in Δ. Δ is said to follow Zipfian distribution if the frequency $R_{w(i)}$ of the keyword $w(i)$ of Δ satisfies

$$\frac{R_{w(i)}}{R_{w(j)}} = \frac{j}{i}.$$

Due to the fact that $R_{w(1)} = C_{max}$, we have

$$R_{w(i)} = \frac{C_{max}}{i}, \forall i \in [|\Delta|]\,. \tag{8.6}$$

Note. Usually some stopwords like *the, is, and, etc.* in Δ would be removed when building the index. In this case C_{max} would be the frequency of the $(t+1)^{th}$ keyword (assuming there are t such stopwords) resulting that $R_{w(i)} = \frac{t+1}{i} C_{max}, \forall t+1 \leq i \leq |\Delta|$. However, such difference will not change the following approximation of communication complexity or overhead. Therefore, we use $R_{w(i)} = C_{max}/i$ for simplicity in the following.

To approximate the communication complexity of an SSE, we also need to assume some query distribution. Here we give our analysis when assuming queries are following *Zipfian* distribution. Let e_i be the event that $w(i)$ is queried, queries are said to follow Zipfian distribution if

$$Pr[e_i] = \frac{\frac{1}{i}}{N_{|\Delta|}}, \text{where } N_{|\Delta|} = \sum_{j=1}^{|\Delta|} \frac{1}{j}. \tag{8.7}$$

The assumption that queries also follow Zipfian distribution suggests that the more frequent a keyword appears, the more likely it would be queried.

Theorem 8.1.2. DP-SSE has $O\left(\log|\Delta| \cdot \frac{|h| \cdot counter_{max}}{C_{max}}\right)$ amortized communication overhead when keywords and queries both follow the Zipfian distribution.

Proof. Define $w(i)$ and e_i as above.

In order to search for a keyword, the (average) communication volume in an SSE (when retrieving them in plaintext) would be:

$$\begin{aligned} V^z_{SSE} &= \sum_{i=1}^{|\Delta|} Pr[e_i] \cdot |\mathcal{D}(w(i))| \cdot S_{max} \\ &= \sum_{i=1}^{|\Delta|} \frac{1/i}{N_{|\Delta|}} \cdot \frac{C_{max}}{i} \cdot S_{max} \text{ (refer to (8.6), (8.7))} \\ &= O\left(\frac{C_{max} \cdot S_{max}}{\log|\Delta|}\right). \end{aligned} \tag{8.8}$$

due to the fact that

$$\sum_{i=1}^{|\Delta|} \frac{1}{i} = O(\log|\Delta|) \text{ and } \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}.$$

Thus, the communication overhead of DP-SSE would be

$$\begin{aligned}\frac{V_{DPSSE}}{V^z_{SSE}} &= \frac{O(|h| \cdot counter_{max} \cdot S_{max})}{O\left(\dfrac{C_{max} \cdot S_{max}}{\log |\Delta|}\right)} \\ &= O\left(\log |\Delta| \cdot \frac{|h| \cdot counter_{max}}{C_{max}}\right). \end{aligned} \tag{8.9}$$

□

Note: In the above analysis, the size of one document is treated the same as the size of its keyword list. In the case where the size of a document is much larger than its keyword list, the communication overhead of DP-SSE would be smaller. The reason is that communication overhead comes mostly from query tokens in DP-SSE which is of the same size as keyword list. If every document is $\frac{|h| \cdot counter_{max}}{C_{max}}$ times larger than its keyword list, the above amortized overhead could decrease to $O(\log |\Delta|) = O(\log n)$.

8.2 Computation Complexity

In this section, we focus on the efforts that the server needs to compute the query results given the tokens searching for keyword w. Specifically, we use the number of calls for function `FHIPPE.Query` to measure the computation complexity. It worth mentioning that each call for `FHIPPE.Query` will take $O(S_{max})$ time.

DP-SSE reveals the label distribution and the labels of the tokens. Then the server only needs to call `FHIPPE.Query` for each pair of search key and token which share the same label. For every label, there are approximately $O(counter_{max})$ query tokens sharing such label. Therefore, the computation complexity for DP-SSE is $O(n \cdot counter_{max})$ where n is the number of documents.

8.3 Complexity and Label Distribution Function h

From the previous analysis, the communication overhead of DP-SSE is either $O\left(\frac{|h| \cdot counter_{max}}{C_{max}}\right)$ or $O\left(\log n \cdot \frac{|h| \cdot counter_{max}}{C_{max}}\right)$ depending on the keyword and query distribution and the computation complexity of DP-SSE is $O(n \cdot counter_{max})$. Therefore, we can compute the complexity of DP-SSE-1, DP-SSE-2 and DP-SSE-3 as shown in Table 8.1. Note that each scheme has two values in communication overhead cell where the first value is in the setting of uniformly distributed keywords while the second one is in the setting where keywords and queries both follow the Zipfian distribution.

Scheme	$\lvert h\rvert$	$counter_{max}$	Communication Overhead		Computation Complexity
			Uniform	Zipfian	
DP-SSE-1	1	C_{max}	$O(1)$	$O(\log n)$	$O(n^2)$
DP-SSE-2	C_{max}	$O\left(\frac{\log n}{\log\log n}\right)$	$O\left(\frac{\log n}{\log\log n}\right)$	$O\left(\frac{\log^2 n}{\log\log n}\right)$	$O\left(n \cdot \frac{\log n}{\log\log n}\right)$
DP-SSE-3	C_{max}	$O(\log\log n)$	$O(\log\log n)$	$O(\log n \cdot \log\log n)$	$O(n \cdot \log\log n)$

Table 8.1: Complexity of DP-SSE-1, DP-SSE-2 and DP-SSE-3

DP-SSE-1 has the least communication overhead but the highest computation complexity. DP-SSE-2 reduces the computation complexity to $O\left(n \cdot \frac{\log n}{\log\log n}\right)$ by utilizing a hash function at the expense of increasing the communication overhead by a factor of $O\left(\frac{\log n}{\log\log n}\right)$. DP-SSE-3 further reduces the computation complexity to $O(n \cdot \log\log n)$ compared to DP-SSE-2. It also reduces the communication overhead but at the expense of decreasing the privacy level compared to DP-SSE-2 as pointed out in Theorem 7.0.4.

As what can be seen from Table 8.1, the label distribution function h can influence the communication and computation complexity. It is not hard to see that when $|h|$ decreases, $counter_{max}$ will remain or increase and when $counter_{max}$ decreases the computation complexity decreases no matter what $|h|$ is. To make DP-SSE computationally efficient,

$counter_{max}$ should not be too large. In other words, $|h|$ should not be too small. There might also be some other options of h where $|h|$ is not too small besides the candidates introduced in 5.4. In general, given a not-too-small $|h|$, we can formalize the problem of finding the best assignment of labels to documents in DP-SSE such that *$counter_{max}$* is minimized as a *defective vertex coloring problem*. Unfortunately, the defective vertex coloring problem is proved to be *NP-complete*[7].

Theorem 8.3.1. Given $|h| = c \geq 3$, finding the best assignment of labels to documents such that *$counter_{max}$* is minimized is NP-hard.

Proof. This problem can be formalized as follows:
Given n $(0, 1)$-binary vectors $\{X_1, X_2, ..., X_n\}$ of length S_{max}, group them into c disjoint groups, $\{g_1, g_2, ..., g_c\}$ such that A is minimized over i, h where

$$A = \max_{i,h} \|(\sum_{g_i} X_j)\|_\infty$$

where h is the grouping strategy and $\|X_j\|_\infty = \max_i |X_j[i]|$. Call the decision version of this problem FP, namely $\langle X, c, A\rangle$ is said to be in FP if and only if there exists a grouping strategy such that X can be grouped into c groups and no group has sum whose maximum norm is greater than A.
Let COLOR $= \{\langle G, k, d\rangle \ |$ G is (k,d)-colorable$\}$ where G is (k,d)-colorable means one can color all vertices of G with at most k colors such that no vertex is adjacent to more than d vertices of the same color. Now we reduce COLOR to FP. Assume $G = \langle V, E\rangle$ where $|V| = n$ and $|E| = S_{max}$,

- Every vertex $v_i \in V$ is mapped to a vector X_i and set $X = \{X_1, X_2, ..., X_n\}$;
- Every edge $e_j \in V$ is mapped to a number j;
- In a vector X_i, set $X_i[j] = 1$ if edge e_j is incident on the vertex v_i; otherwise set $X_i[j] = 0$;
- c = k;
- A = d;

We now claim that $\langle G, k, d \rangle \in$ Color if and only if $\langle X, c, A \rangle \in$ Fp. As pointed by [7], in general graphs, the (k, d)-coloring problem is NP-complete for $k \geq 3, d \geq 0$. Therefore, given $|h| \geq 3$, the problem of finding the best assignment of labels to documents such that $counter_{max}$ is minimized is NP-hard. □

Chapter 9

Evaluation

In this chapter, we will evaluate the effectiveness and efficiency of our proposed differentially private SSE. For effectiveness, we empirically show that DP-SSE can resist access pattern attacks, i.e. the IKK attack. We also show that the search pattern is hard to recover by evaluating a clustering algorithm, KMeans clustering. Our experimental results suggest that the accuracy for both the IKK attack and Kmeans clustering are less than 20% (baseline 15%) and 17% (baseline 5%), respectively, even when only a small fraction of false positive documents are added. For efficiency, we report the running time when running DP-SSE-3 as a representative on *Enron* dataset in an Intel(R) E7-8870 160-core Ubuntu 16.04 machine clocked at 2.40 GHz with 2 TB of system memory. The results suggest that one can perform one search in 25 minutes while utilizing 128 cores in parallel.

The rest of this chapter is organized as follows. We first describe the experiment settings, then describe the experiments and explain their results, and finally show the running time of DP-SSE-3.

9.1 Experiment Settings

Dataset Used and Keyword Generation. We use Enron email dataset for our following experiments. The Enron email dataset has 30109 emails in *_sent* directory which we considered as our document collection. We conduct the same keyword extraction process as in the IKK paper [14] and then we remove the 200 most common keywords and use the x most common keywords remaining as keyword universe. We vary the value of x in our experiments.

Query Generation. It is hard to describe the query distribution even when the whole dataset is available since search habits of users are expected to vary significantly. Therefore, we use the *Zipfian distribution* as in [14, 5] to sample queries from the keyword universe when conducting access pattern attacks in order to maintain comparability. While conducting search pattern clustering attacks, we use the *Zipfian distribution* and the *uniform distribution* to sample queries from the keyword universe in order to show the general effectiveness of the clustering algorithm, since this algorithm is expected to be sensitive to query distributions.

Unified Access Patterns. Recall that the access pattern in DP-SSE is a multinary vector of length $n + |h|$ denoted by $\Pi = (b_1, ..., b_n, b_{n+1}, ..., b_{n+|h|})$, while the access pattern of a traditional SSE is a binary vector of length n. To apply *IKK*-like attacks, we need to unify these two types of access patterns. Since performing an *IKK*-like attack on a multinary vector is non-trivial, we map Π into a length-n binary vector $\Pi' = (b'_1, ..., b'_n)$ where $b'_i = 0$ if $b_i = 0$ otherwise $b'_i = 1$ for $i \leq n$.

9.2 Hiding Access Pattern

We replicated the IKK attack on Enron email dataset as in the IKK paper [14] with a keyword universe size x of 500, 1000, 1500, 2000 and 2500.

Unlike in the original IKK attack setting where the queries are all unique, we allow repeated queries. In each keyword universe setting, we generate 200 queries to be issued. 15% of them are known to the attacker, which are chosen from all queries uniformly at random. We also modifies the IKK attack to adapt to the new setting where queries can be repeated. The modified IKK denoted by mIKK is detailed in Appendix B.2.

We compare the attack resistance of DP-SSE to this attack with a traditional SSE scheme (denoted by SSE) and the mechanism in [5] (denoted by OSSE) in their *iIKK* setting where the attacker knows the shards distribution. The true positive rate and false positive rate are set to be 99.99% and 2%, respectively. In other words, the utility is fixed as $0.9999(1 - 0.02) \approx 0.98$. In each keyword universe setting, we run each attack 20 times for long enough and take the average recovery rate.

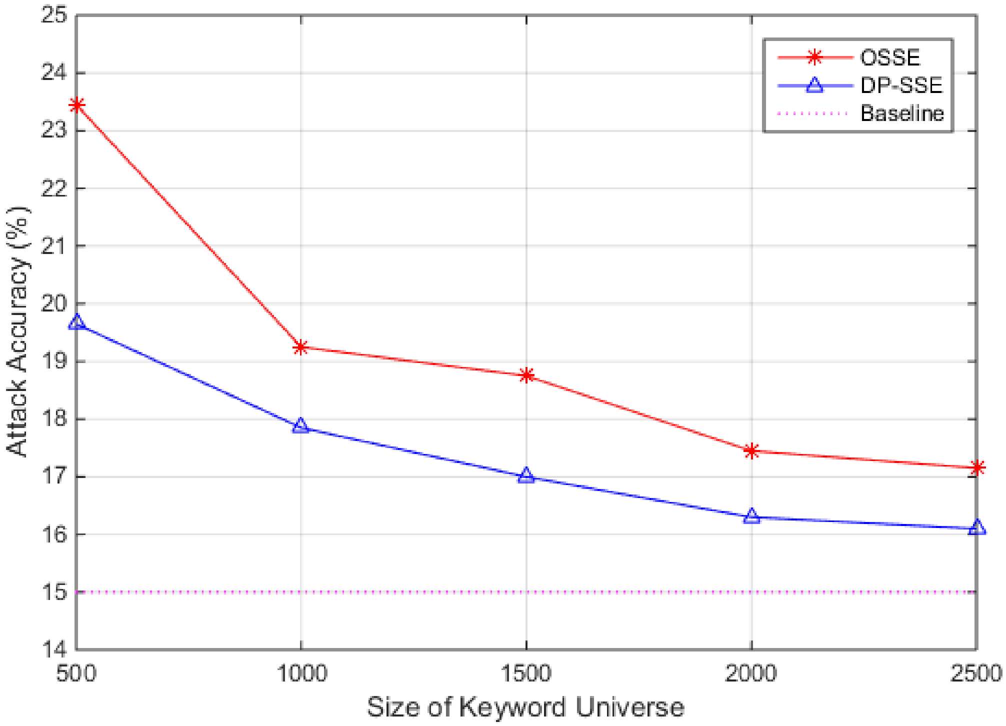


Figure 9.1: Access Pattern Attack Accuracy Varying Keyword Universe Size

Figure 9.1 displays the query recovery rate versus the keyword universe size. The query recovery rate for SSE, which leaks access patterns, is 97% when the size of keyword universe is set to 500. The query recovery rate goes down to 60.8% steadily as the keyword universe size gets larger. This is because the search space becomes larger which makes it harder for the IKK algorithm (a heuristic algorithm) to find a good solution. After applying DP-SSE, the query recovery rate drops sharply to 19.5% from 97%. Since 15% of queries are prior knowledge indicated by baseline, only 4.5% more queries are actually recovered. Although OSSE [5] also greatly decreases the query recovery rate (from 97%

to 23.5%), it is still slightly worse than DP-SSE for all universe sizes. The advantage of DP-SSE over OSSE in resisting the attack comes from two parts. First, DP-SSE provides independent randomness per query, i.e. two queries searching for the same keyword will almost always get different results. Therefore, the attacker cannot easily recover queries which are searching for keywords previously known to the attacker. Second, the mIKK attack to DP-SSE has to search in a larger space for good solutions. Perfectly conducting the IKK attack needs solving an NP-complete problem, which is infeasible. Thus, usually a heuristic solution is taken to look for good but maybe suboptimal solutions. The solution space in DP-SSE is $|\Delta|^{|\Delta|}$ which is much larger than $|\Delta|!$, the size of the solution space in the other setting.

9.3 Hiding Search Pattern

In order to show DP-SSE can actually hide search patterns, we run a clustering algorithm (KMeans) to cluster queries based on their access patterns. If the queries cannot be well-clustered, it means that it is difficult to determine whether a set of queries are searching for the same keywords or not, namely the search patterns are hidden.

The set to be clustered is composed of unified access patterns, namely (0,1)-binary vectors of length n where n is the number of documents. The size of the dataset is limited to 500, namely 500 binary vectors per set. The underlying keywords per dataset are sampled from a keyword population of size 25 based on some distribution (Zipfian or uniform distribution). The keyword population contains either 25 consecutive keywords in a sorted keyword universe list in decreasing order of frequency or 25 randomly chosen keywords in Δ. Let $\Delta_{i \to j}$ be a keyword population of size $j - i$ which contains $w(t)$ for $t \in (i, j]$ where $w(t)$ is the t^{th} most frequent keyword in Δ.

We run *KMeans* clustering algorithm for 15 times per access pattern set and report the average clustering accuracy. The clustering accuracy is measured by the ratio of the number of vectors that are correctly clustered v.s. the total number of vectors. By correctly clustered, we mean that a vector is generated by searching for the same keyword as the label of its cluster. The label of a cluster is the keyword which generates the most vectors in it.

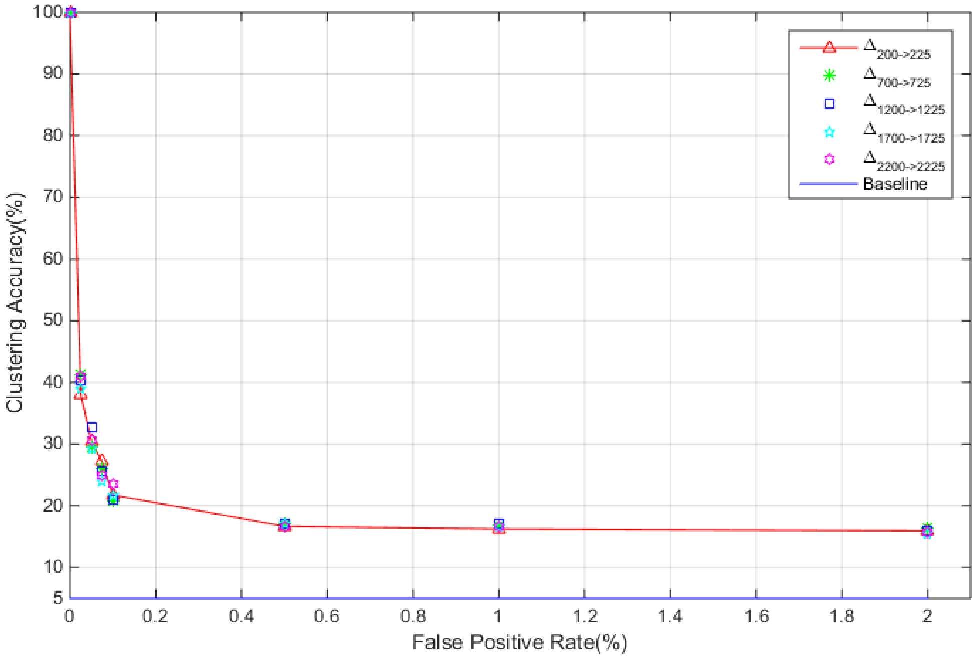


Figure 9.2: Clustering Accuracy, Queries Are Sampled From Uniform Distribution

Figure 9.2 and figure 9.3 show the clustering accuracy for different keyword populations when varying the false positive rate when queries are sampled from a Zipfian distribution and a uniform distribution, respectively. The true positive rate is fixed as 99.99%. The baseline represents the clustering accuracy for random guessing. It should be noted that the mechanism in [5] does not hide search patterns and simply running our clustering algorithm on that mechanism has an accuracy of approximately 100%. While for our scheme, DP-SSE, the clustering accuracy decreases sharply when increasing the false positive rate from 0 to 0.1%, then the accuracy decreases smoothly when further increasing the false positive rate. When the false positive rate is larger than 0.5%, the clustering accuracy becomes less than 17% which is sufficiently low to hide search patterns. While focusing on each false positive rate, where the keyword populations are sampled seem to have no clear impact on clustering accuracy, namely queries are all hard to cluster no matter whether they are generated from

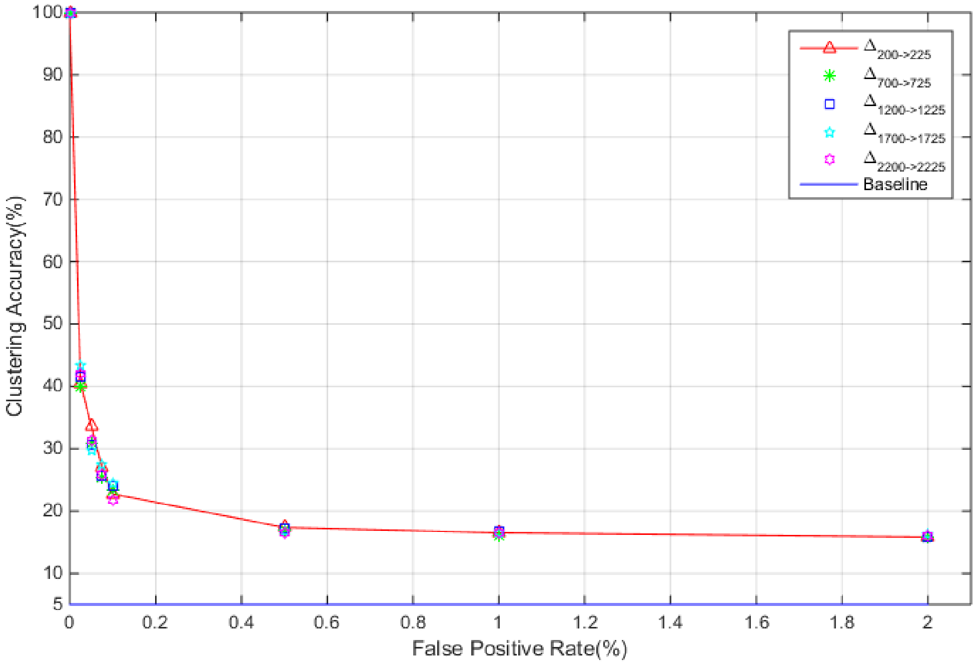


Figure 9.3: Clustering Accuracy, Queries Are Sampled From Zipfian Distribution

frequent or relatively non-frequent keywords. Comparing figure 9.2 to figure 9.3, it can be seen that how queries are sampled does not influence the clustering accuracy significantly. The reason might be the sample size is too small to show the potential impact of query distributions.

Figure 9.4 shows the clustering accuracy when the keyword population is sampled uniformly at random from a large keyword universe of size 2500. It should be noted that such keyword population might contain both frequent and non-frequent keywords whose real access patterns are quite different and easy to distinguish. However, the experimental results show that the differentially private access patterns for such keywords are still hard to separate.

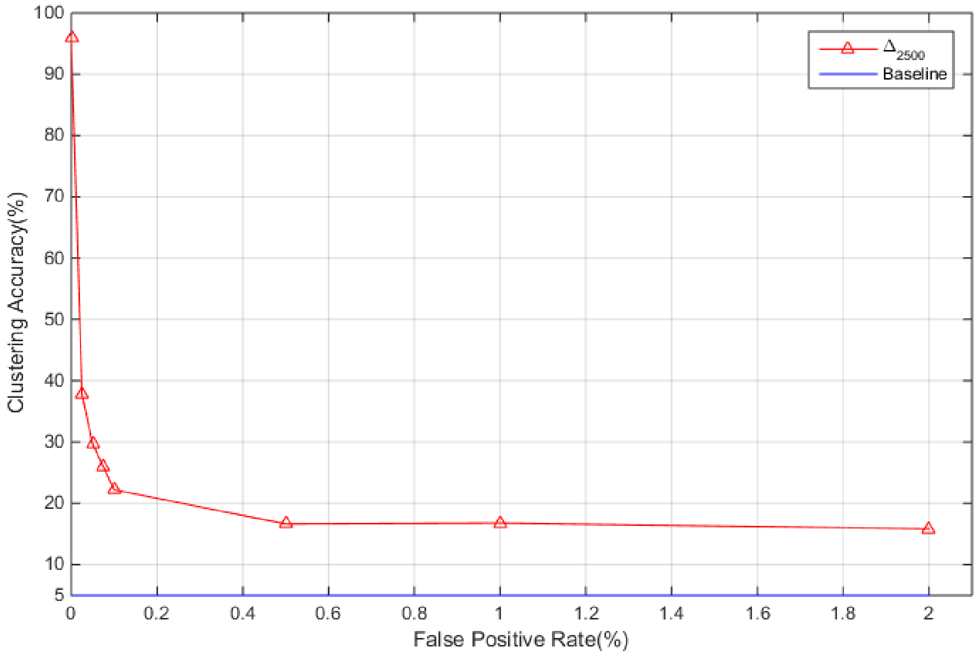


Figure 9.4: Clustering Accuracy, Keyword Population are Randomly Chosen from $\Delta_{0\to2500}$.

9.4 Running Time

Here we report the running time to build and run DP-SSE-3 on *Enron* dataset including the running time of `Keygen, BuildIndex, Trapdoor` and `Search` which compose DP-SSE-3. Here we assume the index of the database is already generated and the content of documents is already encrypted under some symmetric encryption scheme since these are all common operations for almost all SSE schemes. In all of the following experiments, we set p and q be 0.9999 and 0.01, respectively. In other words, the true positive rate is set to be $1-(1-p)(1-q)^2 \approx 0.9999$ and the false positive rate is set to be $1-(1-q)^2 \approx 0.02$.

In our implementation, we use the function-hiding inner product encryption scheme (denoted by IPE) proposed in [19] to implement our function-hiding inner product predicate encryption scheme FHIPPE (refer to Appendix B.1 for details). We limit the number of keywords for each document to be no more than 300 by splitting large documents into smaller ones. It should be noted that more than 97% of documents have no more than 300 keywords. After splitting, the number of documents increases by 1.5% to 30562, i.e. $n = 30562$ and C_{max} becomes approximate 2000. By using 2 hash functions, we manage to limit $counter_{max}$ to 3.

Keygen	FHIPPE.Encrypt	FHIPPE.GenToken	FHIPPE.Query
17.1 hours	1.77s	0.28s	1.25s

Table 9.1: Running Time

# cores	BuildIndex(min)	Trapdoor(s)	Search(min)
4	272.5	580.7	933.1
8	136.3	290.5	463.4
16	68.2	145.3	235.21
32	34.1	72.8	119.9
64	17.1	36.4	62.2
128	8.5	18.2	34.0
160	6.9	14.7	27.9

Table 9.2: Running Time in Parallel

Table 9.1 displays the running time for functions Keygen, FHIPPE.Encrypt, FHIPPE.GenToken and FHIPPE.Query. Due to their computation nature, such functions cannot be trivially parallelized. It should be noted that although Keygen takes long to run, it only needs to run once in the setup phase. Recall that in section 5.2, BuildIndex, Trapdoor and Search are constructed by constantly invoking Encrypt, GenToken, Query in FHIPPE, respectively. Such property makes it easy to parallelize these functions. It should be noted that BuildIndex only needs to run once in setup phase. For each keyword search, Trapdoor and Search are both invoked once. In each invocation of Trapdoor, FHIPPE.GenToken is invoked approximately $C_{max} \cdot counter_{max} \cdot p + 2 \cdot q \cdot n + q \cdot C_{max} \approx 6630$ times. Since each invocation of FHIPPE.GenToken takes about 0.35s, the function Trapdoor takes about 38.5 minutes while running in single CPU core. In each invocation of Search, FHIPPE.Query would be invoked approximately $2 \cdot n \cdot (counter_{max} \cdot p + q) \approx 183965$ times. Since each invocation of FHIPPE.Query takes about 1.25s, the function `Search` takes about 64 hours in single thread.

Table 9.2 shows the running time for `BuildIndex, Trapdoor` and `Search` when running on multiple CPU cores. `BuildIndex` only needs to run once and it takes 4.5 hours on 4 cores and only 6.9 minutes on 160 cores in parallel. Trapdoor and Search are invoked per query. It takes about 10 minutes to run Trapdoor on 4 cores and less than 1 minute when running on more than 64 cores. Search, which is the most computationally intensive, takes more than 15 hours on 4 cores and less than 30 minutes on 160 cores.

Chapter 10

Conclusions

Searchable symmetric encryption allows a data owner to outsource its data to a cloud server while maintaining the ability and therefore to search over it. Most existing SSE schemes leak access patterns, and therefore are vulnerable to attacks like the IKK attack. Oblivious RAM can be used to construct SSE scheme that fully hides access patterns. However such schemes suffer from heavy communication overhead making them impractical. Chen et al. proposed an obfuscation framework to protect existing SSE schemes against access-pattern leakage. The framework can produce differentially private access patterns per keyword. However, it cannot hide whether the same keyword is being searched multiple times or, in other words, the search patterns.

In this work, we proposed a stronger security definition for differentially private searchable symmetric encryption scheme and presented a real construction, DP-SSE, fulfilling it. On the one hand, DP-SSE is adaptively semantically secure and provides differential privacy for both keywords and documents implying search-pattern hiding and access-pattern hiding. On the other hand, DP-SSE has communication overhead as small as $O(\log\log n)$ and computation complexity of $O(n \cdot \log\log n)$ while searching for relatively frequent keywords. When assuming queries follow Zipfian distribution, the amortized communication overhead would be $O(\log n \log\log n)$. By replicating the IKK attack, we showed that our proposed scheme could actually hide access patterns and make it difficult for the server to extract useful information from differentially private access-pattern leakage. By performing

KMeans clustering, we were able to show that inferring search patterns from differentially private access pattern leakage is difficult, namely search patterns are hidden.

References

[1] Daniel Apon, Jonathan Katz, Elaine Shi, and Aishwarya Thiruvengadam. Verifiable oblivious storage. In *International Workshop on Public Key Cryptography*, pages 131–148. Springer, 2014.

[2] Raphael Bost. $\sum o\varphi o\varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154. ACM, 2016.

[3] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in cryptology–CRYPTO 2013*, pages 353–373. Springer, 2013.

[4] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679. ACM, 2015.

[5] Guoxing Chen, Ten-Hwang Lai, Michael K Reiter, and Yinqian Zhang. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 810–818. IEEE, 2018.

[6] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 41–50. IEEE, 1995.

[7] Lenore Cowen, Wayne Goddard, and C Esther Jesurum. Defective coloring revisited. *Journal of Graph Theory*, 24(3):205–219, 1997.

[8] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[9] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography Conference*, pages 145–174. Springer, 2016.

[10] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014.

[11] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: efficient oblivious ram in two rounds with applications to searchable encryption. In *Annual International Cryptology Conference*, pages 563–592. Springer, 2016.

[12] Eu-Jin Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[13] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[14] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Ndss*, volume 20, page 12, 2012.

[15] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 94–124. Springer, 2017.

[16] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International Conference on Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.

[17] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 965–976. ACM, 2012.

[18] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 146–162. Springer, 2008.

[19] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J Wu. Function-hiding inner product encryption is practical. In *International Conference on Security and Cryptography for Networks*, pages 544–562. Springer, 2018.

[20] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373. IEEE, 1997.

[21] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious ram lower bound! In *Annual International Cryptology Conference*, pages 523–542. Springer, 2018.

[22] Chang Liu, Liehuang Zhu, Mingzhong Wang, and Yu-An Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265: 176–188, 2014.

[23] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[24] Tarik Moataz, Travis Mayberry, and Erik-Oliver Blass. Constant communication oram with small blocksize. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 862–873. ACM, 2015.

[25] Muhammad Naveed. The fallacy of composition of oblivious ram and searchable encryption. *IACR Cryptology ePrint Archive*, 2015:668, 2015.

[26] Muhammad Naveed, Manoj Prabhakaran, and Carl A Gunter. Dynamic searchable encryption via blind storage. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 639–654. IEEE, 2014.

[27] Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In *Annual Cryptology Conference*, pages 502–519. Springer, 2010.

[28] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *Theory of Cryptography Conference*, pages 457–473. Springer, 2009.

[29] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[30] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[31] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit oram: On tightness of the goldreich-ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 850–861. ACM, 2015.

[32] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security Symposium*, pages 707–720, 2016.

[33] George Kingsley Zipf. Selected studies of the principle of relative frequency in language. 1932.

APPENDICES

Appendix A

Proofs

It should be noted that proofs of Lemma A.2 and A.3 are inspired by the proofs in section 1.2 of Mitzenmacher's thesis [23].

A.1 Proof of Theorem 4.3.1

Proof by example. Here we give two SSE schemes in which each only gives either differential privacy for keywords (dp-keywords) or differential privacy for documents (dp-documents) but not both.

An SSE gives only dp-keywords. Let A be an SSE scheme. Modify A to A' in the following way:

- When intending to query w, query w' instead where $w' = w$ with probability p and $w' = w''$ with probability $q = \dfrac{1-p}{|\Delta \setminus w|}$ for $w'' \in \Delta \setminus w$.

It can be shown that A only gives dp-keywords but not dp-documents.

An SSE gives only dp-documents. We modify DP-SSE to DP-SSE$'$ in the following way:

- When intending to query w, add plaintext w into query tokens set.
- All the other parts remain the same.

It can be shown that DP-SSE′ gives dp-documents but not dp-keywords. The reason is the actual search content is revealed in each query token set. Anyone can obtain the search pattern once upon receiving the query tokens.

A.2 Proof of Lemma 5.4.1

Let $e_{i,j}$ be the event that the i^{th} bin has at least j balls, then

$$Pr[e_{i,j}] \leq \binom{k}{j}\left(\frac{1}{k}\right)^j \leq \left(\frac{ke}{j}\right)^j \cdot \left(\frac{1}{k}\right)^j = \left(\frac{e}{j}\right)^j$$

Let $j = \dfrac{c\ln k}{\ln\ln k}$ where $c \geq 3$ is a constant,

$$\begin{aligned}
Pr[e_{i,j}] \leq \left(\frac{e}{j}\right)^j &= exp\left(\frac{c\ln k}{\ln\ln k} \cdot \ln\frac{e\ln\ln k}{c\ln k}\right) \\
&= exp\left(\frac{c\ln k}{\ln\ln k} \cdot (1 + \ln\ln\ln k - \ln\ln k - \ln c)\right) \\
&\leq exp\left(\frac{c\ln k}{\ln\ln k} \cdot (\ln\ln\ln k - \ln\ln k)\right) \\
&= exp\left(-c\ln k + \frac{c\ln k \cdot \ln\ln\ln k}{\ln\ln k}\right) \\
&\leq exp\left(-(c-1)\ln k\right) \text{ when k is large.} \\
&= \frac{1}{k^{c-1}}.
\end{aligned}$$

A.3 Proof of Lemma 5.4.3

We first define some notations, Lemmas and Claims. Then we use such things to do the proof.

Define the height of a ball as i if it is the i^{th} ball thrown into the bin. Let $Ball_{\#\leq i}$ and $Bin_{\#\leq i}$ be the number of balls of height at most i and the number of bins that has at most i balls, respectively. Define $Ball_{\#=i}, Ball_{\#\geq i}, Bin_{\#\geq i}$ and $Bin_{\#=i}$ in a similar way, then we can obtain

$$Ball_{\#=i} \geq Bin_{\#\geq i}.$$

Lemma A.3.1. Suppose at most an α fraction of the bins have been marked. Let X be the number of marked balls (a ball is marked if both of the bins it inspects are marked). Then $E[X] \leq k\alpha^2$.

Claim A.3.2. Suppose $\alpha^2 \geq \dfrac{3c\log k}{k}$, then $X \leq 2k\alpha^2$ with probability at least $1 - \frac{1}{k^c}$ where c is a constant.

Proof. Let $\mu = E[X], \delta = \dfrac{\lambda}{\mu} = \dfrac{n\alpha^2}{\mu}$, use the Chernoff bound,

$$\begin{aligned} Pr[X \geq 2k\alpha^2] &\leq Pr[X - \mu \geq k\alpha^2] \\ &\leq exp\left(-\frac{\delta^2\mu}{2\mu+\lambda}\right) = exp\left(-\frac{\lambda^2}{2\mu+\lambda}\right) \\ &\leq exp\left(-\frac{\lambda}{3}\right), \text{ since } \lambda \geq \mu \\ &\leq exp\left(-c\log k\right) \leq \frac{1}{k^c}, \text{ since } \lambda = k\alpha^2 \geq 3c\log k. \end{aligned}$$

□

Define $\alpha_3 = \dfrac{1}{3}$ and $\alpha_i = 2\alpha_{i-1}^2$ for $i \geq 4$, then we have

$$\alpha_i = \frac{1}{2}\cdot\left(\frac{2}{3}\right)^{2^{i-3}}.$$

Define e_i be the event that $Bin_{\#\geq i} \leq k\alpha_i$. Notice that at most a third of the bins can have 3 balls or more, we have

$$Pr[e_3] = Pr[Bin_{\#\leq \frac{k}{3}}] = 1.$$

Claim A.3.3. If $\alpha_i^2 \geq \dfrac{3c\log k}{k}$, then $Pr[\neg e_{i+1}] \leq \dfrac{i+1}{k^c}$.

Proof. Proof by induction.

Base case. When $i = 3$, $Pr[\neg e_3] = 1 - Pr[e_3] = 0 < \dfrac{4}{k}$.

When $i = j$, assume $Pr[\neg e_j] \leq \dfrac{j}{k^c}$ for $j \geq 3$.

When $i = j + 1$, since $\neg e_{j+1} \subset \neg e_j$.

$$Pr[\neg e_{j+1}] = Pr[\neg e_{j+1} | e_j] Pr[e_j] + Pr[\neg e_j] \leq Pr[\neg e_{j+1} | e_j] + Pr[\neg e_j].$$

By Claim A.3.2,

$$Pr[\neg e_{j+1} | e_j] = Pr[Bin_{\# \geq k 2 \alpha_j^2} | Bin_{\# \leq k \alpha_j}] \leq \frac{1}{k^c}.$$

Therefore,

$$Pr[\neg e_{j+1}] \leq Pr[\neg e_{j+1} | e_j] + Pr[\neg e_j] \leq \frac{1}{k^c} + \frac{j}{k^c} = \frac{j+1}{k^c}.$$

□

Now we do the final proof in two steps. We first show that when i is small i.e. $i = O(\log \log k)$, $Bin_{\# \leq i}$ is bounded by $k\alpha_i$ which will decrease quadratically. Then we show that when i becomes larger than $O(\log \log n)$, $Bin_{\# \geq i}$ will be smaller than 1 with high probability, in other words, no bin will have balls more than $O(\log \log n)$.

When $\alpha_i^2 \geq \dfrac{3c \log k}{k}$, by union bound

$$Pr[\cup_i \neg e_i] \leq \sum_i Pr[\neg e_i] \leq \frac{1}{k^{c-2}}. \tag{A.1}$$

This means that the fraction of bins to have at most i balls is bounded by α_i. The largest i denoted by i^* to make $\alpha_i^2 \geq \dfrac{3c \log k}{k}$ is $i^* \approx \log_2 \log_{3/2} k + 2$.

When $\alpha_i^2 \le \dfrac{3c\log k}{k}$, or in other words, $i \ge i^*$, by union bound,

$$\begin{aligned} Pr[Ball_{\#=i^*+1} = j] &\le \binom{k}{j}(\alpha_{i^*}^2)^j \le \binom{k}{j}\left(\frac{3c\log k}{k}\right)^j \\ &\le \left(\frac{ke}{j}\right)^j \cdot \left(\frac{3c\log k}{k}\right)^j = \left(\frac{3ce\log k}{jk}\right)^j \\ &\le \frac{1}{k^{j-1}}, \text{ when k is large.} \end{aligned}$$

$$\begin{aligned} Pr[Ball_{\#=i^*+2} \ge 1] &= \sum_{j=2}^{k} \binom{j}{2}\left(\frac{j}{k}\right)^2 Pr[Ball_{\#=i^*+1} = j] \\ &\le \sum_{j=2}^{k} \frac{j^2}{2}\left(\frac{j}{k}\right)^2 \frac{1}{k^{j-1}} \\ &\le \frac{2^2}{2}\left(\frac{2}{k}\right)^2 \frac{1}{k} + \frac{3^2}{2}\left(\frac{3}{k}\right)^2 \frac{1}{k^2} + \sum_{j=4}^{k} \frac{j^2}{2}\left(\frac{j}{k}\right)^2 \frac{1}{k^{j-1}} \\ &= O(\frac{1}{k^3}). \end{aligned} \tag{A.2}$$

Combining equation (A.1), (A.2), we can obtain that the number of balls in any bin is no more than $i^* + 2 = O(\log\log k)$ with probability at least $min\left(1 - \dfrac{1}{k^{c-2}}, 1 - \dfrac{1}{k^3}\right)$ when $c \ge 3$.

A.4 Proof of Theorem 5.4.2

Recall that applying the *1-choice* strategy to DP-SSE is analogous to play the *Balls into Bins* game in the *1-choice* setting for $|\Delta|$ times. Since $C_{max} = O(n), |\Delta| = O(n)$, choose c in Lemma A.2 as $c = \log_k n|\Delta| + 1$, then by union bound,

$$Pr[counter_{max} = O\left(\frac{\ln k}{\ln\ln k}\right)] \ge 1 - |\Delta| \cdot \frac{1}{k^c} = 1 - |\Delta|\frac{1}{k^{\log_k n|\Delta|}} = 1 - \frac{1}{n}.$$

A.5 Proof of Theorem 5.4.4

Recall that applying the *2-choice* strategy to DP-SSE is analogous to play the *Balls into Bins* game in the *2-choice* setting for $|\Delta|$ times. Since $C_{max} = O(n), |\Delta| = O(n)$, choose c in Lemma A.3 as $c = \log_k n|\Delta| + 2$ which is no more than 5, then by union bound,

$$Pr[counter_{max} = O(\log\log k)] \geq 1 - |\Delta| \cdot \frac{1}{k^c} = 1 - |\Delta| \frac{1}{k^{\log_k n|\Delta|}} = 1 - \frac{1}{n}.$$

A.6 Proof of Theorem 7.0.4

We only focus on the differential privacy for documents part of DP-SSE-3.

Recall what has been mentioned in section 5.4.3, the access pattern of DP-SSE-3 when querying w is $(\Pi_w^{(1)}, \Pi_w^{(2)})$ where $\Pi_w^{(t)}$ is a multinary vector of length $n + |h|$ for $t \in \{1, 2\}$. Let $\Pi_w^{(t)} = (b_1^{(t)}, ..., b_n^{(t)}, b_{n+1}^{(t)}, ..., b_{n+h}^{(t)})$ for $t \in \{1, 2\}$ For and where for enumerating. Let $\hat{\Pi}_w = (a_1, ..., a_n)$ be the real access pattern when querying w.

Recall that

- for $i \leq n$,
  - if $a_i = 0$, then $b_i^{(1)} = v_i^{(1)}$ and $b_i^{(2)} = v_i^{(2)}$ where $v_i^{(1)} \sim \mathbf{G}_0(1 - q)$ and $v_i^{(2)} \sim \mathbf{G}_0(1 - q)$;
  - if $a_i = 1$ and assuming $h_{t^*}(i)$ is selected by w to generate the concatenation when generating search key for $\mathcal{D}(i)$, then $b_i^{(t^*)} = u_i + v_i^{(t^*)}$ and $b_i^{(1-t^*)} = v_i^{(1-t^*)}$ where $u_i \sim \mathbf{Bern}(p), v_i^{(1)} \sim \mathbf{G}_0(1 - q)$, $v_i^{(2)} \sim \mathbf{G}_0(1 - q)$ and $t^* \in \{1, 2\}$;
- for $i \leq |h|$, $b_{n+i}^{(t)} = u_{n+i}^{(t)} + v_{n+i}^{(t)}$ for $t \in \{1, 2\}$ where $u_{n+i}^{(t)} \sim \mathbf{B}(g^{(t)}(w, i), p)$ and $v_{n+i}^{(t)} \sim \mathbf{G}_0(1 - q)$.

The value of $g^{(t)}(w, i)$ indicates the number of possible non-match tokens originated from `TruePosSet` to have label i when `Search` considers only h_t to label documents. In other words,

$$g^{(t)}(w, i) = counter_{max} - |\{\mathcal{D}[j] \mid h_t(j) = i, w \in \mathcal{D}[j] \text{ and } h_t(j) \text{ is selected by } w\}|.$$

Now we begin our proof.

Differential Privacy for Documents. Given a pair of neighbouring databases $\mathcal{D}$ and $\mathcal{D}'$ where $\mathcal{D}[i] \neq \mathcal{D}'[i]$ and $|\mathcal{D}[i]| = |\mathcal{D}'[i]|$, and a list of querying keywords $\vec{w} \in \Delta^{|\vec{w}|}$. Let w be the keyword such that it is only in one of $\mathcal{D}$ and $\mathcal{D}'$.

If $w \notin \vec{w}$, then $Pr[\mathcal{M}(\mathcal{D}, \vec{w}) \in S] = Pr[\mathcal{M}(\mathcal{D}', \vec{w}) \in S]$.

If $w \in \vec{w}$, let $\Lambda = \{j \mid \vec{w}[j] = w\}$ and $\vec{w}_\Lambda = (\vec{w}[\Lambda[1]], \vec{w}[\Lambda[2]], ..., \vec{w}[\Lambda[|\Lambda|]])$ where $\Lambda[j]$ represents the j^{th} smallest element in Λ.

$$\begin{aligned} \frac{Pr[\mathcal{M}(\mathcal{D}, \vec{w}) \in S]}{Pr[\mathcal{M}(\mathcal{D}', \vec{w}) \in S]} &= \frac{Pr[\mathcal{M}(\mathcal{D}, \vec{w}_\Lambda) \in S]}{Pr[\mathcal{M}(\mathcal{D}', \vec{w}_\Lambda) \in S]} = \left(\frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]}\right)^{|\Lambda|} \\ \frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]} &\leq \max_{(\Pi^{(1)},\Pi^{(2)}) \in S} \frac{Pr[\mathcal{M}(\mathcal{D}, w) = (\Pi^{(1)}, \Pi^{(2)})]}{Pr[\mathcal{M}(\mathcal{D}', w) = (\Pi^{(1)}, \Pi^{(2)})]}. \end{aligned} \tag{A.3}$$

It should be noted that when querying w, the distributions of all $b_j^{(t)}$s (and $b_j^{\prime(t)}$) are determined given the content of database $\mathcal{D}$ (and $\mathcal{D}'$). Let $\Upsilon^{(t)} = \{j \mid b_j^{(t)} \neq b_j^{\prime(t)}\}$ where $b_j^{(t)} \neq b_j^{\prime(t)}$ stands for that $b_j^{(t)}$ and $b_j^{\prime(t)}$ follow different distributions. Let $\Upsilon^{(t)}_{\leq n} = \{j \mid j \in \Upsilon^{(t)}, j \leq n\}$ and $\Upsilon^{(t)}_{>n} = \{j \mid j \in \Upsilon^{(t)}, j > n\}$. Then we have $|\Upsilon^{(t)}_{\leq n}| \leq C_{max}$ and $\Upsilon^{(t)}_{>n} \leq |h| = C_{max}$. The reason for such bounds is that adding or removing one keyword w to or from a document $\mathcal{D}[i]$ to obtain $\mathcal{D}'[i]$ can only affect the search keys of the documents containing w of which the number is $|\mathcal{D}'(w)| \leq C_{max}$. Let $\Upsilon^{(t)}_{\leq n, \mathcal{D} > \mathcal{D}'} = \{j \mid j \in \Upsilon^{(t)}_{j \leq n}, b_j^{(t)} > b_j^{\prime(t)}\}$ where $b_j^{(t)} > b_j^{\prime(t)}$ means $b_j^{(t)} = u_j + v_j^{(t)}$ but $b_j^{\prime(t)} = v_j^{\prime(t)}$ where $u_j \sim \mathbf{Bern}(p), v_j^{(t)} \sim \mathbf{G}_0(1-q)$ and $v_j^{\prime(t)} \sim \mathbf{G}_0(1-q)$. Let $\Upsilon^{(t)}_{\leq n, \mathcal{D} < \mathcal{D}'} = \Upsilon^{(t)}_{j \leq n} \setminus \Upsilon^{(t)}_{j \leq n, \mathcal{D} > \mathcal{D}'}$. It should be noted that

$$\begin{aligned} \forall j \in \Upsilon^{(t)}_{j \leq n, \mathcal{D} > \mathcal{D}'}, \quad \frac{Pr[b_j^{(t)} = \alpha]}{Pr[b_j^{\prime(t)} = \alpha]} &\leq \frac{p + (1-p)q}{q} \text{ (refer to (7.1))}; \\ \forall j \in \Upsilon^{(t)}_{j \leq n, \mathcal{D} < \mathcal{D}'}, \quad \frac{Pr[b_j^{(t)} = \alpha]}{Pr[b_j^{\prime(t)} = \alpha]} &\leq \frac{1}{1-p} \text{ (refer to (7.3))}. \end{aligned}$$

Let $\Upsilon^{(t)}_{>n, \mathcal{D} > \mathcal{D}'} = \{j \mid j \in \Upsilon^{(t)}_{>n}, b_j^{(t)} > b_j^{\prime(t)}\}$ where $b_j^{(t)} > b_j^{\prime(t)}$ means that $g^{(t)}(w, j-n) > g^{\prime(t)}(w, j-n)$ where $b_j^{(t)} = u_j^{(t)} + v_j^{(t)}$ and $b_j^{\prime(t)} = u_j^{\prime(t)} + v_j^{\prime(t)}$ where $u_j^{(t)} \sim \mathbf{B}(g^{(t)}(w, j-n), p), u_j^{\prime(t)} \sim \mathbf{B}(g^{\prime(t)}(w, j-n), p)$, $v_j^{(t)} \sim \mathbf{G}_0(1-q)$ and $v_j^{\prime(t)} \sim \mathbf{G}_0(1-q)$. Let $\Upsilon^{(t)}_{j > n, \mathcal{D} < \mathcal{D}'} =$

$\Upsilon^{(t)}_{>n} \setminus \Upsilon^{(t)}_{>n,\mathcal{D}>\mathcal{D}'}$. It should be noted that

$$\forall j \in \Upsilon^{(t)}_{>n,\mathcal{D}>\mathcal{D}'},\ \ \frac{Pr[b^{(t)}_j = \alpha]}{Pr[b'^{(t)}_j = \alpha]} \le \left(1 - p + \frac{p}{q}\right)^{g^{(t)}(w,j-n) - g'^{(t)}(w,j-n)} \quad (\text{ refer to } (7.8));$$

$$\forall j \in \Upsilon^{(t)}_{>n,\mathcal{D}<\mathcal{D}'},\ \ \frac{Pr[b^{(t)}_j = \alpha]}{Pr[b'^{(t)}_j = \alpha]} \le \left(\frac{1}{1-p}\right)^{g'^{(t)}(w,j-n) - g^{(t)}(w,j-n)} \quad (\text{ refer to } (7.7)).$$

Now we can rewrite Equation (A.3) as

$$\begin{aligned} \frac{Pr[\mathcal{M}(\mathcal{D},w) \in S]}{Pr[\mathcal{M}(\mathcal{D}',w) \in S]} &\le \max_{(\Pi^{(1)},\Pi^{(2)})\in S} \frac{Pr[\mathcal{M}(\mathcal{D},w) = (\Pi^{(1)},\Pi^{(2)})]}{Pr[\mathcal{M}(\mathcal{D}',w) = (\Pi^{(1)},\Pi^{(2)})]} \\ &= \max_{\alpha^{(t)}_j} \prod_{t\in\{1,2\}} \prod_{j\in\Upsilon^{(t)}} \frac{Pr[b^{(t)}_j = \alpha^{(t)}_j]}{Pr[b'^{(t)}_j = \alpha^{(t)}_j]} \end{aligned} \tag{A.4}$$

When $w \in \mathcal{D}[i]$ but $w \notin \mathcal{D}'[i]$,
$\sum_{t\in\{1,2\}} |\Upsilon^{(t)}_{\le n,\mathcal{D}>\mathcal{D}'}| - |\Upsilon^{(t)}_{\le n,\mathcal{D}<\mathcal{D}'}| = 1$ and $\sum_{t\in\{1,2\}} \sum_{j\in\Upsilon^{(t)}_{>n}} g^{(t)}(w,j-n) - g'^{(t)}(w,j-n) = -1$.
Let $A = \sum_{t\in\{1,2\}} |\Upsilon^{(t)}_{\le n,\mathcal{D}>\mathcal{D}'}|$ and $B = \sum_{t\in\{1,2\}} \sum_{j\in\Upsilon^{(t)}_{>n,\mathcal{D}<\mathcal{D}'}} g'^{(t)}(w,j-n) - g^{(t)}(w,j-n)$. Then $A \le C_{max}$, $B \le C_{max}$ and

$$\begin{aligned} \frac{Pr[\mathcal{M}(\mathcal{D},w) \in S]}{Pr[\mathcal{M}(\mathcal{D}',w) \in S]} &\le \max_{\alpha^{(t)}_j} \prod_{t\in\{1,2\}} \prod_{j\in\Upsilon^{(t)}} \frac{Pr[b^{(t)}_j = \alpha^{(t)}_j]}{Pr[b'^{(t)}_j = \alpha^{(t)}_j]} \\ &\le \left(\frac{p+(1-p)q}{q}\right)^{A} \cdot \left(\frac{1}{1-p}\right)^{A-1} \cdot \left(1-p+\frac{p}{q}\right)^{B-1} \cdot \left(\frac{1}{1-p}\right)^{B} \\ &= \left(\frac{p+(1-q)q}{(1-p)q}\right)^{A+B-1} < \left(1+\frac{p}{(1-p)q}\right)^{2C_{max}}. \end{aligned} \tag{A.5}$$

When $w \notin \mathcal{D}[i]$ but $w \in \mathcal{D}'[i]$,
$\sum_{t\in\{1,2\}} |\Upsilon^{(t)}_{\le n,\mathcal{D}>\mathcal{D}'}| - |\Upsilon^{(t)}_{\le n,\mathcal{D}<\mathcal{D}'}| = -1$ and $\sum_{t\in\{1,2\}} \sum_{j\in\Upsilon^{(t)}_{>n}} g^{(t)}(w,j-n) - g'^{(t)}(w,j-n) = 1$.
Let $A = \sum_{t\in\{1,2\}} |\Upsilon^{(t)}_{\le n,\mathcal{D}<\mathcal{D}'}|$ and $B = \sum_{t\in\{1,2\}} \sum_{j\in\Upsilon^{(t)}_{>n,\mathcal{D}>\mathcal{D}'}} g^{(t)}(w,j-n) - g'^{(t)}(w,j-n)$. Then

$A \leq C_{max}$, $B \leq C_{max}$ and

$$\begin{aligned}\frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]} &\leq \max_{\alpha_j^{(t)}} \prod_{t \in \{1,2\}} \prod_{j \in \Upsilon^{(t)}} \frac{Pr[b_j^{(t)} = \alpha_j^{(t)}]}{Pr[b_j'^{(t)} = \alpha_j^{(t)}]} \\ &\leq \left(\frac{p + (1-p)q}{q}\right)^{A-1} \cdot \left(\frac{1}{1-p}\right)^{A} \cdot \left(1 - p + \frac{p}{q}\right)^{B} \cdot \left(\frac{1}{1-p}\right)^{B-1} \\ &= \left(\frac{p + (1-q)q}{(1-p)q}\right)^{A+B-1} < \left(1 + \frac{p}{(1-p)q}\right)^{2C_{max}}. \qquad \text{(A.6)}\end{aligned}$$

As a consequence,

$$\frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]} \leq \left(1 + \frac{p}{(1-p)q}\right)^{2C_{max}}.$$

Therefore, we obtain

$$\frac{Pr[\mathcal{M}(\mathcal{D}, \vec{w}) \in S]}{Pr[\mathcal{M}(\mathcal{D}', \vec{w}) \in S]} \leq \left(\frac{Pr[\mathcal{M}(\mathcal{D}, w) \in S]}{Pr[\mathcal{M}(\mathcal{D}', w) \in S]}\right)^{|\Lambda|} \leq \left(1 + \frac{p}{q(1-p)}\right)^{|\vec{w}| \cdot 2C_{max}}.$$

Appendix B

Algorithms

B.1 A Function-Hiding Inner Product Predicate Encryption Scheme

Fix a security parameter λ, and let n be a positive integer. The function-hiding inner product predicate encryption scheme is constructed as follows.

- $\mathsf{Setup}(1^\lambda)$: The Setup algorithm takes as input the security parameter and output a secret key sk as follows. It first samples an asymetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, e)$ and chooses generators $g1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$, then it samples $\mathbf{B} \leftarrow \mathbb{GL}_n(\mathbb{Z}_q)$ and sets $\mathbf{B}* = \det(\mathbf{B}) \cdot (\mathbf{B}^{-1})^T$. Finally, it sets $sk = (g_1, g_2, \mathbf{B}, \mathbf{B}^*)$.
- $\mathsf{Encrypt}(\mathsf{sk}, \vec{x})$: The $\mathsf{Encrypt}$ takes as input the secret key sk and an attribute $\vec{x}$, then samples $\alpha \xleftarrow{R} \mathbb{Z}_q$ and finally outputs

$$c_{\vec{x}} = (C1, C_2) = \left(g_1^{\alpha \cdot \det(\mathbf{B})}, g_1^{\alpha \cdot \vec{x} \cdot \mathbf{B}}\right).$$

- $\mathsf{GenToken}$ $(sk, \vec{f})$: The $\mathsf{GenToken}$ takes as input the secret key sk and an predicate $\vec{f}$, then samples $\beta, \gamma \xleftarrow{R} \mathbb{Z}_q$ and finally outputs

$$\tau_{\vec{f}} = (T_1, T_2) = \left(g_2^{\beta}, g_2^{\beta\gamma \cdot \vec{f} \cdot \mathbf{B}^*}\right).$$

- $\mathsf{Query}(c_{\vec{x}}, \tau_{\vec{f}})$: The Query algorithm takes as input a ciphertext $c_{\vec{x}}$ and a token $\tau_{\vec{f}}$, then computes

  $$D_1 = e(C_1, T_1) \text{ and } D_2 = e(C_2, T_2)$$

  finally returns *True* if $D_1^0 = D_2$, otherwise returns *False*.

Correctness.

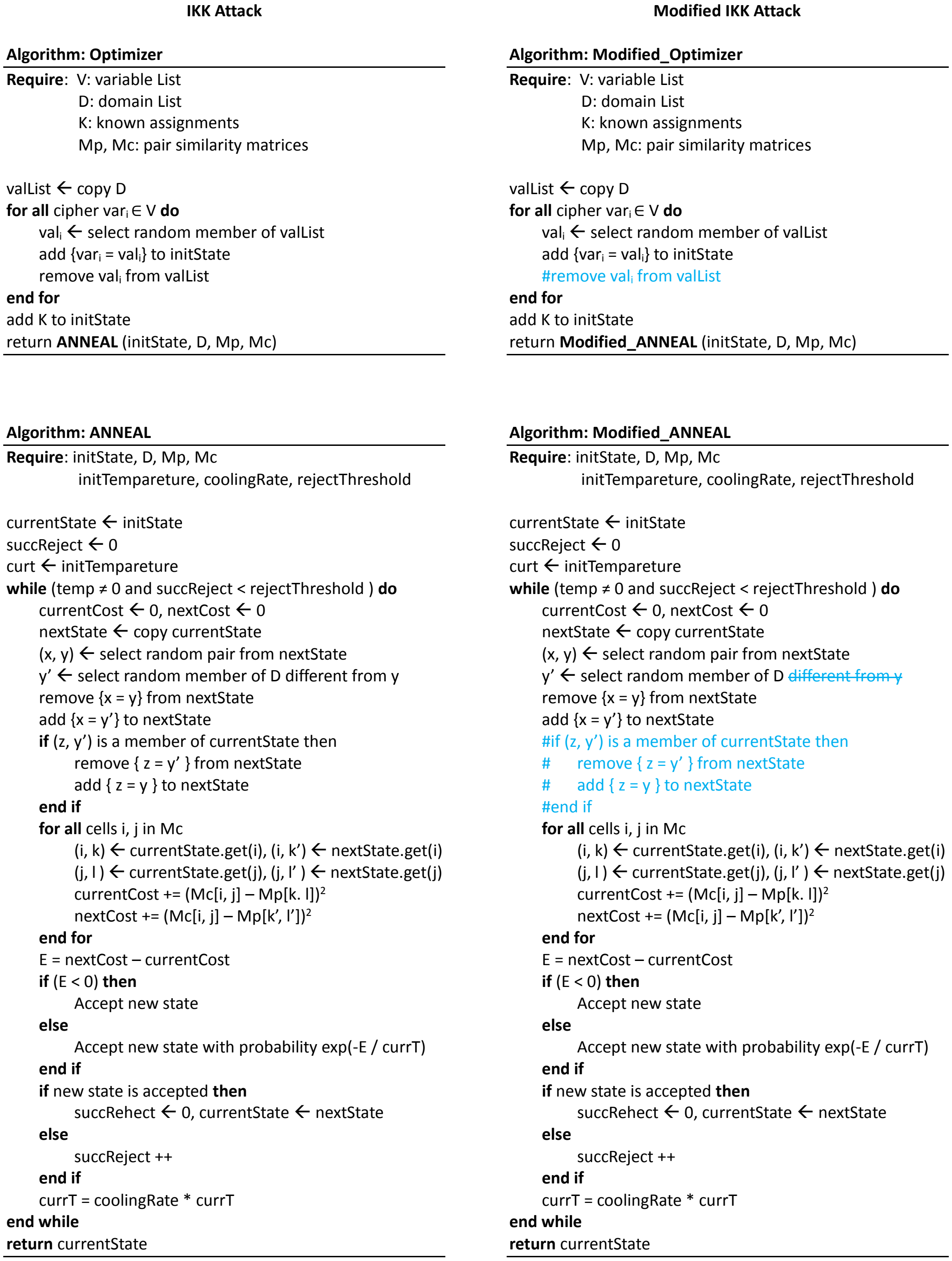
$$D_1 = e(C_1, T_1) = e(g_1, g_2)^{\alpha\beta\cdot\det(\mathbf{B})}$$

$$D_2 = e(C_2, T_2) = e(g_1, g_2)^{\alpha\beta\gamma\cdot\vec{x}\mathbf{B}(\mathbf{B}^*)^T\vec{f}^T} = e(g_1, g_2)^{\alpha\beta\cdot\det(\mathbf{B})\gamma\langle\vec{x},\vec{f}\rangle} \text{ since } \mathbf{B}\cdot\mathbf{B}^* = \det(\mathbf{B})\cdot\mathbf{I}$$

$D_1^0 = D_2$ holds if and only if $\gamma\langle\vec{x}, \vec{f}\rangle = 0$. Due to the fact that γ is chosen uniformly random, $\gamma\langle\vec{x}, \vec{f}\rangle$ holds if and only if $\langle\vec{x}, \vec{f}\rangle = 0$ with overwhelming probability.

Security. With the same proof strategy as in [19], the above IPPE scheme can be proved SIM-secure in the generic group model.

B.2 IKK Attack and Modified IKK Attack



IKK Attack

Algorithm: Optimizer

```
Require:  V: variable List
          D: domain List
          K: known assignments
          Mp, Mc: pair similarity matrices

valList ← copy D
for all cipher $var_i \in V$ do
    $val_i$ ← select random member of valList
    add {$var_i = val_i$} to initState
    remove $val_i$ from valList
end for
add K to initState
return ANNEAL (initState, D, Mp, Mc)
```

Algorithm: ANNEAL

```
Require: initState, D, Mp, Mc
         initTempareture, coolingRate, rejectThreshold

currentState ← initState
succReject ← 0
curt ← initTempareture
while (temp ≠ 0 and succReject < rejectThreshold ) do
    currentCost ← 0, nextCost ← 0
    nextState ← copy currentState
    (x, y) ← select random pair from nextState
    y' ← select random member of D different from y
    remove {x = y} from nextState
    add {x = y'} to nextState
    if (z, y') is a member of currentState then
        remove { z = y' } from nextState
        add { z = y } to nextState
    end if
    for all cells i, j in Mc
        (i, k) ← currentState.get(i), (i, k') ← nextState.get(i)
        (j, l ) ← currentState.get(j), (j, l' ) ← nextState.get(j)
        currentCost += $(Mc[i, j] – Mp[k. l])^2$
        nextCost += $(Mc[i, j] – Mp[k', l'])^2$
    end for
    E = nextCost – currentCost
    if (E < 0) then
        Accept new state
    else
        Accept new state with probability exp(-E / currT)
    end if
    if new state is accepted then
        succRehect ← 0, currentState ← nextState
    else
        succReject ++
    end if
    currT = coolingRate * currT
end while
return currentState
```

Modified IKK Attack

Algorithm: Modified_Optimizer

```
Require:  V: variable List
          D: domain List
          K: known assignments
          Mp, Mc: pair similarity matrices

valList ← copy D
for all cipher $var_i \in V$ do
    $val_i$ ← select random member of valList
    add {$var_i = val_i$} to initState
    #remove $val_i$ from valList
end for
add K to initState
return Modified_ANNEAL (initState, D, Mp, Mc)
```

Algorithm: Modified_ANNEAL

```
Require: initState, D, Mp, Mc
         initTempareture, coolingRate, rejectThreshold

currentState ← initState
succReject ← 0
curt ← initTempareture
while (temp ≠ 0 and succReject < rejectThreshold ) do
    currentCost ← 0, nextCost ← 0
    nextState ← copy currentState
    (x, y) ← select random pair from nextState
    y' ← select random member of D ~~different from y~~
    remove {x = y} from nextState
    add {x = y'} to nextState
    #if (z, y') is a member of currentState then
    #     remove { z = y' } from nextState
    #     add { z = y } to nextState
    #end if
    for all cells i, j in Mc
        (i, k) ← currentState.get(i), (i, k') ← nextState.get(i)
        (j, l ) ← currentState.get(j), (j, l' ) ← nextState.get(j)
        currentCost += $(Mc[i, j] – Mp[k. l])^2$
        nextCost += $(Mc[i, j] – Mp[k', l'])^2$
    end for
    E = nextCost – currentCost
    if (E < 0) then
        Accept new state
    else
        Accept new state with probability exp(-E / currT)
    end if
    if new state is accepted then
        succRehect ← 0, currentState ← nextState
    else
        succReject ++
    end if
    currT = coolingRate * currT
end while
return currentState
```