Searching on Encrypted Data

Submitted by

Sarah Louise Renwick

for the degree of Doctor of Philosophy

of the

Royal Holloway, University of London

2017

Declaration

I, Sarah Louise Renwick, hereby declare that this thesis and the work presented in it is entirely my own. Where I have consulted the work of others, this is always clearly stated.

Signed . . . . (Sarah Louise Renwick)
Date:

For Harry and Ker.

Abstract

Searchable encryption allows a user to outsource encrypted data to a remote server, whilst preserving the user's ability to locate specific data items within the encrypted data that satisfy some query. In its simplest form, searchable encryption allows a user to locate all data items that contain a particular keyword.

In this thesis, we analyse searchable encryption schemes and assess their suitability for various scenarios occurring in the real world. Despite the existence of practical searchable encryption schemes in the literature, there is limited evidence of their deployment. We discuss issues that we consider to be inhibiting the widespread adoption of searchable encryption. This work aims to present searchable encryption as a useable technology and, by analysing the efficiencies of the various schemes within the different scenarios, we intend to make the design of new real-world searchable encryption protocols an easier task.

We also present two new searchable encryption schemes. A number of searchable encryption schemes have been proposed that are secure in the presence of a *semi-honest* server, which may deviate from the protocol in order to conserve its own resources. In existing schemes, the search queries a user can perform are not particularly expressive. We use techniques from publicly verifiable computation to build a searchable encryption scheme that can evaluate more expressive queries in a verifiable manner; that is, the user is able to verify whether or not the server has computed the search honestly.

Our second construction allows users with different access rights to the data to receive search results which are dependent on their access rights. We call this type of scheme searchable encryption with multi-level access. Most existing searchable encryption with multi-level access schemes are built using attribute based encryption, a form of public key cryptography. Symmetric key cryptography uses simpler and easier to implement primitives, compared to its public-key counterpart. We present a construction that extends a well-known searchable symmetric encryption scheme to support users searching the data at different access levels.

Acknowledgement

The last five years have been an incredible journey, one which I would not have been able to complete on my own. For me, it has been not only an intense intellectual journey, but also a more personal one. I have learnt a lot about myself: my limitations and also ways of working and being that allow me to achieve my potential and be my best self. Special thanks to Rich, Mother and Ponyo for being there every step of the way. I am looking forward to all the new adventures we will embark upon together in this next chapter.

I would like to thank my supervisor, Keith Martin, for all his support and for guiding me through my research. Our meetings would always inspire new ideas and give me a renewed energy to continue, even when I felt most defeated. Many thanks to my examiners Kenny Patterson and Liqun Chen for taking the time to go through my thesis with a fine tooth comb. Their discussions and comments helped me enormously in forming the final version of my thesis and in producing a piece of work I am very proud of. Thanks also to Sarah Meiklejohn for her advice and support, and for being such a great work buddy and wonderful friend.

Many thanks to my two co-authors, James Alderman and Christian Janson. It was a pleasure to work and share ideas with these two brilliant people.

I would like to thank the Department of Mathematics at the University of Bristol for providing such a lively and rigorous undergraduate course and for introducing me to the mind-altering world of pure mathematics, my thoughts will never be the same again. Thanks to Richard Porter and Andrey Bovykin for their kind words of encouragement when I approached them with the idea of doing a PhD and especially to Andrey for pointing me in the direction of cryptography and for all the magic philosophical conversations we shared during my last year in Bristol.

Thanks to the team at Thales for being so welcoming and supportive during my work placements and always being eager to listen to my ideas. Special thanks to Adrian Waller and Glyn Jones for mentoring me during my time at Thales.

I share my life with some incredible, outrageous and brilliant people who support me in so many ways. A giant collective thank you to all my friends! Thank you for inspiring me, exploring with me and helping me weather the storms.

This thesis is dedicated to Harry Swordy and Ker Standerwick, two very dear friends who would have been so proud of me for finally reaching the top of this mountain and not looking back.

Contents

List of Figures

List of Games

List of Tables

List of Notation

$x\|\|y$	the concatenation of x and y
$\|M\|$	length of bit string M
$\{0,1\}^n$	all binary strings of length n
$\{0,1\}^*$	all binary strings
κ	security parameter
1^κ	the unary representation of the security parameter
$\mathbb{P}[A]$	the probability of an event A occurring
$\emptyset$	the empty set
$[i,j]$	the set of integers {i,i+1,...,j-1,j}
$[i]$	the set of integers {1,...,i}
$S\backslash s$	the set S with the element s removed
$\{s_i\}_{i\in[n]}$	the set $\{s_1,...,s_n\}$
$negl(x)$	a negligible function on input x
PRF	pseudorandom function
PRP	pseudorandom permutation
$\oplus$	the binary operation XOR
$y \leftarrow x$	assigning the value x to variable y
$y \leftarrow A(y)$	the output of A being assigned to variable y, where A is a deterministic algorithm
$y \overset{\$}{\leftarrow} A(y)$	the output of A being assigned to variable y, where A is a probabilistic algorithm
$x \in X$	the element x is a member of the set X

List of Abbreviations

SKE	Symmetric Key Encryption
PKE	Public Key Encryption
VSE	Verifiable Searchable Encryption
VC	Verifiable Computation
VDC	Verifiable Delegable Computation
eVSE	Extended Verifiable Searchable Encryption
DE	Deterministic Encryption
CP-ABE	Ciphertext Policy Attribute Based Encryption
ABE	Attribute-Based Encryption
PBE	Predicate-Based Encryption
BE	Broadcast Encryption
SE	Searchable Encryption
SSE	Symmetric Searchable Encryption
MSSE	Multi-User Symmetric Searchable Encryption
MLSSE	Multi-Level Symmetric Searchable Encryption
IND-CPA	Indistinguishability against Chosen Plaintext Attacks

Chapter 1

Introduction

Contents

This chapter provides the motivation for, and structure of, the thesis.

1.1 Motivation

Nowadays, almost all data is stored digitally. Public cloud service providers provide an infrastructure that gives businesses and individuals access to computing power and storage space to support this digital data on a flexible pay-as-you-go basis. This allows them to bypass the costs associated with having their own data centres such as hardware, construction, air conditioning and security costs. This makes cloud computing a very cost-effective solution for both bulk data processing and data storage.

According to a government survey on information security and data breaches relating to digital data in 2017 [103], 46 per cent of all UK businesses identified at least one attack by an unauthorised outsider and suffered a data breach in the last 12 months (up from 24 per cent the previous year). The number is higher for medium and large firms, of which 66 per cent and 68 per cent respectively suffered at least one data breach.

This demonstrates a clear need for secure data storage in order to ensure the confidentiality of data. However, encrypting data is just one part of the solution. Encrypting data ensures that, in the event of a compromise, no meaningful information should leak about the data itself if the data is compromised, yet it also reduces the possibility of performing computations on the ciphertexts, such as searching for keywords or specific items within the data.

Suppose that, just as in a typical cloud storage environment, your data is stored in encrypted form on a remote server due to local data storage constraints. To locate a piece of data we could download the entirety of the encrypted data, decrypt it, and search over the unencrypted data. Alternatively, perhaps we could create an index for the encrypted data that is stored locally and used to navigate the encrypted files. Both of these methods provide adequate solutions in theory, yet in practice they present several problems. Firstly, the size of the encrypted data may not be known a priori, or be known in advance to be very large, both of which deem the process of downloading all of the encrypted data highly inefficient and costly. Furthermore, the reason for storing the data remotely in the first place is due to the unavailability of local storage, so downloading all the data in this case would not be an option. Creating an index would require locally storing a file which may be of size in the order of the number of encrypted files, which again may not be feasible due to local storage restrictions. In addition, the index itself could potentially leak information about the encrypted data, compromising confidentiality.

Searchable encryption (SE) provides a solution to this problem by supporting the outsourcing of encrypted data to a remote server, whilst maintaining the ability to search for specific keywords within the encrypted data.

1.2 Thesis Outline and Contributions

In this thesis we investigate different methods for searching on encrypted data. In Chapter 2, we introduce some notation and background material relevant to the remainder of the thesis. In Chapter 3, we define the paradigm of searchable encryption, along with associated security definitions and adversarial models. We also give an overview of the related literature in the field.

Chapter 4 establishes four scenarios in order to categorize the searchable encryption schemes in the literature. These scenarios are categorized in terms of the number of users and the tasks that these users are able to perform within the system. The aim of this chapter is to present searchable encryption in a real-world setting and map the various schemes in the literature to each of these scenarios. We define a set of features that can vary within each scenario, such as the number of data items to be searched over and the type of search query required. We also analyse the efficiencies of a number of searchable encryption schemes, and use this information to map the schemes into the scenarios. We highlight which scheme is most suitable according to the varying features.

Although the literature on searchable encryption is vast, searchable encryption schemes do not seem to be widely deployed. The aim of this work is to aid the deployment of searchable encryption by helping potential users to choose the most suitable scheme for a given scenario.

Chapter 5 focuses on a branch of searchable encryption called verifiable searchable encryption, which assumes a more powerful adversary. In this model the server is not trusted to compute the search honestly, hence an extra algorithm is needed to perform verification of search results. We first identify some similarities between the models of searchable encryption and verifiable outsourced computation. We then use techniques from the field of verifiable outsourced computation to construct a verifiable searchable encryption scheme with more expressive queries.

The focus of Chapter 6 is on multi-level access within searchable encryption schemes. In many real-world scenarios for searchable encryption, not all users will have the same access rights to the encrypted data. Most schemes in the literature that provide a solution to this issue are based on computationally intensive cryptographic paradigms, such as attribute-based encryption. We present a symmetric key solution to multi-level access within searchable encryption which provides a more practical alternative. Within this chapter we also present numerous generic ways of extending a single-user searchable encryption scheme to one that supports multi-level access.

We draw our conclusions and identify areas of future work in Chapter 7.

1.3 Author Contributions

The work presented in this thesis is derived from three published papers [7, 120, 121], all of which I am listed as the main author. As my supervisor, Keith Martin is a co-author on all three papers, and helped to inspire and guide my ideas throughout the research.

The work in Chapter 4 was co-authored by Keith Martin. The main body of work was developed by myself, guided by helpful notes and comments from Martin. The ideas in Section 4.5 were inspired through discussions with delegates at the 'Future Directions in Computing on Encrypted Data' symposium which was held in Bristol in November 2015.

The paper that formed the basis of Chapter 5 was co-authored by James Alderman and Christian Janson. I recognised the similarity of the system models within their area of research, verifiable computation and my own area of research which inspired the work. I lead the development of the system model and the security notion of index privacy. The rest of the work of [120] was jointly developed by all authors.

The paper from which Chapter 6 was derived was co-authored by James Alderman. Most of the main ideas of the paper were developed by myself through invaluable discussions with Alderman.

Chapter 2

Background Material

Contents

This chapter provides details on the notation used throughout the thesis, along with background information on relevant cryptographic primitives and security notions.

2.1 Notation

In this section, we give details of some common notation used throughout the remainder of the thesis.

We use $\mathbb{Z}$ to denote the set of integers, $\mathbb{N} \subset \mathbb{Z}$ to denote the set of natural numbers, $\mathbb{Z}_p{}^*$ is the multiplicative group $\{1, ...p-1\}$, and the symbol $\emptyset$ represents the empty set. We use $[i, j]$ and $[n]$ to represent the sets of integers $\{i, i+1, ..., j-1, j\}$ and $\{1, ..., n\}$, respectively, where $i, j, n \in \mathbb{N}$ and $i < j$. The notation $S \backslash \{s\}$ represents the set S with the element s removed, and $\{s_i\}_{i \in [n]}$ represents the set $\{s_1, ..., s_n\}$.

In our binary operations *true* is denoted as 1, and *false* as 0. We use $\vee$ to denote the binary operator OR, which produces a result of 1 if either one of the operands are 1. We use $\wedge$ to denote the binary operation AND, which produces a result of 1 only if both operands are 1.

We use $\mathbb{P}[A]$ to denote the probability of event A occuring.

2.2 Cryptographic Primitives

In this section, we give details of some common cryptographic primitives used in the remainder of the thesis.

2.2.1 Negligible functions

We use *negligible functions* when defining the security of an encryption scheme. In this thesis we deal with encryption schemes that are *computationally* secure as opposed to *information-theoretically* secure, which remain secure against computationally unbounded adversaries. The one-time pad is a common example of an encryption scheme exhibiting this form of security. It is not always possible to achieve such high levels of security. In practice, where the presence of a computationally unbounded adversary may be unrealistic, a computationally limited adversary is assumed which is bounded to run in polynomial time with respect to the security parameter used in the key generation algorithm of the encryption scheme. Similarly inverse polynomial probabilities are considered to be a significant threshold. If there exists a polynomial $p(n)$ and an adversary can break the security of an encryption scheme with probability $\frac{1}{p(n)}$ then the scheme is considered to be insecure. If the probability of breaking the security of the encryption is asymptotically smaller than $\frac{1}{p(n)}$ for every possible polynomial, then the encryption scheme is considered secure as the probability is so small [87, Definition 3.4]. This is referred to as *negligible* probability. We define a negligible function as follows:

Definition 2.2.1 (Negligible Function). *A function negl* $: \mathbb{N} \to \mathbb{R}$ *is defined as negligible if for any polynomial* $p(x)$ *there exists a natural number* $\lambda \in \mathbb{N}$ *such that, for all natural numbers* $x > \lambda$*:*

$$negl(x) < \frac{1}{p(x)}.$$

2.2.2 Pseudorandom functions

A *pseudorandom function* is a family of functions that, given a random instance from this family of functions, and the ability to pass it inputs and receive the outputs, it is hard to tell whether the function you are interacting with is a random instance from the family of functions or a truly random function. In practice pseudorandom functions are used in place of truly random functions as the family of functions defining a pseudorandom function is much smaller than that defining the set of all truly random functions. We define random a pseudorandom function as follows:

Definition 2.2.2 (Pseudorandom Function (PRF)). *A function* $f : \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^m$ *is a PRF if:*

- *Given a key* $K \in \{0,1\}^\kappa$ *and an input* $x \in \{0,1\}^n$ *one can efficiently compute* $f_K(x)$.
- *For all probabilistic polynomial-time algorithms* D*, there exists a negligible function negl such that:*

$$|\mathbb{P}[K \overset{\$}{\leftarrow} \{0,1\}^\kappa, D^{f_K}(1^\kappa)] - \mathbb{P}[f \overset{\$}{\leftarrow} \mathcal{F}, D^f(1^\kappa)]| \leq negl(\kappa),$$

where $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}^m\}$.

2.2.3 Pseudorandom permutations

Pseudorandom permutations are defined similarly to pseudorandom functions, except it is required that their output be indistinguishable from that of a truly random permutation, rather than a truly random function. A pseudorandom permutation is in fact an invertible pseudorandom function which we define as follows:

Definition 2.2.3 (Pseudo Random Permutation (PRP))**.** *Let* $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be an efficient, length-preserving, keyed function. We say that* f *is a pseudorandom permutation if, for every* $K \leftarrow \{0,1\}^\kappa$, $f_K(\cdot)$ *is one-to-one (this in fact implies that* f *is a bijection as it also length-preserving) and, for all probabilistic polynomial-time distinguishers* D*, there exists a negligible function* $negl$ *such that:*

$$|\mathbb{P}[D^{f_K(\cdot)}(1^\kappa) = 1] - \mathbb{P}[D^{f(\cdot)}(1^\kappa) = 1]| \leq negl(\kappa),$$

where $K \leftarrow \{0,1\}^\kappa$ *is chosen uniformly at random and* f *is chosen uniformly at random from the set of permutations mapping* κ*-bit strings to* κ*-bit strings [87, Section 3.6.3].*

Definition 2.2.4 (Strong PRP)**.** *A PRP* f *is a strong PRP if the inverse* f^{-1} *of the PRP is also a PRP.*

2.2.4 Hash functions

A *hash function* is an algorithm that maps arbitrary sized input to a (usually) fixed size output. Hash functions are generally used to compare values which cannot be stored in the clear. In this these we use a type of hash function known as a *keyed* hash function, which is defined as follows:

Definition 2.2.5 (Keyed Hash Function)**.** *A keyed hash function is a pair of probabilistic polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{H})$ *that satisfy the following:*

- $K \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa)$ *is a probabilistic algorithm which takes as input a security parameter* 1^κ *and outputs a key* K*;*
- *There exists a polynomial* ℓ *such that* H *takes as input a key* K *and a string* $x \in \{0,1\}^*$ *and outputs a string* $\mathsf{H}_K(x) \in \{0,1\}^{\ell(k)}$.

A hash function is defined as collision resistant if the probability of two different inputs hashing to the same output is negligible. That is, for any probabilistic polynomial time algorithm $\mathcal{A}$ *we have that:*

$$\mathbb{P}[K \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa), (x_1, x_2) \leftarrow \mathcal{A}(K, 1^\kappa) : x_1 \neq x_2, H_K(x_1) = H_K(x_2)] \leq negl(k).$$

2.2.5 One-way functions

A *one-way function* is a function that is easy to compute on the entirety of its domain, but it is hard to invert the function given the image of a random input. It is formally defined as follows:

Definition 2.2.6 (One-way Function)**.** *A function $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function if:*

- *f can be evaluated in polynomial time.*
- *For every probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ there is a negligible function such that:*

$$\underset{X \overset{\$}{\leftarrow} \{0,1\}^\kappa}{\mathbb{P}} [\mathcal{A}(f(X)) \in f^{-1}(f(X))] \leq negl(\kappa), \forall \kappa.$$

2.3 Encryption schemes

This section presents information on encryption schemes that are used in the thesis, along with their formal correctness definitions. Encryption is a common method for protecting the confidentiality of data. In order to make sense of encrypted data you need to have access to the secret key that is required to decrypt the data. Access to encrypted data is usually granted on an all-or-nothing basis, meaning that you can either decrypt the entire data, or you cannot. However, advanced encryption techniques such as attribute-based encryption (Section 2.3.3), predicate-based encryption (Section 2.3.4) and searchable encryption (Chapter 3) are able to facilitate partial access to the encrypted data, such as evaluating a query over the encrypted data.

The algorithms in the encryption schemes can either be deterministic or probabilistic. A deterministic algorithm always outputs the same ciphertext when supplied with the same key and plaintext, whenever it is executed. In contrast, a probabilistic algorithm produces a random ciphertext, even on executions involving the same key and plaintext.

Throughout this thesis, we will use $\mathcal{M}$ to denote the plaintext space, $\mathcal{C}$ to denote the ciphertext space, $\mathcal{K}$ to denote the keyspace, and $\kappa \in \mathbb{N}$ to denote the security parameter of an encryption scheme..

2.3.1 Symmetric-key encryption

In *symmetric-key encryption*, the key used to encrypt the data is the same as the key used to decrypt the data. This presents the major issue associated with using symmetric-key encryption, which is how to share the secret key. The symmetric secret key must be kept secret at all times in order to ensure the confidentiality of the encrypted data and a secure method is required to distribute the key to all parties whom we wish to have the capability to encrypt and decrypt data.

Definition 2.3.1 (Symmetric-key Encryption (SKE))**.** *Let $\mathcal{K}$ be the key space, $\mathcal{M}$ be the plaintext space and $\mathcal{C}$ be the ciphertext space. A symmetric-key encryption scheme consists of the following three algorithms:*

- $SK \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa)$*: A probabilistic algorithm that takes as input the security parameter $\kappa \in \mathbb{N}$ and outputs a secret key $SK \in \mathcal{K}$.*
- $c \overset{\$}{\leftarrow} \mathsf{Enc}(m, SK)$*: A probabilistic or deterministic algorithm that takes as input a plaintext $m \in \mathcal{M}$ and the secret key SK, and outputs a ciphertext $c \in \mathcal{C}$.*
- m *or* $\perp \leftarrow \mathsf{Dec}(c, SK)$*: A deterministic algorithm that takes as input a ciphertext $c \in \mathcal{C}$ and outputs either $m \in \mathcal{M}$ or $\perp$.*

An SKE scheme is correct if, when using the same key, the decryption algorithm reverses encryption. That is, if a plaintext is encrypted using a key SK, then running the decryption algorithm on this ciphertext using the same key will return the original plaintext.

Definition 2.3.2 (Correctness of SKE)**.** *An SKE scheme is correct if for all $\kappa \in \mathbb{N}$, for all $m \in \mathcal{M}$:*

$$\begin{aligned}\mathbb{P}[SK &\overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa),\\ c &\overset{\$}{\leftarrow} \mathsf{Enc}(m, SK),\\ m &\leftarrow \mathsf{Dec}(c, SK)] = 1.\end{aligned}$$

2.3.2 Public-key encryption

In *public-key encryption*, the key used to encrypt the data is different from the key used to decrypt it. The decryption key must be kept secret, however the encryption key can be made public. This solves the issue associated with key transfer in SKE, since a user is able to encrypt a plaintext for someone using a public key which can be made freely available, for example by publishing it online. This means that the party encrypting the data needs to know the recipient of the data before encryption in order to use the correct public key.

Definition 2.3.3 (Public Key Encryption (PKE))**.** *Let $\mathcal{K}$ be the key space, $\mathcal{M}$ be the plaintext space and $\mathcal{C}$ be the ciphertext space. A public-key encryption scheme consists of the following three algorithms:*

- $(PK, SK) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa)$*: A probabilistic algorithm that takes as input a security parameter $\kappa \in \mathbb{N}$ and outputs a public and secret key pair $(PK, SK) \in \mathcal{K}$.*
- $c \overset{\$}{\leftarrow} \mathsf{Enc}(m, PK)$*: A probabilistic or deterministic algorithm that takes as input a plaintext $m \in \mathcal{M}$, the public key PK, and outputs a ciphertext $c \in \mathcal{C}$.*
- *m or* $\perp \leftarrow \mathsf{Dec}(c, SK)$*: A deterministic algorithm that takes as input a ciphertext $c \in \mathcal{C}$ and outputs either a plaintext m or $\perp$.*

The correctness of PKE is similar to that of SKE, in that the decryption algorithm reverses the encryption algorithm.

Definition 2.3.4 (Correctness of PKE)**.** *A PKE scheme is correct if for all $\kappa \in \mathbb{N}$, for all $m \in \mathcal{M}$:*

$$\begin{aligned}\mathbb{P}[(PK, SK) &\overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa),\\ c &\overset{\$}{\leftarrow} \mathsf{Enc}(m, PK),\\ m &\leftarrow \mathsf{Dec}(c, SK)] = 1.\end{aligned}$$

2.3.3 Attribute-based encryption

The paradigm of *attribute-based encryption* is a form of PKE where a user's key is associated with general descriptive attributes. These attributes can be descriptors such as the user's role, location, type of subscription or the department the user works in, for example. Ciphertexts are associated with *monotonic* access structures, which means that if there exists a set of attributes that satisfy the access structure, then any superset of this set will also satisfy the access structure. The access structures are composed of $\vee$ and $\wedge$ gates, and a ciphertext can only be decrypted if the attributes on the secret key satisfy the policy determined by the access structure.

There are two variants of attribute-based encryption. The one we will describe is referred to as *ciphertext-policy attribute-based encryption*, while the other is known as *key-policy attribute-based encryption.* In key-policy attribute-based encryption the monotonic access structure is associated with the keys instead of the ciphertext, and the ciphertext is associated with a set of descriptive attributes. We give a formal definition of ciphertext-policy attribute-based encryption only as this is the only variant we utilise in the thesis.

Attribute-based encryption allows a user to encrypt data without knowing the specific identity of the recipient(s); the user only needs to specify a policy involving a general set of attributes that they require the recipient to possess. For example, an employee at a company could encrypt data so that every manager in the finance department is able to decrypt it by embedding the policy 'FINANCE $\wedge$ MANAGER' into each ciphertext. Applying an access structure to each ciphertext allows the data owner to enforce a more fine-grained access policy to the encrypted data, whereas in standard SKE and PKE (Definitions 2.3.1 and 2.3.3), access to the encrypted data is all-or-nothing, meaning that a user is either authorized to decrypt all data items or none at all. We denote a set of attributes A that satisfy a policy $\mathbb{A}$ as $A \models \mathbb{A}$.

Definition 2.3.5 (Ciphertext Policy Attribute Based Encryption (CP-ABE))**.** *Let $\mathcal{U}$ be the attribute space, $\mathcal{M}$ be the plaintext space, $\mathcal{K}$ be the key space, $\mathcal{C}$ be the ciphertext space and $\mathbb{A}^*$ be the access structure space. A ciphertext policy attribute based encryption scheme consists of the following four algorithms:*

- $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{U})$*: A probabilistic algorithm that takes as input a security parameter $\kappa \in \mathbb{N}$ and outputs a master public key and master secret key pair $(PK, SK) \in \mathcal{K}$.*

- $c \xleftarrow{\$} \mathsf{Encrypt}(m, \mathbb{A}, PK)$*: A probabilistic algorithm that takes as input an access structure $\mathbb{A} \in \mathbb{A}^*$, a plaintext $m \in \mathcal{M}$ and the public key PK, and outputs a ciphertext $c \in \mathcal{C}$ associated with access structure $\mathbb{A}$.*

- $SK_A \xleftarrow{\$} \mathsf{KeyGen}(SK, A, PK)$*: A probabilistic algorithm that takes as input the master secret key SK and a set of attributes $A \in \mathcal{U}$ that describe the key (or the user that the key is produced for). It outputs a secret key $SK_A \in \mathcal{K}$ associated with the set of attributes A.*

- *m or* $\perp \leftarrow \mathsf{Decrypt}(SK_A, c, PK)$*: A deterministic algorithm that takes as input a ciphertext $c \in \mathcal{C}$ associated with an access structure $\mathbb{A}$, a secret key SK_A associated with some set of attributes A, and the public key PK. It outputs a plaintext $m \in \mathcal{M}$ or $\perp$.*

Definition 2.3.6 (Correctness of CPABE)**.** *A CP-ABE scheme is correct if for all $\kappa \in \mathbb{N}$, for all $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, for all attribute sets $A \in \mathcal{U}$, for any key $SK_A \xleftarrow{\$} \mathsf{KeyGen}(SK, A, PK)$, and for all access structures $\mathbb{A} \in \mathbb{A}^*$, if A satisfies $\mathbb{A}$,* *then:*

$$m \leftarrow \mathsf{Decrypt}(SK_A, c, PK).$$

2.3.4 Predicate-based encryption

Predicate-based encryption is a generalisation of ABE that associates ciphertexts with a set of attributes X, and the secret keys with a predicate F (as opposed to the access structure in ABE). Decryption succeeds if $F(X) = 1$. Unlike ABE, predicate based encryption is both *attribute hiding* and *predicate hiding*, meaning that both the predicates and attributes are concealed (there are some examples of ABE that also conceal this information [108, 96] but, ABE is not necessarily attribute hiding by definition). Predicate based encryption can support the following types of queries over the encrypted data: disjunctive queries, polynomial evaluation (where the set of attributes is $\mathbb{Z}_n$ and the predicates are in the form of polynomials in $\mathbb{Z}_n[x]$; the predicate evaluates to 1 if the corresponding polynomial evaluates to 0 on the attributes), threshold queries, and predicates corresponding to conjunctive normal form (CNF) or disjunctive normal form (DNF) formulae.

Definition 2.3.7 (Predicate Based Encryption (PBE))**.** *Let $\mathcal{P}$ be the predicate space, $\mathcal{K}$ be the key space, $\mathcal{M}$ be the plaintext space, $\mathcal{C}$ be the ciphertext space and $\mathcal{U}$ be the attribute space. A Predicate Based Encryption (PBE) scheme consists of the following four algorithms:*

- $(PK, SK) \overset{\$}{\leftarrow} \mathsf{Setup}(1^\kappa, \mathcal{P})$*: A probabilistic algorithm that takes as input a security parameter $\kappa \in \mathbb{N}$ and a predicate space $\mathcal{P}$, and outputs a master public key and master secret key pair $(PK, SK) \in \mathcal{K}$.*
- $SK_p \overset{\$}{\leftarrow} \mathsf{KeyGen}(SK, p, PK)$*: A probabilistic algorithm that takes as input the master secret key SK and the description of a predicate $p \in \mathcal{P}$. It outputs a secret key $SK_p \in \mathcal{K}$ associated with the predicate p.*
- $c \overset{\$}{\leftarrow} \mathsf{Encrypt}(m, A, PK)$*: A probabilistic algorithm that takes as input a set of attributes $A \in \mathcal{U}$, a plaintext $m \in \mathcal{M}$ and the public key PK, and outputs a ciphertext $c \in \mathcal{C}$ associated with the set of attributes A.*
- *m or* $\perp \leftarrow \mathsf{Decrypt}(SK_p, c, PK)$*: A deterministic algorithm that takes as input a ciphertext $c \in \mathcal{C}$ associated with some set of attributes $A \in \mathcal{U}$, and a secret key SK_p associated with a predicate $p \in \mathcal{P}$. It outputs a plaintext $m \in \mathcal{M}$ or $\perp$.*

Definition 2.3.8 (Correctness of PBE). *A PBE scheme is correct if for all $\kappa \in \mathbb{N}$, for all $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$, for all plaintexts $m \in \mathcal{M}$, for all $p \in \mathcal{P}$, for any key $SK_p \xleftarrow{\$} \mathsf{KeyGen}(SK, p)$, and for all sets of attributes $A \in \mathcal{U}$, if $f(A) = 1$, then:*

$$m \leftarrow \mathsf{Decrypt}(SK_p, c, PK),$$

and if $f(A) \neq 1$:

$$\begin{aligned}\mathbb{P}[(PK, SK) &\xleftarrow{\$} \mathsf{Setup}(1^\kappa),\\ SK_p &\xleftarrow{\$} \mathsf{KeyGen}(SK, p),\\ c &\xleftarrow{\$} \mathsf{Encrypt}(m, A, PK),\\ \perp &\leftarrow \mathsf{Decrypt}(SK_p, c, PK)] = 1 - negl(\kappa).\end{aligned}$$

2.3.5 Broadcast encryption

Broadcast encryption provides an efficient way of allowing different sets of authorised users access to encrypted data delivered via a broadcast channel. The trivial way of achieving this would be to distribute a unique public and secret key pair to each user in the authorised set, and then encrypt the data using each of the different public keys. Hence, if there are n users in the authorised set then this method requires n different encryptions of the same data to be broadcast to all users in order for each authorised user to be able to decrypt the data.

Using broadcast encryption requires only one ciphertext to be broadcast to all users, which can reduce the bandwidth required. When encrypting data using a broadcast encryption scheme, a set of authorised users are input into the encryption algorithm, along with the plaintext. Encryption ensures that only users belonging to the set of authorised users input into the encryption algorithm are able to decrypt the resulting ciphertext. Broadcast encryption works by distributing keying information to each user so that authorised users are able to construct the decryption key, whereas unauthorised users are not.

A *Broadcast encryption (*BE*)* scheme consists of a set of three polynomial time algorithms $\mathsf{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Enc}, \mathsf{BE.Dec})$ such that KeyGen generates the public key and n secret keys for each user in the system, Enc takes the set of authorised users, a plaintext and the public key as input, and outputs the BE encryption for the set of authorised users, and Dec takes a user's identity, their secret key, the BE encryption of a plaintext,the public key and the set of authorised users, and outputs either the data (if the user is authorised to view the data), or $\perp$ otherwise.

In the following definition we consider a data owner broadcasting a ciphertext to a set of users, some of which may not be authorised to decrypt it.

Definition 2.3.9 (Broadcast Encryption (BE))**.** *Let $\mathcal{U}$ be the user identity space, $\mathcal{G} \subseteq \mathcal{U}$ a set of authorised users, $\mathcal{K}$ be the keyspace, $\mathcal{C}$ be the ciphertext space and $\mathcal{M}$ be the plaintext space. A Broadcast Encryption (*BE*) scheme consists of the following four algorithms:*

- $(PK, SK) \overset{\$}{\leftarrow} \mathsf{Keygen}(1^\kappa, h)$*: A probabilistic algorithm that takes a security parameter $\kappa \in \mathbb{N}$ (and possibly a value h which determines the maximum number of users that can be revoked from the system), and outputs a public and secret key pair $(PK, SK) \in \mathcal{K}$.*
- $SK_u \overset{\$}{\leftarrow} \mathsf{Add}(SK, u)$*: A probabilistic algorithm that takes a user identity $u \in \mathcal{U}$ and the secret key SK, and outputs a secret key $SK_u \in \mathcal{K}$.*
- $c \overset{\$}{\leftarrow} \mathsf{Encrypt}(m, \mathcal{G}, PK)$*: A probabilistic algorithm that takes a plaintext $m \in \mathcal{M}$, a set of users $\mathcal{G} \subseteq \mathcal{U}$ that are authorized to decrypt the resulting ciphertext and the public key PK. It outputs a ciphertext $c \in \mathcal{C}$.*
- $(m \text{ or } \perp) \leftarrow \mathsf{Decrypt}(SK_u, c)$*: A deterministic algorithm run by a user that takes a ciphertext c and the user's secret key SK_u. The algorithm outputs either a plaintext m or $\perp$.*

Definition 2.3.10 (Correctness of BE)**.** *A BE scheme is correct if for all $\kappa \in \mathbb{N}$, $(PK, SK) \overset{\$}{\leftarrow} \mathsf{Keygen}(1^\kappa, h)$, for all $u \in \mathcal{U}$, for all $\mathcal{G} \in \mathcal{U}$, for any key $SK_u \overset{\$}{\leftarrow} \mathsf{Add}(SK, u)$, for all plaintexts m, if $u \in \mathcal{G}$:*

$$\begin{aligned}\mathbb{P}[(PK, SK) &\overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa, h),\\ SK_u &\overset{\$}{\leftarrow} \mathsf{Add}(SK, u),\\ c &\overset{\$}{\leftarrow} \mathsf{Encrypt}(m, \mathcal{G}, PK),\\ m &\leftarrow \mathsf{Decrypt}(SK_u, c, PK)] = 1.\end{aligned}$$

2.3.6 Order-Preserving Encryption

Order-preserving encryption is a form of deterministic encryption that produces ciphertexts that preserve the ordering of the underlying plaintexts. In order to define OPE *we first give a definition of an order-preserving function:*

Definition 2.3.11 (Order-Preserving Function). *For $A, B \subseteq \mathbb{N}$ with $|A| \leq |B|$, a function $f : A \to B$ is order-preserving if for all $i, j \in A$, $f(i) > f(j)$ iff $i > j$.*

Definition 2.3.12 (Order-Preserving Encryption (OPE)). *Let $\mathsf{DE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a deterministic encryption scheme with keyspace $\mathcal{K}$, plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. DE is an order-preserving encryption scheme if $\mathsf{Enc}(K, \cdot)$ is an order preserving function from $\mathcal{M}$ to $\mathcal{C}$ for all K output by KeyGen.*

Definition 2.3.13 (Correctness of OPE). *An OPE scheme is correct if the underlying deterministic encryption scheme DE is correct. See definitions 2.3.2 and 2.3.4.*

2.4 Verifiable Computation

This section presents information on verifiable computation schemes

2.4.1 Verifiable and publicly verifiable computation

Verifiable Computation (VC) allows a client with limited resources to efficiently outsource the computation of a function f on various inputs $x_1, ..., x_n$ to a more powerful server, and to verify the correctness of results. In verifiable computing we consider a semi-honest but curious server (Definition 3.3.2) which may not perform the computation honestly or give a false result in order to save its bandwidth or computational resources. Due to this a user receiving computation results from the server needs to be able to verify that the server executed the computation correctly. The server computes the function on the specified input and returns the results, e.g. $f(x_i)$, along with a proof that the server computed the function correctly on the given input. It is required that the verification of the proof requires less computation by the client than if the client computed the actual function.

Other notable approaches in the realm of querying remote data can be found in [8, 10, 11, 23, 24, 28, 53].

Definition 2.4.1 (Verifiable Computation (VC)). *Let $\mathcal{F}$ be the function space, $\mathcal{X}$ be the input space and $\mathcal{K}$ be the keyspace. A VC scheme consists of the following four algorithms:*

- $(EK_f, SK_f) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa, f)$*: A probabilistic algorithm that takes as input the security parameter 1^κ along with a function $f \in \mathcal{F}$ to be computed. It outputs a public and secret key pair $(EK_f, SK_f) \in \mathcal{K}$. The public key EK_f will be used by the server to compute the function f.*

- $(\sigma_{f,x}, VK_{f,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_f)$: *A probabilistic algorithm that takes as input x along with the secret key SK_f. It outputs the public encoded input $\sigma_{f,x}$, along with a secret verification key $VK_{f,x}$.*
- $\theta_{f(x)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{f,x}, EK_f)$: *A probabilistic function that takes as input the encoded input $\sigma_{f,x}$ along with the public evaluation key EK_f. It outputs an encoding of the computation result, $\theta_{f(x)}$, evaluating $f(x)$.*
- $y \leftarrow \mathsf{Verify}(\theta_{f(x)}, VK_{f(x)}, SK_f)$: *A deterministic algorithm that takes as input the encoded result $\theta_{f(x)}$, the verification key VK_f and the secret key SK_f. It outputs a value y which is either the result of the computation $f(x)$ or $\perp$ if $\theta_{f(x)}$ is not a valid input.*

Definition 2.4.2 (Correctness of VC)**.** *A VC scheme for a set of functions $\mathcal{F}$ is correct if for all $f \in \mathcal{F}$ and for all x in the domain of f:*

$$\begin{aligned}\mathbb{P}[(EK_f, SK_f) &\xleftarrow{\$} \mathsf{Setup}(1^\kappa, f),\\ (\sigma_{f,x}, VK_{f,x}) &\xleftarrow{\$} \mathsf{ProbGen}(x, SK_f),\\ \theta_{f(x)} &\xleftarrow{\$} \mathsf{Compute}(\sigma_{f,x}, EK_f),\\ f(x) &\leftarrow \mathsf{Verify}(\theta_{f(x)}, VK_{f(x)}, SK_f)] = 1.\end{aligned}$$

Gennaro et al. [67] considered the use of garbled circuits, whilst Parno et al. [115] introduced *publicly* verifiable computation (PVC) built from key policy attribute based encryption (KP-ABE), where a single client computes an evaluation key for the server and publishes information enabling other clients to outsource computation to the server. *Any* client may verify the correctness of a result.

2.4.2 Verifiable delegable computation

Alderman et al. [6] considered an alternative system model that used ciphertext policy attribute based encryption (CP-ABE) to allow clients to query computations on data held by the server (or initially outsourced by a client) called *Verifiable Delegable Computation* (VDC). The data is given public, descriptive labels to allow the clients to request computations on subsets of the data. This can naturally be applied to problems like querying on remote data.

Definition 2.4.3 (Verifiable Delegable Computation (VDC))**.** *Let $\mathcal{F}$ be the function space, $\mathcal{S}$ be the server space and $\mathcal{K}$ be the keyspace. A verifiable delegable computation scheme consists of the following algorithms:*

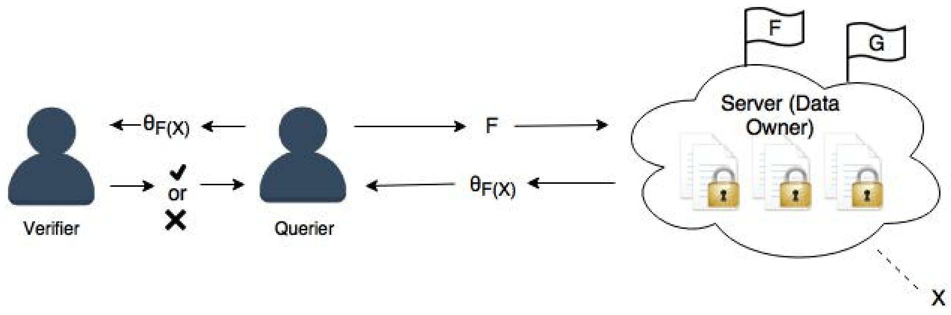

Figure 2.1: Verifiable Delegable Computation.

- $(MK, PP) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{F})$: *A probabilistic algorithm that takes as input security parameter and a family of functions $\mathcal{F}$. It outputs a master secret key MK and public parameters PP.*

- $PK_f \xleftarrow{\$} \mathsf{FnInit}(f, MK, PP)$: *A probabilistic algorithm that takes as input the master secret key, public parameters and a function $f \in \mathcal{F}$. It outputs a public delegation key PK_f which can be used to request computations of the function f.*

- $SK_{S_i} \xleftarrow{\$} \mathsf{Register}(S_i, MK, PP)$: *A probabilistic algorithm used to enrol a computation server into the system. It takes as input the server identity $S_i \in \mathcal{S}$ along with the master key and the public parameters. It outputs the server's secret key, which is used to sign computations carried out by that server.*

- $EK_{D_i,S_i} \xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{\ell(x_{i,j})\}_{x_{i,j}\in D_i}, \mathcal{F}_i, MK, PP)$: *A probabilistic algorithm that takes as input a server identity, a set of data points D_i and associated labels $\{\ell(x_{i,j})\}$, the family of functions $\mathcal{F}_i$ and the master secret key. It outputs an evaluation key EK_{D_i,S_i} which enables server S_i to compute all functions $f \in \mathcal{F}_i$ on data D_i, where $D_i = \{x_{i,j}\}_{j=1}^{m_i}$ is a set of m_i data points each labelled $\ell(x_{i,j})$.*

- $(\sigma_{f,X}, VK_{f,X}, RK_{f,X}) \xleftarrow{\$} \mathsf{ProbGen}(f, \{x_{i,j}\}_{x_{i,j}\in X}, S_i, PK_f, PP)$: *A probabilistic algorithm that takes as input a function f, a set of data point labels $\{\ell(x_{i,j})\}$ belonging to server S_i, the public delegation key PK_f and the public parameters. It outputs a value encoding the function f and the input values, denoted $\sigma_{f,X}$, a public verification key $VK_{f,X}$ and a retrieval key $RK_{f,X}$.*

- $\theta_{f(X)} \xleftarrow{\$} \mathsf{Compute}(\sigma_{f,X}, EK_{D_i,S_i}, SK_{S_i}, PP)$: *A probabilistic algorithm that takes as input the encoded input value $\sigma_{f,X}$, the evaluation key EK_{D_i,S_i}, the server's secret key SK_{S_i} and the public parameters. It computes the function f on the encoded inputs and outputs and produces an encoded result $\theta_{f(X)}$ for $f(X)$.*

- $y_{f(X)} \leftarrow \mathsf{Verify}(\theta_{f(X)}, VK_{f(X)}, RK_{f(X)}, PP)$: *This deterministic algorithm can be run in two parts:*
  - $(RT_{f(X)}, \tau_{\theta_{f(X)}}) \leftarrow \mathsf{BVerif}(\theta_{f(X)}, VK_{f,X}, PP)$: *This algorithm takes as input the encoded output $\theta_{f(X)}$ along with the verification key and the public parameters. It outputs a retrieval token $RT_{f(X)}$ which encodes the output of the computation $f(X)$ and a token $\tau_{\theta_{f(X)}}$ which is assigned value 1 if $\theta_{f(X)}$ is a valid encoding of the computation result, or 0 otherwise.*
  - $y_{f(X)} \leftarrow \mathsf{Retrieve}(RT_{f(X)}, \tau_{\theta_{f(X)}}, VK_{f,X}, RK_{f,X}, PP)$: *This algorithm takes as input the retrieval token $RT_{f(X)}$, the encoded result $\tau_{\theta_{f(X)}}$, the verification key $VK_{f,X}$, the retrieval key $RK_{f,X}$ along with the public parameters. It outputs the value $y_{f(X)}$ which is either the result of the computation $f(X)$ if the server performed correctly, or $\perp$ otherwise.*

Definition 2.4.4 (Correctness of VDC). *A VDC scheme for a family of functions $\mathcal{F}$ is correct if; for all $S_i \in \mathcal{S}$, for all D_i and for all $f \in \mathcal{F}$:*

$$\begin{aligned}
\mathbb{P}[(PP, MK) &\xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{F}),\\
PK_f &\xleftarrow{\$} \mathsf{FnInit}(f, MK, PP),\\
SK_{S_i} &\xleftarrow{\$} \mathsf{Register}(S_i, MK, PP),\\
EK_{D_i,S_i} &\xleftarrow{\$} \mathsf{Certify}(S_i, D_i, \{\ell(x_{i,j})\}_{x_{i,j}\in D_i}, \mathcal{F}_i, MK, PP),\\
(\sigma_{f,X}, VK_{f,X}, RK_{f,X}) &\xleftarrow{\$} \mathsf{ProbGen}(f, \{x_{i,j}\}_{x_{i,j}\in X}, PK_f, PP),\\
\theta_{f(X)} &\xleftarrow{\$} \mathsf{Compute}(\sigma_{f,X}, EK_{D_i,S_i}, SK_{S_i}, PP),\\
(RT_{f(X),\tau_{\theta_{f(X)}}}) &\leftarrow \mathsf{BVerif}(\theta_{f(X)}, VK_{f,X}, PP),\\
f(X) &\leftarrow \mathsf{Retrieve}(RT_{f(X)}, \tau_{\theta_{f(X)}}, VK_{f,X}, RK_{f,X}, PP)] = 1.
\end{aligned}$$

2.5 Types of data

In this section we define the data types that we use in the thesis.

Definition 2.5.1 (Static). *A static dataset is one that does not change over time. That is, it is a fixed dataset in the sense that no data items can be added, removed or updated in the dataset once initialised.*

Definition 2.5.2 (Dynamic). *A dynamic dataset is one that can change over time. Data items can be added, removed or updated after the dataset has been initialised.*

2.6 Methods of proof

There are two main methods of proving the security of a cryptographic scheme: game-based methods and simulation-based methods. In this section we outline both of these methods.

2.6.1 Game-based security proofs

This method of proof is defined in terms of an attack game between two probabilistic entities: an adversary and a challenger. The game is modelled as a probability space and the security notion is related to the probability of some event in this probability space occurring. The challenger can be thought of as a legitimate user and will usually have access to the secret parameters in the system. The adversary models an entity that wants to achieve something against the wishes of the legitimate user and has access to the public parameters and usually some oracles.

An oracle is a system that the challenger provides to the adversary in order to give them some extra information that would otherwise be unavailable to them. Oracles are used to create a realistic environment for the adversary, in order to give them access to all the information that an adversary attacking the scheme in the real world might have. For example, an adverwsary might be able to observe the inputs and outputs to an encryption algorithm in a real world system that requires a secret key as input. However, if the adversary in the game does not have access to the secret key then they cannot run this algorithm themselves, so an oracle is used to allow the adversary access to this information. In this case the oracle gives the adversary the ability to encrypt any plaintext of their choosing and receive a valid ciphertext without having access to the secret key. We denote an adversary $\mathcal{A}$ having access to an oracle $\mathcal{O}$ that has fixed inputs $(x_1, ..., x_n)$ and computes a function F as $\mathcal{A}^{\mathcal{O}^F(\cdot,...,\cdot,x_1,...,x_n)}$, where $\cdot$ denotes an input that is chosen by the adversary.

2.6.2 Simulation-based security proofs

In a simulation-based proof there are two different worlds: an *ideal* world, where the cryptographic scheme is secure by definition and in which an adversary interacts with a simulated version of the cryptographic scheme, and a *real* world which reflects an adversary interacting with the actual cryptographic scheme. The idea behind simulation-based proof is that if the cryptographic scheme in question is secure, then the adversary cannot tell which world it is in, i.e. whether it's interacting with a simulation or the actual scheme. The only information the simulator in the ideal world receives is that which is assumed to be leaked to an adversary in the real world, so it cannot run any of the algorithms or know any secret parameters of the cryptographic scheme in question. Using only the specified information, the simulator has to fulfil the following three tasks [101]:

1. It must generate a view for the real adversary that is indistinguishable from its real view in the real world;
2. It must extract the effective inputs used by the adversary in the execution; and
3. It must make the view generated be consistent with the output that is based on this input.

A simulation-based proof shows that if an adversary cannot tell whether it is interacting with a simulator or the real cryptographic scheme then the cryptographic scheme does not leak any information other than that which is known to be leaked. To illustrate this concept, take the example of a symmetric-key encryption scheme. In the real world, the adversary issues a plaintext to the challenger and receives a genuine ciphertext in return. In the ideal world, the adversary receives a simulated ciphertext that the simulator generates without the secret key or access to the algorithms of the cryptographic scheme. If the adversary cannot tell which world they are interacting with then this shows that the real ciphertext leaks no secret information. This is because the simulator is not in possession of any of the secret information, hence it would be impossible for them to output a ciphertext that leaks this information or any information that the adversary does not already know. If the adversary is unable to distinguish between a simulated ciphertext (that leaks no information) and a real ciphertext, then this implies that the real ciphertext also leaks no secret information, because if it did then this would enable the adversary to distinguish which world it is in. The inability of the adversary to distinguish which world it is operating in proves the security of the cryptographic scheme.

Oracles can also be used in simulation-based proofs. However, if used, it needs to be shown how their output can be simulated in the ideal world.

There are two main types of security associated with game-based and simulation-based proofs in the SE literature: *selective security* and *adaptive security*. We discuss the differences between these two notions of security in Definitions 2.6.1 and 2.6.2.

Definition 2.6.1 (Selective security)**.** *If a cryptographic scheme is selectively secure, then the adversary must make all oracle queries and challenge selections in a batch prior to receiving any inputs such as oracle query responses and public parameters.*

Definition 2.6.2 (Adaptive security)**.** *If a cryptographic scheme is adaptively secure then the adversary is able to make oracle queries and select challenge values that depend on the results of previous queries and any inputs they receive.*

Adaptive security is considered to be the stronger, more desirable, notion of security, since it more accurately reflects the abilities of a real-world adversary. However, these types of proof can be hard to construct, so the notion of selective security is commonly used.

We use game-based methods to define the security of our encryption schemes in Chapters 5 and 6. Our verifiable searchable encryption scheme presented in Chapter 5 is constructed using attribute-based encryption and verifiable computation techniques. It is standard practice in the literature surrounding these paradigms to use game-based proofs. We used this method also to keep in line with the relevant literature and the intuition behind our security notions were well represented using games. In Chapter 6 we explored both methods of proof for our construction. The simulation-based method required the construction to have three random oracles in order to achieve adaptive security and support a dynamic dataset. The issue with using the simulation-based method for this construction came with simulating the index for the adversary ahead of time whilst still being able to simulate consistent search results before and after the addition of data items to the index. The random oracles allowed the challenger to alter the content of the simulated index using the search queries and the tokens for adding and deleting data items. Due to the complexity of our construction the simulation-based method was very complex and difficult to follow. The game-based method was more intuitive and easy to follow for the reader and allowed us to prove security in the standard model without the use of random oracles.

In Section 2.3 we defined several types of encryption scheme and their associated definitions of correctness. In Sections 2.6.3-2.6.7 we define the security of each of these encryption schemes.

2.6.3 Security Model for Symmetric-Key Encryption

The two most widely adopted notions of security for SKE are that of *indistinguishability under chosen plaintext attack (IND-CPA)* and *indistinguishability under chosen ciphertext attack (IND-CCA)*. The notion of security that we use in this thesis is that of IND-CPA, which ensures that the ciphertexts in an IND-CPA secure SKE scheme do not leak any information about the underlying plaintext. We assume that an adversary is able to obtain encryptions of arbitrary plaintexts, i.e. they are able to build up a dictionary of plaintext-ciphertext pairs. However, when given a ciphertext, they are unable to distinguish the underlying plaintext from a choice of two possible plaintexts that were chosen initially by the adversary. Let $\mathsf{SKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathcal{A}$ be any probabilistic polynomial-time adversary.

We formally define the notion of IND-CPA security in Game 2.1 and Definition 2.6.3. The game is initialized with the challenger $\mathcal{C}$ generating a secret key SK and choosing a random bit b. The adversary $\mathcal{A}$ is given the security parameter 1^κ and access to the LoR oracle. The LoR oracle takes as input a pair of plaintexts m_0, m_1 chosen by $\mathcal{A}$. The oracle firsts checks if the two plaintexts are the same size and, if so, returns the encryption of plaintext m_b under key SK to the adversary. If the two plaintexts are not the same size, the oracle returns the failure symbol $\perp$ and terminates. The adversary eventually outputs a bit b' as its guess for the value of b. If $(b' = b)$ then the challenger outputs 1; and otherwise outputs 0.

```
$\mathbf{Exp}^{\mathbf{IND-CPA}}_{\mathcal{A},\mathsf{SKE}}(1^\kappa)$
1 :  $SK \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$
2 :  $b \xleftarrow{\$} \{0,1\}$
3 :  $b' \leftarrow \mathcal{A}^{\mathcal{O}^{LoR(\cdot,\cdot,1^\kappa)}}(1^\kappa)$
4 :  if $(b' = b)$
5 :    return 1
6 :  else
7 :    return 0

$\mathcal{O}^{\mathbf{LoR}}(m_0, m_1, 1^\kappa)$
1 :  if $(|m_0| \neq |m_1|)$
2 :    return $\perp$
3 :  else return $\mathsf{Enc}(m_b, SK)$
```

Game 2.1: IND-CPA for symmetric-key encryption

Definition 2.6.3 (IND-CPA for Symmetric-Key Encryption)**.** *For a SKE scheme* $\mathsf{SKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *we define the advantage of a probabilistic polynomial time adversary* $\mathcal{A}$ *as:*

$$Adv^{\mathbf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{SKE}}(1^\kappa) = \mathbb{P}[\mathbf{Exp}^{\mathbf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{SKE}}(1^\kappa) = 1] - \frac{1}{2}.$$

A SKE scheme SKE *is IND-CPA secure if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *we have:*

$$Adv^{\mathbf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{SKE}}(1^\kappa) \leq \mathsf{negl}(1^\kappa).$$

2.6.4 Security Model for Public-Key Encryption

Analogously, the most widely adopted notions of security for PKE are that of IND-CPA and IND-CCA. We define the notion of IND-CPA security for PKE formally in

Game 2.2 and Definition 2.6.4. The IND-CPA games are defined similarly for SKE and PKE, with the exception that a public key as well as a secret key is generated at the start of the game in the PKE version. The public key is given to the adversary.

```
$\mathbf{Exp}^{\mathbf{IND-CPA}}_{\mathcal{A},\mathsf{PKE}}(1^\kappa)$
1 : $(SK, PK) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$
2 : $b \xleftarrow{\$} \{0,1\}$
3 : $(m_0, m_1) \leftarrow \mathcal{A}(PK, 1^\kappa)$
4 : $c_b \xleftarrow{\$} \mathsf{Enc}(m_b, PK)$
5 : $b' \leftarrow \mathcal{A}(c_b, 1^\kappa)$
6 : if $(b' = b)$
7 :     return 1
8 : else
9 :     return 0
```

Game 2.2: IND-CPA for public-key encryption

Definition 2.6.4 (IND-CPA for Public-key-Encryption)**.** *For a PKE scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *we define the advantage of a probabilistic polynomial time adversary* $\mathcal{A}$ *against a notion* **IND-CPA** *as:*

$$Adv^{\mathbf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(1^\kappa) = \mathbb{P}[\mathbf{Exp}^{\mathbf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(1^\kappa) = 1] - \frac{1}{2}.$$

A PKE scheme PKE *is* **IND-CPA** *secure if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *we have:*

$$Adv^{\mathbf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{PKE}}(1^\kappa) \leq \mathsf{negl}(1^\kappa).$$

2.6.5 Security Model for Ciphertext-Policy Attribute-Based Encryption

The security for CP-ABE is similar to that of SKE. It is required that the ciphertexts leak no information regarding the underlying plaintexts. As there are multiple users in the CP-ABE scheme it is also required that the users cannot collude to learn more information regarding the plaintexts outside the union of their access rights. To mimic the collusion of users in the IND-CPA game for CP-ABE, the adversary is given access to an oracle, KeyGen$(\cdot)$, which takes as input an attribute set of the adversary's choosing and outputs the secret key for that attribute set. This allows the adversary in the game access to multiple decryption keys for different the attribute sets that they query to KeyGen$(\cdot)$. We define a selective game (Game 2.3) and an adaptive game (Game 2.4) for CP-ABE. We define the attribute universe as $\mathcal{U}$.

```
$\mathbf{Exp}^{\mathbf{sIND\text{-}CPA}}_{\mathcal{A}}[\mathsf{CPABE}, 1^\kappa, \mathcal{U}]$ :
1 :  $(\mathbb{A}', st) \xleftarrow{\$} \mathcal{A}(1^\kappa, \mathcal{U})$
2 :  $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{U})$
3 :  $(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(st, PK)$
4 :  if $|m_0| \neq |m_1|$
5 :    return $0$
6 :  $b \xleftarrow{\$} \{0, 1\}$
7 :  $c' \xleftarrow{\$} \mathsf{Encrypt}(m_b, \mathbb{A}', PK)$
8 :  $b' \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(c', st)$
9 :  if $b' = b$
10 :    return $1$
11 :  else
12 :    return $0$

Oracle $\textsc{KeyGen}(A)$
1 :  if $A \not\models \mathbb{A}'$
2 :    return $\mathsf{Keygen}(SK, A, PK)$
3 :  else
4 :    return $\perp$
```

Game 2.3: Selective IND-CPA game for CP-ABE.

In the selective game the adversary receives the security parameter and policy universe as input and chooses their challenge policy $\mathbb{A}'$ before any keys are generated. The challenger then generates the master secret key SK and the public key PK and gives PK to the adversary. The adversary is then allowed a polynomial number of queries to query the oracle, $\textsc{KeyGen}(\cdot)$, for secret keys associated with any attribute set $A \subseteq \mathcal{U}$ of their choosing, except for those such that $A \models \mathbb{A}'$. The adversary then outputs two plaintexts (m_0, m_1), the game is terminated if $|m_0| \neq |m_1|$ to prevent a trivial win for the adversary. The challenger chooses a bit b uniformly at random and encrypts plaintext m_b using the challenge policy chosen by the adversary as input to the $\mathsf{Encrypt}$ algorithm. The adversary is then allowed to query $\textsc{KeyGen}(\cdot)$ under the same conditions as previously and eventually outputs their guess b' for b. The adversary wins the game if $b' = b$.

The adaptive game, Game 2.4, is run similarly to that of the selective game, except that the adversary is allowed to query $\textsc{KeyGen}(\cdot)$ before selecting their challenge policy. The $\textsc{KeyGen}(\cdot)$ oracle keeps track of which attribute sets have been queried before the adversary selects the challenge policy. If the adversary chooses a policy $\mathbb{A}'$ such that $A \models \mathbb{A}'$ for some previously queried attribute set then the game terminates, to avoid a trivial win for the adversary. The adaptive game then proceeds analogously to the selective game.

Definition 2.6.5. *For a CP-ABE scheme* $\mathsf{CPABE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ *we define the advantage of a probabilistic polynomial time adversary* $\mathcal{A}$ *against a notion*

```
$\mathbf{Exp}_{\mathcal{A}}^{\mathbf{IND-CPA}}[\mathsf{CPABE}, 1^\kappa, \mathcal{U}]$ :
1 :  $Q \leftarrow \{\emptyset\}$
2 :  $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{U})$
3 :  $(m_0, m_1, \mathbb{A}', st) \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(PK)$
4 :  if $|m_0| \neq |m_1|$
5 :    return $0$
6 :  for all $A \in Q$ do
7 :    if $A \models \mathbb{A}'$
8 :      return $0$
9 :  $b \xleftarrow{\$} \{0, 1\}$
10 : $c' \xleftarrow{\$} \mathsf{Encrypt}(m_b, \mathbb{A}', PK)$
11 : $b' \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(c', st)$
12 : if $b' = b$
13 :   return $1$
14 : else
15 :   return $0$

Oracle $\textsc{KeyGen}(A)$
1 :  if $A \not\models \mathbb{A}'$
2 :    $Q \leftarrow Q \cup \{A\}$
3 :    return $\mathsf{Keygen}(SK, A, PK)$
4 :  else
5 :    return $\perp$
```

Game 2.4: Adaptive IND-CPA game for CP-ABE.

$X \in \{\mathbf{IND-CPA}, \mathbf{sIND-CPA}\}$ *as:*

$$Adv^{X}_{\mathcal{A},\mathsf{CPABE}}(1^\kappa, \mathcal{U}) = \mathbb{P}[1 \leftarrow \mathbf{Exp}^{X}_{\mathcal{A},\mathsf{CPABE}}(1^\kappa, \mathcal{U})] - \frac{1}{2}.$$

A CP-ABE scheme CPABE *is secure with respect to a notion* $X \in \{\mathbf{IND-CPA}, \mathbf{sIND-CPA}\}$ *if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *we have:*

$$Adv^{X}_{\mathcal{A},\mathsf{CPABE}}(1^\kappa, \mathcal{U}) \leq \mathsf{negl}(1^\kappa).$$

2.6.6 Security Model for Predicate-Based Encryption

The security of PBE is similar to that of CP-ABE, except that it also requires ciphertexts to be *attribute hiding* as well as not leaking any information about the plaintext. As in CP-ABE it is also required that colluding users cannot learn any information from the encrypted data outside of the union of their access rights. In the security game for PBE the adversary is given an oracle, $\textsc{KeyGen}(\cdot)$, to allow them to obtain keys for any predicate $p \in \mathcal{P}$ of their choosing, subject to certain restrictions. The predicate universe is denoted by $\mathcal{P}$ and the attribute universe by $\mathcal{U}$.

In the selective game the adversary receives the security parameter, predicate universe and attribute universe as input and chooses their challenge attribute sets A_0, A_1

```
$\mathbf{Exp}^{\mathbf{sIND-CPA}}_{\mathcal{A}}[\mathsf{PBE}, 1^\kappa, \mathcal{U}, \mathcal{P}]$ :
1 :  $\mu \leftarrow 0$
2 :  $(A_0, A_1, st) \xleftarrow{\$} \mathcal{A}(1^\kappa, \mathcal{U}, \mathcal{P})$
3 :  $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{P})$
4 :  $(m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(st, PK)$
5 :  if $|m_0| \neq |m_1|$
6 :    return $0$
7 :  if $(\mu = 1) \wedge (m_0 \neq m_1)$
8 :    return $0$
9 :  $b \xleftarrow{\$} \{0, 1\}$
10 : $c' \xleftarrow{\$} \mathsf{Encrypt}(m_b, A_b, PK)$
11 : $b' \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(c', st)$
12 : if $b' = b$
13 :   return $1$
14 : else
15 :   return $0$

Oracle $\textsc{KeyGen}(p)$
1 :  if $p(A_0) \neq p(A_1)$
2 :    return $\perp$
3 :  if $p(A_0) = p(A_1) = 1$
4 :      $\mu \leftarrow 1$
5 :  else
6 :    return $\mathsf{Keygen}(SK, p, PK)$
```

Game 2.5: Selective IND-CPA game for PBE.

before any keys are generated. The challenger then generates the master secret key SK and the public key PK and gives PK to the adversary. The adversary is then allowed a polynomial number of queries to query the oracle, $\textsc{KeyGen}(\cdot)$, for secret keys associated with any predicate p of their choosing, except for those such that $p(A_0) \neq p(A_1)$. The adversary then outputs two plaintexts, (m_0, m_1). The game is terminated if $|m_0| \neq |m_1|$ and also if the adversary has queried a predicate p to the oracle $\textsc{KeyGen}(\cdot)$ such that $p(A_0) = p(A_1) = 1$ and the chosen plaintexts are not equal, to prevent a trivial win for the adversary. The challenger chooses a bit b uniformly at random and encrypts plaintext m_b using the challenge attribute set A_b chosen by the adversary, as input to the $\mathsf{Encrypt}$ algorithm. The adversary is then allowed to query $\textsc{KeyGen}(\cdot)$ under the same conditions as previously and eventually outputs their guess b' for b. The adversary wins the game if $b' = b$.

The adaptive game, Game 2.6, is run similarly to that of the selective game, except that the adversary is allowed to query $\textsc{KeyGen}(\cdot)$ before selecting their challenge attribute set. The adversary is able to query $\textsc{KeyGen}(\cdot)$ for a key corresponding to any predicate of their choosing. A set Q is generated by $\textsc{KeyGen}(\cdot)$ to keep track of the queries the adversary has made to the oracle. After a polynomial number of queries the adversary selects their challenge attribute sets, A_0, A_1, along with two plaintexts m_0, m_1. If the sizes of the two plaintexts are not equal then the game terminates, to

```
$\mathbf{Exp}_{\mathcal{A}}^{\mathbf{IND-CPA}}[\mathsf{PBE}, 1^\kappa, \mathcal{U}, \mathcal{P}]$ :
    $Q \leftarrow \emptyset, \mathsf{chall} \leftarrow 0$
    $(PK, SK) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{P})$
    $(A_0, A_1, m_0, m_1, st) \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(1^\kappa, PK)$
    if $|m_0| \neq |m_1|$
        return 0
    if there exists $p \in Q : p(A_0) \neq p(A_1)$
        return 0
    if there exists $p \in Q : (p(A_0) = p(A_1) = 1) \wedge (m_0 \neq m_1)$
        return 0
    $b \xleftarrow{\$} \{0, 1\}$
    $\mathsf{chall} \leftarrow 1$
    $c' \xleftarrow{\$} \mathsf{Encrypt}(m_b, A_b, PK)$
    $b' \xleftarrow{\$} \mathcal{A}^{\textsc{KeyGen}(\cdot)}(c', st)$
    if $b' = b$
        return 1
    else
        return 0

Oracle $\textsc{KeyGen}(p)$
    if $\mathsf{chall} = 0$
        $Q \leftarrow Q \cup p$
        return $\mathsf{Keygen}(SK, p, PK)$
    else
        if $p(A_0) \neq p(A_1)$
            return $\perp$
        if $(p(A_0) = p(A_1) = 1) \wedge (m_0 \neq m_1))$
            return $\perp$
        else
            return $\mathsf{Keygen}(SK, p, PK)$
```

Game 2.6: Adaptive IND-CPA game for PBE.

prevent the adversary from a trivial win. There are two other conditions that also need to be met, that is, for any predicate $p \in Q$ it is required that $p(A_0) = p(A_1)$. Lastly it is required for any predicate p such that $p(A_0) = p(A_1) = 1$ we have that $m_0 = m_1$. These requirements are to prevent a trivial win for the adversary. The adaptive game then proceeds analogously to the selective game.

Definition 2.6.6. *For a PBE scheme* PBE = (Setup, Encrypt, KeyGen, Decrypt) *we define the advantage of a probabilistic polynomial time adversary* $\mathcal{A}$ *against a notion* $X \in \{\mathbf{IND-CPA}, \mathbf{sIND-CPA}\}$ *as:*

$$Adv_{\mathcal{A},\mathsf{PBE}}^{X}(1^\kappa, \mathcal{U}, \mathcal{P}) = \mathbb{P}[1 \leftarrow \mathbf{Exp}_{\mathcal{A},\mathsf{PBE}}^{X}(1^\kappa, \mathcal{U}, \mathcal{P})] - \frac{1}{2}.$$

A PBE scheme PBE *is secure with respect to a notion* $X \in \{\mathbf{IND-CPA}, \mathbf{sIND-CPA}\}$ *if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *we have:*

$$Adv_{\mathcal{A},\mathsf{PBE}}^{X}(1^\kappa, \mathcal{U}, \mathcal{P}) \leq \mathsf{negl}(1^\kappa).$$

2.6.7 Security Model for Broadcast Encryption

The security of BE requires that a user be unable to distinguish ciphertexts that are encrypted under a group of identities that the user does not belong to. In the security game for BE the adversary is given access to an oracle, ADD(·), to allow them to obtain keys for any identity outside of the set of challenge identities. The identity universe is denoted by $\mathcal{U}$.

```
Exp_A^{IND-CPA}[BE, 1^κ, U] :
 1 :  $\mathsf{chall} \leftarrow 0, U \leftarrow \emptyset$
 2 :  $(PK, SK) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, \mathcal{U})$
 3 :  $(S^*, m_0, m_1, st)\mathcal{A}^{\textsc{Add}(\cdot)}(1^\kappa, PK, \mathcal{U})$
 4 :  $b \xleftarrow{\$} \{0,1\}$
 5 :  $c^* \xleftarrow{\$} \mathsf{Encrypt}(m_b, S^*, PK)$
 6 :  $\mathsf{chall} = 1$
 7 :  $b' \xleftarrow{\$} \mathcal{A}^{\textsc{Add}(\cdot)}(c^*, st)$
 8 :  if $b' = b$
 9 :    return 1
10 :  else
11 :    return 0

Oracle ADD(u)
 1 :  if $\mathsf{chall} = 0$
 2 :    $U \leftarrow U \cup u$
 3 :    return $SK_u \xleftarrow{\$} \mathsf{Add}(SK, u)$
 4 :  else
 5 :    if $u \in S^*$
 6 :      return $\perp$
 7 :    else
 8 :      return $SK_u \xleftarrow{\$} \mathsf{Add}(SK, PK, u)$
```

Game 2.7: Adaptive IND-CPA game for BE.

The game is initialized by setting the value of chall to 0. The challenger then runs KeyGen to generate the keys for the system, (PK, SK). The public key PK is given to the adversary, who is then also given access to the oracle, ADD(·). The adversary can make a polynomial number of queries to this oracle, before outputting their challenge identity set, S^*, and a pair of plaintexts (m_0, m_1). The set U keeps track of the queries made to the oracle, if there exists an identity $u \in U$ such that $u \in S^*$ then the game is terminated to prevent a trivial win for the adversary. The challenger selects a bit b uniformly at random and encrypts the plaintext m_b under the set of identities S^*, to create the challenge ciphertext c^*, which is sent to the adversary. The adversary is then allowed another round of queries to ADD(·) subject to the same restrictions as before. Eventually the adversary outputs their guess b' for b and wins the game if $b' = b$.

Definition 2.6.7. *For a BE scheme* BE = (KeyGen, Add, Encrypt, Decrypt) *we define the advantage of a probabilistic polynomial time adversary* $\mathcal{A}$ *against the notion IND-CPA as:*

$$Adv_{\mathcal{A},\mathsf{BE}}^{\mathbf{IND-CPA}}(1^\kappa, \mathcal{U}) = \mathbb{P}[1 \leftarrow \mathbf{Exp}_{\mathcal{A},\mathsf{BE}}^{\mathbf{IND-CPA}}(1^\kappa, \mathcal{U})] - \frac{1}{2}.$$

A BE scheme BE *is secure with respect to the notion IND-CPA if for all probabilistic polynomial time adversaries $\mathcal{A}$ we have:*

$$Adv^{\mathbf{IND-CPA}}_{\mathcal{A},\mathsf{BE}}(1^{\kappa},\mathcal{U}) \leq \mathsf{negl}(1^{\kappa}).$$

2.6.8 Security Model for Verifiable Computation

The security of VC requires that an incorrectly computed result cannot be verified as a correct result by the client. In the security game for VC the adversary is given access to an oracle $\textsc{ProbGen}(\cdot)$, to allow them to obtain the encoded inputs for a function f and an input of their choosing.

```
$\mathbf{Exp}^{\mathbf{VERIFY}}_{\mathcal{A}}[\mathsf{VC}, 1^{\kappa}, f]$ :
    $(EK_f, SK_f) \xleftarrow{\$} \mathsf{KeyGen}(1^{\kappa}, f)$
    $(x, st) \xleftarrow{\$} \mathcal{A}^{\textsc{ProbGen}(\cdot)}(1^{\kappa}, f, EK_f)$
    $(\sigma_{f,x}, VK_{f,x}) \xleftarrow{\$} \mathsf{ProbGen}(x, SK_f)$
    $\theta_{f(x)} \xleftarrow{\$} \mathcal{A}^{\textsc{ProbGen}(\cdot)}(\sigma_{f,x}, st)$
    $y \leftarrow \mathsf{Verify}(\theta_{f(x)}, VK_{f,x}, SK_f)$
    if $(y \neq \perp) \wedge (y \neq f(x))$
        return $1$
    else
        return $0$

Oracle $\textsc{ProbGen}(z)$
    $(\sigma_{f,z}, VK_{f,z}) \xleftarrow{\$} \mathsf{ProbGen}(z, SK_f)$
    return $\sigma_{f,z}$
```

Game 2.8: Adaptive VERIFY game for VC.

The game is initialized by the challenger running Setup to generate key pair (EK_f, SK_f) for a function f. The key EK_f is given to the adversary, along with access to oracle $\textsc{ProbGen}(\cdot)$. The adversary can make a polynomial number of queries to this oracle before choosing a challenge input x. The challenger then runs ProbGen using the challenge input x to generate the encoded input $\sigma_{f,x}$, which is given to the adversary. This algorithm also generates the verification key $VK_{f,x}$, however this is withheld from the adversary to prevent them from verifying their computations themselves. The adversary is then allowed another round of queries to the oracle $\textsc{ProbGen}(\cdot)$, and eventually outputs an encoded result, $\theta_{f,x}$, that they believe to be an incorrect computation of $f(x)$ that will be verified as correct by the client. The adversary wins the game if the experiment outputs 1, meaning $\theta_{f,x}$ encodes an incorrect result that is accepted as correct.

Definition 2.6.8. *For a VC scheme* $\mathsf{VC} = (\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *we define the advantage of a probabilistic polynomial time adversary $\mathcal{A}$ against the notion*

VERIFY as:

$$Adv^{\mathbf{VERIFY}}_{\mathcal{A},\mathsf{VC}}(1^\kappa, f) = \mathbb{P}[1 \leftarrow \mathbf{Exp}^{\mathbf{VERIFY}}_{\mathcal{A},\mathsf{VC}}(1^\kappa, f)] - \frac{1}{2}.$$

A VC scheme VC *is secure with respect to the notion VERIFY if for all probabilistic polynomial time adversaries* $\mathcal{A}$ *we have:*

$$Adv^{\mathbf{VERIFY}}_{\mathcal{A},\mathsf{VC}}(1^\kappa, f) \leq \mathsf{negl}(1^\kappa).$$

2.6.9 Security Model for Verifiable Delegable Computation

The security of VDC is described similarly to that of VC (Section 2.6.8). It ensures that an incorrectly computed result by a server cannot be verified as correct. In this setting the verification can be done publicly. In the security game for VDC the adversary is given access to the following oracles: FnInit$(\cdot)$, Register$(\cdot)$ and Certify$(\cdot,\cdot,\cdot,\cdot)$, which are run as per the algorithms FnInit, $\mathsf{Register}$ and $\mathsf{Certify}$ respectively. We denote the adversary having access to these three oracles as $\mathcal{A}^{\mathcal{O}}$.

```
$\mathbf{Exp}^{\mathbf{PUB-VERIFY}}_{\mathcal{A}}[\mathsf{VDC}, 1^\kappa, \mathcal{F}]$ :
1 :  $(MK, PP) \xleftarrow{\$} \mathsf{Setup}(1^\kappa, \mathcal{F})$
2 :  $(f, X, \{\ell(x_j)\}_{x_j \in X}, st) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(1^\kappa, \mathcal{F}, PP)$
3 :  $PK_f \xleftarrow{\$} \mathsf{FnInit}(f, MK, PP)$
4 :  $(\sigma_{f,X}, VK_{f,X}, RK_{f,X}) \xleftarrow{\$} \mathsf{ProbGen}(f, \{\ell(x_j))\}_{x_j \in X}, PK_f, PP)$
5 :  $\theta \xleftarrow{\$} \mathcal{A}^{\mathcal{O}}(\sigma_{f,X}, VK_{f,X}, RK_{f,X}, PK_f, PP, st)$
6 :  $(RT_{f,X}, \tau_\theta) \leftarrow \mathsf{BVerif}(\theta, VK_{f,X}, PP)$
7 :  $y_{f(X)} \leftarrow \mathsf{Retrieve}(RT_{f(X)}, \tau_\theta, VK_{f,X}, RK_{f,X}, PP)$
8 :  if $((y_{f(X)}, \tau_\theta) \neq (\perp, reject)) \wedge (y_{f(X)} \neq f(X))$
9 :      return 1
10 : else
11 :     return 0
```

Game 2.9: Adaptive PUB-VERIFY game for VDC.

The game is initialized by the challenger running Setup to generate the key MK and the public parameters, PP. These public parameters are given to the adversary. The adversary can then make a polynomial number of queries to the specified oracles before outputting the challenge inputs: the function f, the input values X and the labels $\ell(x_j)$ for each input value in X. The challenger then runs FnInit and $\mathsf{ProbGen}$ to generate the key for the function, PK_f and the encoded input $\sigma_{f,X}$, the verification key $VK_{f,X}$ and the retrieval key $RK_{f,X}$. The adversary is given these four values and access to the specified oracles. Eventually the adversary outputs their challenge response θ, which they believe to be an incorrectly computed result for $f(X)$ that will be successfully verified. The challenger then runs BVerif and $\mathsf{Retrieve}$ to obtain $y_{f(X),\tau_\theta}$. The adversary wins if θ encodes an incorrect result that is verified as correct i.e. if $(y_{f(X)}, \tau_\theta) \neq (\perp, reject)$ and $y_{f(X)} \neq f(X)$.

Definition 2.6.9. *For a VDC scheme* $\mathsf{VDC} = (\mathsf{Setup}, \mathsf{FnInit}, \mathsf{Register}, \mathsf{Certify}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *we define the advantage of a probabilistic polynomial time adversary $\mathcal{A}$ against the notion PUB-VERIFY as:*

$$Adv_{\mathcal{A},\mathsf{VDC}}^{\mathbf{PUB-VERIFY}}(1^\kappa, \mathcal{F}) = \mathbb{P}[1 \leftarrow \mathbf{Exp}_{\mathcal{A},\mathsf{VDC}}^{\mathbf{PUB-VERIFY}}(1^\kappa, \mathcal{F})] - \frac{1}{2}.$$

A VDC scheme VDC *is secure with respect to the notion PUB-VERIFY if for all probabilistic polynomial time adversaries $\mathcal{A}$ we have:*

$$Adv_{\mathcal{A},\mathsf{VDC}}^{\mathbf{PUB-VERIFY}}(1^\kappa, \mathcal{F}) \leq \mathsf{negl}(1^\kappa).$$

2.7 Summary

In this chapter we presented the notation that we use throughout the thesis. We also define several encryption schemes, along with their security models, and cryptographic primitives that we make use of in the remainder of the thesis.

Chapter 3

Searchable Encryption

Contents

This chapter introduces searchable encryption and reviews the relevant literature.

3.1 Introduction

Searchable Encryption allows an entity to search over encrypted data that they have outsourced to a remote server. In its simplest form, this involves locating encrypted data items on the server that contain a specified keyword. A searchable encryption scheme generally involves three entities:

- **Server**: A remote *server* (such as a pay-as-you-go cloud server) on which encrypted data is stored. We assume that the data stored on the server is of a sensitive nature, so it is undesirable to reveal it to any third parties, including the server. Note that in some cases, for example medical records, it may be illegal

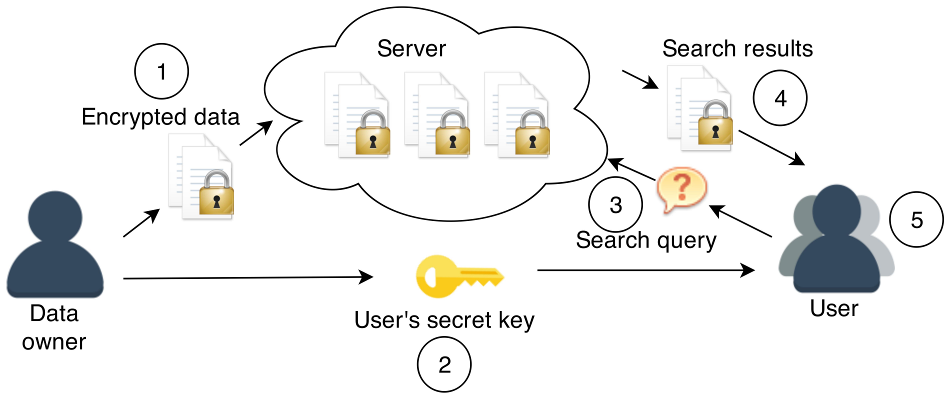

Figure 3.1: Searchable Encryption model.

to reveal the records. We thus assume that the data items remain encrypted at all times when on the server.

- **Data Owner**: This entity initiates the scheme and holds the master secret key (used to generate search queries in the single user case or user secret keys in the multi-user case[1]). The *data owner* is always able to submit search queries to the server[2] (referred to as *reading* data), as well as encrypt data to be stored on the server (referred to as *writing* data). The data owner is able to control which additional users (if any) are able to read or write data in the scheme.
- **User(s)**: SE schemes contain a varying numbers of *users* that have the capability to either write data, read data, or do both of these tasks, as determined by the data owner.

The data owner associates metadata with each data item to be encrypted, usually consisting of keywords contained within the data item or keywords describing the data item's contents along with an identifier for the data item, and then creates an encrypted index using this metadata. The set of data items is encrypted separately using a standard encryption scheme, so the focus of searchable encryption is how the encrypted index is created. In this work we do not consider the retrieval of the encrypted data items from the server. The encrypted index and encrypted data items are outsourced to the server and, in order to search for a keyword in the encrypted index, the data owner (or a user) produces a search query which the server uses to locate the relevant data items.

[1]In the multi-user setting the data owner is enrolled as user and generates a user secret key in order to produce search queries.

[2]In the multi-user setting this is done by the owner enrolling as a user.

Figure 3.1 depicts the basic architecture of an SE scheme, which consists of the following five steps:

1. **Data encryption.** The data to be outsourced to the server is encrypted by the data owner (or a user, depending on the scenario) and sent to the server. In general the encrypted data will consist of two parts:

   (a) *Encrypted data items*: The data items the data owner wishes to outsource to the server are typically encrypted using symmetric key cryptography. Each encrypted data item is identified by a unique identifier (ID).

   (b) *Secure index*: A *secure index* is created which enables the server to use a *search token* to learn the IDs of the relevant encrypted data items and locate them on the server.

2. **User's secret key transfer.** The data owner generates a secret key for a user and transfers it to them. The user is able generate search queries using their secret key.

3. **Search query.** A user (or data owner) generates a search token for a keyword ω they wish to search for in the encrypted data and sends it to the server, allowing it to ascertain which encrypted data items satisfy their query.

4. **Search results.** The search results returned to the user can take two forms:

   (a) IDs of the encrypted data items that satisfy the query associated with the search token;

   (b) Encrypted data items that satisfy the query associated with the search token.

   The former response is advantageous over the latter in situations where the user does not require access to the actual data item (during statistical analysis of the encrypted data, for example) since it involves lower communication costs. The latter response involves the server locating the encrypted data items satisfying the query by locating them using the set of IDs, which increases the search time. In this thesis, we consider schemes which generate search results consisting of IDs of encrypted data items that satisfy the query associated with the search token only; the retrieval of the actual data items is beyond the scope of this thesis.

5. **Decryption.** If search results contain encrypted data items, a decryption key needs to be obtained from the data owner in order to recover the data items.

Many different SE schemes have been proposed, with a range of properties. There are two main branches of research in the field of searchable encryption: symmetric searchable encryption and public-key searchable encryption. In Section 3.4.3 and 3.4.2 we give a chronological overview of the SE schemes in the literature within these two branches of research. Here we give some general definitions of SE schemes.

In these definitions we consider the model outlined in Section 3.1 where a data owner is outsourcing some encrypted data to a server but wishes to maintain the ability to search over the data, or control who is able to search the data.

Definition 3.1.1 (Symmetric Searchable Encryption (SSE))**.** *Let $\mathcal{K}$ be the key space, Ω be the keyword space, $\mathcal{ID}$ be the data item identifier space, $\Delta = \Omega \times \mathcal{ID}$ be the metadata space and $\mathcal{D} \subseteq \Delta$ be the set of metadata associated with data items to be indexed. A searchable symmetric encryption scheme typically consists of the following four algorithms (we assume the data items themselves are encrypted separately using a secure encryption scheme):*

- $SK \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$*: A probabilistic algorithm run by the data owner that takes as input the security parameter $1^\kappa \in \mathbb{N}$ and outputs a secret key, $SK \in \mathcal{K}$.*
- $\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, SK)$*: This algorithm can be probabilistic or deterministic as required and it is run by the data owner. It takes as input the set of metadata $\mathcal{D}$ associated with each data item along with the secret key SK and outputs a secure index $\mathcal{I}_\mathcal{D}$.*
- $T_\omega \leftarrow \mathsf{Query}(\omega, SK)$*: A deterministic algorithm run by the data owner to produce a search query. It takes as input a keyword $\omega \in \Omega$ along with SK and outputs a search query T_ω.*
- $R_\omega \leftarrow \mathsf{Search}(\mathcal{I}_\mathcal{D}, T_\omega)$*: A deterministic algorithm run by the server to produce the search results. It takes as input the secure index $\mathcal{I}_\mathcal{D}$ and a search query T_ω and outputs the search results R_ω.*

Definition 3.1.2 (Correctness of SSE)**.** *An SSE scheme,* $\mathsf{SSE} = (\mathsf{KeyGen}, \mathsf{BuildIndex}, \mathsf{Query}, \mathsf{Search})$*, is correct if for all $\kappa \in \mathbb{N}$, for all $SK \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$, for all $\mathcal{D} \subseteq \Delta$, for all $\mathcal{I}_\mathcal{D} \leftarrow \mathsf{BuildIndex}(\mathcal{D}, SK)$, for all $\omega \in \Omega$:*

$$\begin{aligned}\mathbb{P}[&SK \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa),\\ &\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, SK),\\ &T_\omega \leftarrow \mathsf{Query}(\omega, SK),\\ &R_\omega \leftarrow \mathsf{Search}(\mathcal{I}_\mathcal{D}, T_\omega)] = 1.\end{aligned}$$

In Defintion 3.1.1 we do not specify any additional users in the system. In the literature it is standard for SSE to be defined for a single entity encrypting the data and creating the search tokens, however for Public-Key Searchable Encryption (Definition 3.1.3) it is standard to assume that more than one entity could be creating the encrypted data because it is generated using a public key. The suitability for the different types of searchable encryption to different scenarios and users are discussed further in Chapter 4.

Definition 3.1.3 (Public Key Searchable Encryption (PKSE))**.** *Let $\mathcal{K}$ be the key space, Ω be the keyword space, $\mathcal{ID}$ be the data item identifier space and $\Delta = \Omega \times \mathcal{ID}$ be the metadata space. A public key searchable encryption scheme typically consists of the following four algorithms (we assume the data items themselves are encrypted separately using a secure encryption scheme):*

- $(PK, SK) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$*: A probabilistic algorithm run by the data owner that takes as input the security parameter $1^\kappa \in \mathbb{N}$ and outputs a public and secret key pair, $(PK, SK) \in \mathcal{K}$.*
- $\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, PK)$*: This algorithm can be probabilistic or deterministic as required and is run by either the data owner or an additional user. It takes as input the set of metadata $\mathcal{D} \subseteq \Delta$ associated with each data item to be indexed along with the public key PK and outputs a secure index $\mathcal{I}_\mathcal{D}$.*
- $T_\omega \leftarrow \mathsf{Query}(\omega, SK)$*: A probabilistic algorithm run by the data owner to produce a search query. It takes as input a keyword $\omega \in \Omega$ that the data owner or user wishes to search for along with SK and outputs a search query T_ω.*
- $R_\omega \leftarrow \mathsf{Search}(PK, \mathcal{I}_\mathcal{D}, T_\omega)$*: A deterministic algorithm run by the server to produce the search results. It takes as input the secure index $\mathcal{I}_\mathcal{D}$ and a search query T_ω and outputs the search results R_ω.*

Definition 3.1.4 (Correctness of PKSE)**.** *A PKSE scheme is correct if for all $\kappa \in \mathbb{N}$, for all $(PK, SK) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$, for all $\mathcal{D} \subseteq \Delta$, for all $\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, SK)$, for all $\omega \in \Omega$:*

$$\begin{aligned}\mathbb{P}[(PK, SK) &\xleftarrow{\$} \mathsf{KeyGen}(1^\kappa),\\ \mathcal{I}_\mathcal{D} &\xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, PK),\\ T_\omega &\leftarrow \mathsf{Query}(\omega, SK),\\ R_\omega &\leftarrow \mathsf{Search}(\mathcal{I}_\mathcal{D}, T_\omega)] = 1 - negl(\kappa).\end{aligned}$$

Multi-user searchable symmetric encryption (mSSE) was introduced by Curtmola et al. [55]. Work in SSE previous to this considered the single-user setting where only the data owner was able to query the encrypted data. The work of [55] considers the extension of SSE to allow multiple users to query the encrypted data as determined by the data owner. It supports a dynamic set of users where revocation and addition of users is controlled by the data owner.

In the following definition we specify which entity runs each algorithm, we assume a *data owner* that owns the data and sets up the system and additional users that are able to generate search queries. These entities are defined in more detail in Section 3.1. Here we assume an honest-but-curious model where the server is trusted to update the server state, st_S (see Definition 3.1.5), after a user is revoked.

Definition 3.1.5 (Multi-user Searchable Encryption (MSSE))**.** *Let $\mathcal{K}$ be the keyspace, Ω be the keyword space, $\mathcal{ID}$ be the data item identifier space, $\Delta = \Omega \times \mathcal{ID}$ be the metadata space and $\mathcal{U}$ be the universe of users. An MSSE scheme consists of the following six algorithms:*

- $SK \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$*: A probabilistic algorithm run by the data owner that takes as input the security parameter $1^\kappa \in \mathbb{N}$ and outputs a secret key $SK \in \mathcal{K}$.*
- $(\mathcal{I}_\mathcal{D}, st_O, st_S) \xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, \mathcal{G}, SK)$*: A probabilistic algorithm run by the data owner. It takes as input the set of metadata $\mathcal{D}$ associated with each data item along with the secret key SK and a set $\mathcal{G} \subseteq \mathcal{U}$ of authorised users and outputs a secure index $\mathcal{I}_\mathcal{D}$ along with a server state st_S and a data owner state st_O.*
- $(K_u, st_O, st_S) \xleftarrow{\$} \mathsf{AddUser}(SK, st_O, u)$*: A probabilistic algorithm run by the data owner to add a user to the system. It takes as input the new user's identity u and the data owner's state st_O along with the secret key SK and outputs the new user's cryptographic key $K_u \in \mathcal{K}$.*
- $(st_O, st_S) \leftarrow \mathsf{Revoke}(SK, st_O, u)$*: A probabilistic algorithm run by the data owner to revoke a user from the system. It takes as input the identity of the user that is*

to be revoked u and the data owner's state st_O along with the secret key SK and outputs updated data owner and server states, st_O, st_S.

- $T_\omega \leftarrow \mathsf{Query}(\omega, K_u)$*: A deterministic algorithm run by a user (note that the data owner can also be enrolled as a user) to produce a search query. It takes as input a keyword $\omega \in \Omega$ that the user wishes to search for along with their secret key K_u and st_S (which is retrieved from the server). It outputs a search query T_ω.*

- $R_\omega \leftarrow \mathsf{Search}(st_S, \mathcal{I}_\mathcal{D}, T_\omega)$*: A deterministic algorithm run by the server to produce the search results. It takes as input the secure index $\mathcal{I}_\mathcal{D}$ and a search query T_ω along with the server state st_S and outputs the search results R_ω, which is either the set of data items containing the keyword ω, denoted D_ω, or $\perp$.*

Definition 3.1.6 (Correctness of MSSE)**.** *An MSSE scheme is correct for all $\kappa \in \mathbb{N}$, for all $\mathcal{G} \subseteq \mathcal{U}$ and $u \in \mathcal{G}$, for all SK output by $\mathsf{KeyGen}(1^\kappa)$, for all $\mathcal{D} \subseteq \Delta$, for all $\mathcal{I}_\mathcal{D}$ output by $\mathsf{Buildindex}(\mathcal{D}, \mathcal{G}, SK)$, for all $\omega \in \Omega$, for all K_u output by $\mathsf{AddUser}(SK, st_O, u)$, we have that:*

$$\begin{aligned} \mathbb{P}[SK &\xleftarrow{\$} \mathsf{KeyGen}(1^\kappa), \\ (\mathcal{I}_\mathcal{D}, st_O, st_S) &\xleftarrow{\$} \mathsf{BuildIndex}(\mathcal{D}, SK), \\ (K_u, st_O, st_S) &\xleftarrow{\$} \mathsf{AddUser}(SK, st_O, u), \\ T_\omega &\leftarrow \mathsf{Query}(\omega, K_u), \\ D_\omega &\leftarrow \mathsf{Search}(st_S, \mathcal{I}_\mathcal{D}, T_\omega)] = 1. \end{aligned}$$

Definition 3.1.7 (Verifiable Searchable Encryption (VSE))**.** *Let $\mathcal{K}$ be the keyspace, let $\mathcal{U}$ be the universe of users and let A be the universe of attributes. We define a pre-index $\delta(\mathcal{D})$ of a set of data items $\mathcal{D}$ as the set of attributes associated with each data item, such as keywords contained in each data item. A VSE scheme comprises the following algorithms:*

- $(SK, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, A)$: *A probabilistic algorithm run by the data owner and takes as input the security parameter and a universe of attributes (keywords and data values). It outputs the data owner's master secret key SK that is used for further administrative tasks and public parameters PP, both of which are provided to the remaining algorithms where required.*

- $\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\delta(\mathcal{D}), SK, PP)$: *A probabilistic algorithm run by the data owner that takes as input the pre-index of the data $\delta(\mathcal{D}) \subseteq A$, the master secret key SK, and outputs a searchable index $\mathcal{I}_\mathcal{D}$ for the pre-index $\delta(\mathcal{D})$.*

- $T_\omega \xleftarrow{\$} \mathsf{Query}(\omega, SK, PP)$: *A deterministic or probabilistic algorithm run by a data owner using its key SK. It generates a search query token T_ω for a keyword ω.*

- $R_\omega \leftarrow \mathsf{Search}(\mathcal{I}_\mathcal{D}, T_\omega, PP)$: *A deterministic algorithm run by the server to execute a search query T_ω on the index $\mathcal{I}_\mathcal{D}$. It generates a result R_ω which is returned to the data owner.*

- $\{0,1\} \leftarrow \mathsf{Verify}(R_\omega)$: *A deterministic algorithm run by the data owner, which outputs 0 or 1.*

Definition 3.1.8 (Correctness of VSE)**.** *A VSE scheme is correct if for all $\kappa \in \mathbb{N}$, for all $(SK, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, A)$, for all $\delta(\mathcal{D}) \subseteq A$, for all $\mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\delta(\mathcal{D}), SK, PP)$ and for all $T_\omega \xleftarrow{\$} \mathsf{Query}(\omega, SK, PP)$ we have that:*

$$\begin{aligned}\mathbb{P}[(SK, PP) &\xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, A),\\ \mathcal{I}_\mathcal{D} &\xleftarrow{\$} \mathsf{BuildIndex}(\delta(\mathcal{D}), SK, PP),\\ T_\omega &\leftarrow \mathsf{Query}(\omega, SK, PP),\\ R_\omega &\leftarrow \mathsf{Search}(st_S, \mathcal{I}_\mathcal{D}, T_\omega),\\ 1 &\leftarrow \mathsf{Verify}(D_\omega)] = 1.\end{aligned}$$

3.2 Security of Searchable Encryption

One of the main goals of any data encryption scheme is to provide confidentiality of the encrypted data, that is given a ciphertext an adversary should be able to learn nothing regarding the corresponding plaintext. This is called *semantic security*. A paradigm called oblivious RAM (ORAM) [70] can be used to construct an SE scheme which leaks nothing about the underlying plaintexts or the access pattern to an adversary, except the volume of data returned per query. However, ORAM is not currently practical to use on large datasets. SE schemes are generally designed to leak some information, such as the access and search patterns, in order to achieve a more efficient scheme.

We define the access and search patterns for SE precisely as follows:

Definition 3.2.1. *(Access pattern) For a sequence of search queries $\Omega = (T_{\omega_1}, ..., T_{\omega_q})$ where ω_i and ω_j are not necessarily distinct for $i \neq j$, along with an index $\mathcal{I}_\mathcal{D}$, the access pattern is defined as:*

$$AP(\mathcal{I}_\mathcal{D}, \Omega) = (\mathcal{R}_{\omega_1}, ..., \mathcal{R}_{\omega_q}).$$

The access pattern is the set of search results produced by the sequence of queries Ω.

Definition 3.2.2. *(Search pattern) For a sequence of q search queries $\Omega = (T_{\omega_1}, ..., T_{\omega_q})$ where $1 \leq i, j \leq q$ and ω_i and ω_j are not necessarily distinct for $i \neq j$, the search pattern is defined as a $q \times q$ symmetric binary matrix $SP(\mathcal{I}_{\mathcal{D}}, \Omega)$ where:*

$$SP(\mathcal{I}_{\mathcal{D}}, \Omega)_{i,j} = 1 \iff T_{\omega_i} = T_{\omega_j}.$$

Intuitively, the search pattern reveals when the ith and jth queries are the same, which happens when queries are issued for the same keyword.

Definition 3.2.3. *(Setup leakage) For an index $\mathcal{I}_{\mathcal{D}}$ we define the setup leakage as:*

$$\mathcal{L}_{setup}(\mathcal{I}_{\mathcal{D}}) = |\mathcal{I}_{\mathcal{D}}|,$$

(the size of the index). Intuitively the setup leakage is all the information that is leaked from the index alone. This may include other leakage depending on the type of encryption used.

Definition 3.2.4. *(Search leakage) For an index $\mathcal{I}_{\mathcal{D}}$ along with q search queries $\Omega = (T_{\omega_1}, ..., T_{\omega_q})$ we define the search leakage as:*

$$\mathcal{L}_{search}(\mathcal{I}_{\mathcal{D}}, \Omega) = \big(AP(\mathcal{I}_{\mathcal{D}}, \Omega), SP(\mathcal{I}_{\mathcal{D}}, \Omega)\big).$$

Intuitively the search leakage represents all the information that is leaked from the index and the search queries during the search procedure.

3.3 Adversarial models

There are three main adversarial models considered in the SE literature. The server is viewed as the adversary in an SE scheme and the adversarial models are defined as follows:

Definition 3.3.1 (Honest-but-curious)**.** *An honest-but-curious server will follow the protocol set out for it but it may try and learn additional information about the encrypted data from the information it has access to. The server will not try to actively manipulate the protocol in any way in order to gain access to more information outside of what it is authorized to know.*

Definition 3.3.2 (Semi honest-but-curious)**.** *A semi honest-but-curious server may not follow the protocol that has been set out for it. For example, a server may only* *compute a fraction of a search or return an incomplete set of search results in order to conserve its own resources (such as computation power or bandwidth).*

Definition 3.3.3 (Malicious)**.** *A malicious server may actively manipulate or deviate from the protocol and return incorrect and/or incomplete search results to the user.*

3.4 Literature Review

In this section we discuss the relevant literature in the area of searchable encryption. For a detailed review of the literature in verifiable searchable encryption and multi-user searchable encryption, please see Sections 5.1.1 and 6.1.1 respectively.

3.4.1 Oblivious RAM

Oblivious RAM (ORAM), introduced by Goldreich and Ostrovsky [70], allows a user to retrieve data items from a server without revealing to the server which data items were retrieved (access pattern). In order to mask the access pattern each access to the data is designed to be indistinguishable to an adversary. Suppose the user stores n data items on the server, in order to achieve the masking of the access patterns the user is required to store an extra $\sqrt{n}$ dummy data items and a *shelter* storing another $\sqrt{n}$ data items. First off a random permutation π is applied to the data items at position $(1, ..., n + \sqrt{n})$, i.e. everything but the shelter. In order to access the ith data item, the shelter is scanned to check whether the ith data item is in the shelter (this will not be the case on the first access, however after each access the contents of the access is stored in the shelter). The items in the shelter are accessed in a predetermined order, regardless of whether the desired data item has been located. If i is not found in the shelter then it is retrieved by calculating $\pi(i)$, which is the location of the desired data item. If i was located in the shelter then the next dummy data item in the permuted memory is accessed at $\pi(n + j)$, where j is the current step in the access. In either case the retrieved data item, the dummy one or the actual one, is written to the shelter, by scanning all of the items in the shelter again. The contents of the shelter is returned to the user. This ORAM solution is called the quadratic solution. Many other methods for achieving ORAM have been put forward in order to enhance its efficiency [29, 45, 56, 57, 66]. Using ORAM to achieve searchable encryption is one of the most secure methods for searchable encryption. However, due to the high computational costs incurred by the multiple accesses per search it does not scale well to encrypted search over a large database.

3.4.2 Public Key Searchable Encryption

The first public key searchable encryption scheme was presented by Boneh et al. [33]. Their scheme, entitled public key encryption with keyword search (PEKS), allows many users to encrypt data using the data owner's public key, to be stored on a remote server. The data owner is then able to use their private key to produce search tokens to locate specific items of interest in the encrypted data. The motivating example given in the paper is that of a secure email gateway. The gateway intercepts all incoming mail, which has been encrypted using the data owner's public key. The data owner generates search queries which they give to the gateway to allow it to detect when a certain encrypted message contains a specific keyword. The gateway can then route the email accordingly. The encrypted data is made up of two parts, the first being a public key encryption of the data item which is then appended to the second part which consists of a list of m keywords that have been encrypted separately using the PEKS scheme as follows, where PK_A is the public key of the data owner and PEKS is the encrypt algorithm of the PEKS scheme:

$$E_{PK_A}(M)||\mathsf{PEKS}(PK_A,\omega_1)||...||\mathsf{PEKS}(PK_A,\omega_m).$$

The specific PEKS scheme in [33] is based on a variant of the Decision Diffie-Hellman assumption and uses a random oracle in the construction. They define an adaptive notion of semantic security which ensures that an adversary cannot learn anything from the PEKS ciphertexts without a search query, that is, given a PEKS ciphertext and the choice of two keywords the adversary is unable to distinguish which keyword is encrypted in the PEKS ciphertext with probability negligibly greater than that of a random guess. We refer to this notion of security as IND-PEKS-CKA2. It is also shown in this paper that given an IND-PEKS-CKA2 secure PEKS scheme one is able to derive a chosen ciphertext secure identity based encryption (IBE) scheme, where the keywords in PEKS can be viewed as the identities in the IBE scheme. The authors also hypothesise that one can build an IND-PEKS-CKA2 secure PEKS from any anonymous IBE scheme, this claim is confirmed in [2]. Following on from this Abdalla et al. [3] conclude that the IBE scheme also needs to be robust, meaning that it is hard to produce a ciphertext that is valid for two different users. This property was always implicitly assumed in previous constructions.

Questions about the correctness of PEKS were raised by Adballa et al. [2]. In this work the authors describe three new notions of correctness in terms of the advantage of the adversary in a correctness security game:

- **Perfect**: The advantage of any computationally unbounded adversary is zero.

- **Statistical**: The advantage of any computationally unbounded adversary is negligible.
- **Computational**: The advantage of any polynomial time adversary is negligible.

The correctness requirement mentioned in Boneh et al's work [33] is congruent to the notion of perfect correctness as described above, but as the authors of [2] point out the scheme does not meet this criterion due to the fact that there exist distinct keywords ω and ω' such that search algorithm outputs 1 given $(PK_A, \mathsf{PEKS}(PK_A, \omega'), T_\omega)$, where T_ω is the search query corresponding to ω, meaning, in the example scenario of the email gateway, email may get routed incorrectly. They show that PEKS is in fact computationally correct, meaning that in theory, incorrect routing of email is a possibility, however it is unlikely to happen. So although the notion of correctness in the PEKS scheme is slightly weaker than was initially intended, in practice it is still a useable scheme. This work also recognizes an issue with the PEKS model in that a search query from the data owner can be stored and used to search future ciphertexts which may not be desirable by the data owner. They present a scheme, public key encryption with temporary keyword search (PETKS) , where the trapdoors are only valid for a limited period of time after which the trapdoor will expire and and can no longer be used by the server to search the encrypted data. The security of PETKS is defined analogously to that of PEKS with respect to a time period of the adversary's choice.

The first PEKS scheme to be constructed from something other than bilinear maps is due to Di Crescenzo and Saraswat [58]. The security of the scheme is based on the hardness of computing quadratic residues modulo a large composite integer. Due to the theorem proven in [33] regarding the relationship between IBE and PEKS, the PEKS scheme of [58] gives rise to the first anonymous IBE scheme that is not built using bilinear maps. The scheme is based on the IBE scheme due to Cocks [54]. The PEKS scheme defined in this paper is IND-PEKS-CKA2 secure.

A paper by Baek, Safavi-Naini and Susilo [13] called attention to a major issue inherent in PEKS which is the requirement of a secure channel for transferring the trapdoors from the receiver to the server. Building and maintaining a secure channel is costly which could make PEKS unsuitable in practice. They propose a scheme, secure-channel free PEKS (SCF-PEKS), where the server has its own public and secret key pair. A SCF-PEKS ciphertext is created using both the data owner public key and the server public key, ensuring that, given a search query, only the server is able to carry out the search. The security notion defined in this work is analogous to IND-PEKS-CKA2 with the additional notion that an outside adversary, without the server's secret key, cannot learn anything from a SCF-PEKS ciphertext (except its size) even with all possible search queries. These notions are collectively defined as IND-SCF-CKA2. The security of the scheme defined in [13] holds in the random oracle model. Later work presented in [61] presents a scheme where the security holds in the standard model. Khader [91] presents a SCF-PEKS scheme based on K-resilient IBE, which is also secure in the standard model. SCF-PEKS has more recently been investigated in [60]. Work by Rhee at el. [95] points out weaknesses in the security of the scheme in [13]. They point out that the ability of the adversary in the security game is too weak to be realistic and extend the security to allow the adversary to obtain relations between search queries and ciphertexts (excluding the challenge ciphertexts).

Work by Joong et al. [82] was the first to increase the expressiveness of the queries in PEKS. There are potential solutions for conjunctive keyword search using single keyword equality PEKS schemes, namely:

- **Set Intersection.** The data owner sends a separate search query to the server for each keyword in the desired conjunction then computes the intersection of all the search results. This is not an ideal solution in terms of security and efficiency as it involves the server carrying out extra processing in the way of calculating the intersection of the results and perhaps more importantly the server learns more information than necessary as it will know which documents contain each of the separate keywords, instead of just those that contain all the keywords.

- **Meta Keywords.** Meta keywords are created for every possible conjunctive query. However a document containing n keywords will have associated with it 2^n meta keywords, vastly increasing the amount of storage needed.

Due to the disadvantages of these trivial solutions Joong et al. present two schemes that support *conjunctive* keyword searches over the encrypted data. Both of the schemes are constructed using symmetric prime order pairings and require the encrypted data to have m fixed keyword fields, some of which can be assigned null values, however no two keyword fields for a data item can contain the same keyword. The second scheme is more efficient than the first as it does not employ an admissible encoding scheme in the construction, however both schemes have search queries whose size is dependent on the number of keywords in each conjunctive search query. The security of each scheme holds in the random oracle model and is similar to that of IND-PEKS-CKA2, however insufficiencies in the proofs in this paper are pointed out in [76].

Boneh and Waters [34] extended the expressive of queries in PEKS to include subset queries, range queries, equality queries and general conjunctions of each of these types of queries. The construction in this paper uses *hidden vector encryption* [34]. The authors also present a framework that generalises the notion of a public key encryption scheme that supports various queries over the encrypted data. The security of this scheme is slightly weaker than that of IND-PEKS-CKA2 as it is only *selectively* secure. This means that the challenger in the security game has to commit to their challenge keywords before submitting any queries to the challenger, we denote this form of security as SEL-PEKS-CKA. Bethencourt et al. [26] also present a conjunctive range query scheme using binary interval trees over the integers. In this scheme each encrypted data item can be represented as a point in multi-dimensional space and the range query forms a hyper-rectangle in multi-dimensional space. The range query is satisfied if the point lies within the hyper-rectangle. The scheme is constructed using prime order bilinear maps (either symmetric or asymmetric) and achieves SEL-PEKS-CKA security in the standard model. There is extra leakage in this scheme compared to others described in this section, in that the attribute values are leaked if an encrypted data item satisfies a query.

A generalisation of querying on encrypted data in the public key setting, named predicate encryption, was introduced by Katz, Sahai and Waters [89]. The construction uses inner products over $\mathbb{Z}_N$ (where N is a large integer) to evaluate disjunctions, polynomials, threshold predicates and arbitrary conjunctive/disjunction normal form formulae. The main aim of the work was to achieve attribute hiding in the different forms of queries, which is captured in the selective security definition. Each encrypted data item and search query is associated with a vector. In order for an encrypted data item to satisfy the search query the inner product of these two vectors must result in 0. More specifically, viewing the 'attributes' as keywords, HVE can be used to construct an SE scheme supporting queries containing conjunctions and disjunctions of keywords for example.

Byun et al. [39] highlight a vulnerability present in public key based keyword search schemes. These types of schemes are susceptible to an attack known as an *offline keyword guessing attack*. In this attack an adversary is able to uncover the keyword(s) that were used to generate the search query. An adversary is able to intercept search queries as they are not sent through a secure channel to the server. The adversary is also able to generate arbitrary indexes for keyword(s) of their choosing as these are created using a public key. The adversary can then evaluate the intercepted search queries on the arbitrarily generated indexes and if a search query generates a positive result then the adversary will know which keyword the search query corresponds to as it knows the keywords used to generate the index. Offline keyword guessing attacks are also discussed in [69].

Work by Chen and Tang [48] presented a scheme which is not susceptible to the offline keyword guessing attack called public key encryption with registered keyword search (PERKS). They note that the existing security definition for PEKS, IND-PEKS-CKA2, does not capture the vulnerability of PEKS to the offline keyword guessing attack. Their scheme requires an extra round of communication between the user and the data owner before the user can encrypt data using the data owner's public key. The user has to register each keyword used in the PERKS ciphertext with the data owner prior to encryption, this prevents an adversary from being able to generate arbitrary search queries, hence thwarting the offline keyword guessing attack. There are several issues generated with this scheme that are noted in the paper. These include the requirement of a secure channel between the user and the data owner and the need for the extra round of communication between the user and data owner prior to encryption. Their scheme is constructed using bilinear maps.

Other work by Wang et al. [49] present a scheme that is immune to the offline keyword guessing attack but including an extra server in the PEKS system model. The idea is that the search computation is split across two servers, while the security against offline keyword guessing attacks holds so long as the two servers do not collude.

Katz et al. [88] investigate file injection attacks on searchable encryption schemes. They conclude that these types of attack can completely reveal the search queries in a searchable encryption scheme, but are only realistic in some scenarios, such as email routing. In other scenarios, such as a single user storing and searching over the encrypted data items, a file injection attack is not so realistic. Cash et al. [41] also investigate some other *leakage-abuse* attacks which specifically exploit the information leakage rather than the construction of the SE scheme.

Until the work by Baek et al. [12] the encryption of the data items was considered separately to the PEKS ciphertexts and was encrypted using a standard PKE scheme. The work of [12] looks at considering just one scheme, PKE/PEKS, to accomplish both encryption of the data items and the encryption of the PEKS ciphertexts. They consider a stronger adversary which may modify the encrypted data stored on the server and conclude that there should be a mechanism that binds the PEKS ciphertext to the encrypted data item to prevent the modification of the encrypted data. They call this mechanism *tagging*. The scheme is secure against a chosen ciphertext attack in the random oracle model. The notion of combining PKE and PEKS is also investigated in [77, 50].

Most of the schemes that have been discussed so far leak the access pattern and search patterns and achieve linear search times with respect to the number of encrypted data items, which is deemed to be acceptable for SE schemes. Bellare et al. [20] investigate deterministic PKE schemes and how they can used to achieve more efficient SE schemes, whilst sacrificing some privacy.

Using deterministic encryption has ingrained limitations. Firstly if it is known that the plaintext, x, has been drawn from a small space, X, then, given a ciphertext c_x and the public key it would be possible to work out which $x \in X$ corresponds to c_x. This could be done computing the ciphertext for every $x \in X$ and seeing which one is equivalent to c_x. Hence privacy can only be guaranteed if the plaintext comes from a space that has large minimum entropy. It is noted that database fields are not guaranteed to have a high minimum entropy, so it may be the case that no privacy is achieved at all in some cases. Taking these limitations in to account, a new, weaker, notion of privacy is defined for deterministic encryption which is called PRIV. They define a scheme called encrypt-and-hash (EaH) which achieves fast search times and is shown to be PRIV secure in the random oracle model. A process called bucketization can help to increase privacy levels when database fields are not of large minimum entropy. It works by increasing the number of hash collisions and thus the number of false positives, generating noisy results. The user is then required to sift through the noise in order to obtain the true result. Although this process increases the privacy it also increases the cost of computation for the end user so is not an ideal solution.

The scheme of Hwang et al. [76] which we discussed in relation to conjunctive keyword search, can also be extended to support multiple users. In order to send encrypted data to multiple receivers using the single user PECKS scheme defined in the paper, the user would have to generate multiple encryptions of the same data using the public key of each of the recipients. Multi-receiver PKE schemes are investigated in [19, 17]. The multi-user scheme presented in [76] combines multi-user PKE with PECK. The multi-user PECK (mPECK) ciphertexts in this scheme are linear in size with respect to $n \cdot \ell$, where n is the number of receivers of the encrypted data and ℓ is the number of keywords in the conjunctive query. The scheme is shown to be secure against chosen keyword attacks in the random oracle model.

3.4.3 Searchable Symmetric Encryption

The first searchable encryption scheme, presented by Song et al. [126], was built using symmetric-key primitives. The search in this scheme is carried out using a sequential scan of the encrypted data. It is not an index-based scheme unlike the majority of searchable encryption schemes. Each word in the data item is encrypted separately and then this sequence of ciphertexts is post-encrypted using a stream cipher. The weaknesses of this method include its incompatibility with existing encryption schemes and the fact that it cannot be used on data that has been compressed. The search time is also linear in the size of the data item collection. The security is defined in terms of the standard IND-CPA security, however, there is no consideration for the search queries in the security definition, hence the security in this work is not considered adequate for searchable encryption.

The work of Goh [68] presents the notion of index-based searchable encryption. He presents a method for secure indexing which he claims can be used for many purposes, including searchable encryption. However, as this method is not specifically designed for searchable encryption the security notion does not consider the security of the search queries or the information that may be leaked during a search. This notion of security is referred to as IND-CKA. This scheme is shown to be adaptively IND-CKA secure in the standard model. The index is created using a Bloom filter [30] per data item and keywords associated with the data item are inserted into the Bloom filter. This is an example of a *forward* index as there is an entry in the index per data item, hence the search time is linear in the total number of data items. Bloom filters can also produce false positive results and can leak the number of keywords that they store. Padding the Bloom filter with extra keywords can mitigate the leakage of the number of keywords, however this comes at a cost of increasing the false positive rate.

Chang and Mitzenmacher [46] try to improve on the security definition of [68], by considering the security of the search queries in addition to the encrypted index. However, their notion of security was broken by the authors of [55]. The index in their scheme is constructed using a bit string of length m per data item and a fixed dictionary of keywords of size m. If a data item contains the ith keyword in the dictionary, then the ith bit of the associated bit string is set to 1. The search time is linear in the number of data items. They also specify an extension to their scheme that supports updates (limited to the addition of data items only).

The work of Curtmola et al. [55] has been very influential in the field of searchable encryption. It defines defines two notions of security IND-CKA1 and IND-CKA2 which are selective and adaptive respectively. These definitions are accepted to be standard notions of security for searchable symmetric encryption. A rigorous framework for searchable symmetric encryption is also presented, which classifies the information leakage that occurs during search. The index is constructed using a linked list of search results per keyword, this is an example of an *inverted* index. The search query is used by the server to locate the correct linked list and the search results are sequentially decrypted. Hence, the search time is linear in the number of data items matching the search query which the authors argue is optimal as the minimum amount of work required to retrieve the data items from the server is also linear in the number of data items retrieved.

Another scheme with the same search complexity as [55] is that of [100], which also allows updates to the data. Two schemes are presented, the first requiring two rounds of communication between the server and the user during the search phase (as well as for the index generation and update phases), the second only requires one round of communication for each of these phases but the search phase is more computationally intensive. The search time is logarithmic in the number of distinct keywords in the data items and both schemes are shown to be IND-CKA2 secure in the standard model.

Work by Chase and Kamara defined the notion of structured encryption, which generalises searchable symmetric encryption and supports queries on arbitrarily structured data. It generalised the security notion presented by Curtmola et al. [55] for arbitrarily structured data.

The strongest security model used in the SSE literature is that of *universal composability*, meaning the scheme remains secure when it is arbitrarily composed with other protocols. The scheme presented by Kurosawa and Ohtaki [93], which is based on that of Curtmola et al. [55], achieves UC-security.

The first SSE scheme to consider a dynamic data, where data items can be added, deleted and updated, was that of Kamara et al. [86]. Their scheme is based on the linked list construction of [55], which they extend to support dynamic data by augmenting the index to include two additional data structures. The leakage is parameterized using leakage functions to explicitly show the leakage that occurs during setup (before any search queries), search and updates. Security is proven in the random oracle model and under an extended notion of IND-CKA2 security, taking into account the dynamicity of the data.

Due to the sequential nature of the search in [86] it is not possible to parallellize the search to improve the search time. Further work by Kamara et al. [85] uses a tree-based index which is parallellizable for a more efficient search. The index consists of a keyword red-black (KRB) tree per keyword. A KRB tree is a dynamic data structure that stores the data item identifiers at the leaf nodes and at each internal node stores an m-bit vector. The ith bit of this vector is set to 1 if there is a path from that node to a leaf node for data item d_j such that d_j contains the keyword ω_i. The search time complexity is $\mathcal{O}(\mathcal{D}(\omega)\log n)$ where $\mathcal{D}(\omega)$ is the number of data items containing keyword ω and n is the total number of data items. This can be reduced to $\mathcal{O}(\frac{\mathcal{D}(\omega)}{p}\log n)$ where p is the number of processors computing the search. The scheme is shown to be IND-CKA2 secure in the random oracle model.

All the schemes discussed so far only support keyword equality queries. Golle et al. present the first conjunctive keyword search scheme in the symmetric-key setting [71]. They propose the method of assigning keyword fields to each data item in the index. If we consider the data to be emails, for example, the keyword fields could be made up of the sender's email address, the date and the subject. The scheme requires that each field needs to be assigned a value, however these values can be null values if required. A disadvantage of this method is that the user needs to be aware of the keyword fields and ensure that the keywords are in the correct position in the conjunctive search query. They use the same security as that of [68] (IND-CKA) in the random oracle model which they extend to encompass conjunctive queries. Two schemes are presented in this paper, one which is more space efficient on the server but requires offline computation to be sent to the server ahead of querying. In this scheme the search query is of constant size. The second scheme requires twice the amount of storage on the server and the size of the search queries is linear in the number keyword fields, however requires no offline computations prior to querying.

A conjunctive scheme that is proven IND1-CKA secure in the standard model is that of [14], which is constructed using Shamir's Secret Sharing. Byun et al. [38] also present a conjunctive SSE scheme which is made more efficient for large datasets with the use of bilinear maps. This scheme is proven IND1-CKA secure in the random oracle model.

The schemes of [14, 38, 71] all have search queries of size linear in the number of terms in the conjunctive query. Work by Ryu and Takagi [124] reduces the size of the search query to around the size of single keyword query. The scheme is shown to be secure in the sense of IND1-CKA in the random oracle model, similary to the other conjunctive schemes discussed.

All conjunctive SSE schemes discussed so far have required keyword fields in the construction of the index. Pieprzyk et al [116] present a conjunctive scheme that does not require keyword fields. The search queries also conceal the number of keywords in the conjunctive query. The keyword fields are removed by using a different bilinear map on each keyword for each index entry.

The work of [37] give a formal definition of security for the conjunctive keyword search schemes based on a relational database. They also discuss some issues associated with these schemes, such as the reusability of search queries.

Shen et al. [125] present a symmetric-key predicate encryption scheme which supports inner product queries and also achieves search query privacy. The work of Katz, Sahai and Waters [89] in public-key predicate encryption showed that inner products can be used to support a wide range of expressive queries, such as conjunctions, disjunctions, polynomial evaluate and conjunctive normal form and disjunctive normal form formulae. Shen et al. define the notion of *full security* in terms of searchable encryption, meaning that no information is leaked by the encrypted data items, the search query or the search phase except the access pattern. The size of the search query is dependant on the number of data items, meaning that this scheme would not be suitable for large data sets.

Recent work of Cash et al. [43] describes an SSE scheme that supports conjunctive and general Boolean queries. The emphasis on this work is that of practicality. They present a scheme that adapts the schemes of [55] and [47] and is the first example of a sub-linear conjunctive search. They define bounded leakage functions, which leak more information than just the intersection of all the searched keywords, however, this is sacrificed for greater efficiency. Futher work by Cash et al. [42] presents a generalisation of the previous work [43] to support dynamic data and maintain search efficiency in very large databases.

Range queries in the symmetric-key setting can be achieved using OPE (definition 2.3.12). OPE is a form of deterministic encryption that preserves the ordering of the plaintexts in the corresponding ciphertexts, hence can achieve fast search times. OPE was introduced by Agrawal et al. [4], yet Boldyreva et al. [31] were the first to introduce a concrete implementation of an OPE scheme that satisfied their security notion, POPF-CCA (pseudorandom order-preserving function against chosen-ciphertext attack). This security notion requires an adversary cannot distinguish whether they are have oracle access to an instance of the encryption scheme or a random order-preserving function. In this paper the authors highlight their concerns about the security of using this scheme in practice as there is no analysis regarding how the information leakage from the index can be used by an adversary. This work is revisited in [32] where the authors goal is to outline the security and functionality trade-offs associated with OPE in order to help practitioners make informed decisions on implementing OPE. In this work the authors also study some schemes that are not built using OPE, but yet support efficient range queries over encrypted data along with an extension to OPE, namely Modular OPE, that improves the security of OPE. More recent work by Kerschbaum [90] enhances the security of OPE by randomizing the ciphertexts to conceal the frequency of the plaintexts. Work by Naveed et al. [105] demonstrate some attacks on low entropy ciphertexts encrypted using OPE, where the plaintext can be recovered using only the ciphertext and publicly available information.

The work of Ishai et al. [78] presents an SSE scheme using a two-server system model, which is referred to as *distributed* SSE. It leaks no access pattern information up to a certain threshold number of queries.

3.5 Summary

In this chapter we defined SE in the public-key and symmetric-key settings, as well as presenting a general model for SE. In addition we defined multi-user SSE and verifiable

SE which feature prominently in the work in the following chapters. We outlined the security and adversarial models associated with SE and give a literature review of related work in the field of SE.

Chapter 4

Searchable Encryption in the Real World

Contents

This chapter was based on research conducted at Thales Research and Technology (UK), which appears in [120].

4.1 Introduction

The literature regarding SE is extensive (see Section 3.4), however SE is not widely deployed in practice. This chapter identifies and analyses different scenarios to which SE can be applied in the real world and investigates the suitability of certain types of

SE schemes for each scenario. We also explore the reasons as to why SE schemes are not widely implemented and look at the security issues and functionality of protocols that are currently being implemented that achieve some form of search over encrypted data.

The inspiration and research for this chapter came from my internship at Thales UK Research and Technology. One of the aims of the research was to introduce SE as a new technology and demonstrate its uses. In this role at Thales we analyzed practical scenarios that arose from combat situations, working with sensitive data and controlling access to data with varying degrees of classification and looked at ways SE could be used to solve problems within these scenarios. When analyzing the scenarios we looked at features such as the number of users, the adversarial threat, sensitivity of the data involved and whether static or dynamic data is used and assessed the suitability of particular SE schemes to each scenario. We used this research to define four basic scenarios to which SE could be applied based on the number of users and the capabilities of each user. Within each of these basic scenarios we detailed a generic method of SE that is well suited for use in that scenario and identified varying features of these scenarios that occur in the real world. We then mapped specific SE schemes into the different instances of the scenario depending on the varying features.

There is a comprehensive survey of provably secure searchable encryption schemes that post-dates this research that can be found in [35]. The authors of [35] follow a similar categorization of SE schemes, however their survey takes a theoretical approach and does not consider the practical reasons behind the categorisation of the SE schemes. We do not intend to provide a comprehensive survey of every SE scheme to date. Rather, the intention here is to explore what features of a SE scheme might make it more suitable for use in a particular scenario in order to facilitate the design of protocols that use SE to provide solutions to real-world problems.

4.2 Architecture of Searchable Encryption

In this section four basic scenarios for searching over encrypted data in the cloud are described, along with general methods of achieving SE that are potentially suitable for these environments. This section analyses SE schemes that achieve a single keyword equality search over encrypted data. The four basic scenarios are as follows:

1. **Scenario 1**: Only the data owner reads and writes all the data.

2. **Scenario 2**: Only the data owner can read the data, all users can write data.

3. **Scenario 3**: Only the data owner can write the data, many users can read the data.
4. **Scenario 4**: Many users (including the data owner) can both read and write data.

In Section 4.3 we describe each of these scenarios in detail, give a practical example of each scenario and present a generic method of SE that can be applied to each scenario. In the real world each scenario will present a number of variables that can affect which SE scheme would be best to use in each case. We consider the following variables:

- Size and type of data (static or dynamic),
- Frequency of queries,
- Number of additional users (if any),
- Sensitivity of data,
- Type of search query (equality, conjunctive etc.),
- Available local storage.

For each of these variables we identify the most suitable SE scheme according to its construction and features.

4.3 Application of searchable encryption to provide solutions in the Scenarios

4.3.1 Scenario 1: Only the data owner reads and writes all the data

The simplest scenario is where only the data owner is involved in reading and writing data and there are no additional users in the system. The data owner sets up the system and generates the secret key. They create an encrypted index for the data items using the secret key and outsource it to the remote server along with the set of encrypted data items. The data owner generates search queries for the server using their secret key which the server then uses to compute the search results satisfying the search query. The search results are then transferred to the user.

In practice the data owner in this scenario could be the owner of a small business who wishes to outsource their confidential client data to a public cloud server due to an inadequate local storage capacity.

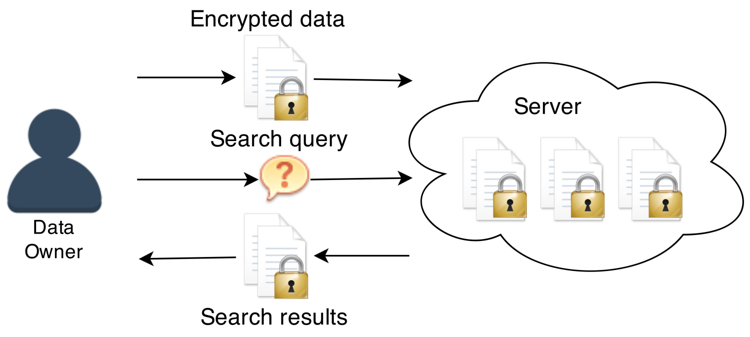

Figure 4.1: Only the data owner reads and writes all the data.

As there is only one entity reading and writing the data in this scenario PKE does not offer advantages over more efficient SKE schemes as no secret key distribution is required. A generic method for performing SE in this scenario is SSE (Definition 3.1.1). There are many different SSE schemes defined in the literature which support static data [68, 46, 47, 55, 93, 100, 126] and dynamic data [86, 42, 43, 85, 106, 127, 114], to provide a few notable examples.

- **Size and type of data.** If the dataset is very large it will be beneficial to choose an SE scheme whose search time does not increase linearly with the size of the data set. SSE schemes that achieve a sub-linear search time using a static dataset are [47, 55]. The schemes of [47, 86] only require $\mathcal{D}(\omega)$ symmetric decryptions per search, where $\mathcal{D}(\omega)$ is the number of data items satisfying the search query. Examples of schemes that support a dynamic data set and sub-linear (with respect to the number of data items) search are those of [43, 86, 85, 100]. The scheme of [85] is also parallellizable so it can achieve a search complexity of $\mathcal{O}(\frac{\mathcal{D}(\omega)}{p} \log n)$ where n is the number of encrypted data items, and p is the number of processors. The work of [42] has been designed specifically to support search over 'very large' databases.

- **Frequency of queries.** If a vast number of search queries will need to be evaluated on the encrypted data, then choosing a scheme with efficient Query and Search algorithms will be beneficial. Fast search times can be produced using deterministic and order-preserving encryption [20, 31]. Searchable encryption schemes using deterministic encryption can produce very fast (logarithmic) search times, however in order to achieve this some security is sacrificed. The index consists of deterministically encrypted keywords, hence repetitions in these keywords will be revealed to the server. Deterministic encryption in the symmetric-key setting is more secure against keyword guessing attacks than in the public-key setting. When the ciphertexts are created using a public key an adversary is able to create ciphertexts corresponding to arbitrary plaintexts of their choosing. Hence, if an adversary obtains a ciphertext they can compare it to any of their arbitrarily created ciphertexts in order to determine the associated plaintext. In the symmetric-key setting an adversary is unable to mount an attack like this as they do not have the secret key to generate any ciphertexts. For more expressive queries, such as range queries, order-preserving encryption can be used to achieve a fast search time. The security sacrifices for the faster search time using order preserving encryption and similar to those for deterministic encryption. If the data owner just wishes to minimize their computation requirements, it will be beneficial to have a Query algorithm which is not so computationally intensive. The schemes of [68, 85, 100] only require the evaluation of one PRF per search query, which makes their Query algorithms very efficient.

- **Number of additional users.** Not applicable to this scenario.

- **Sensitivity of data.** As outlined in Section 3.4, ORAM can be used to perform searches over encrypted data without leaking the access pattern, a downfall of most SE schemes. However these types of schemes can be very computationally expensive so will not be suitable for very large datasets. Several schemes have been presented that improve the efficiency of the ORAM technique [29, 45, 56, 57, 66]. TWORAM, presented in [66], is the most promising ORAM-type searchable encryption scheme that can conceal the search pattern, however, it requires that the data items are stored in oblivious access memory in order to conceal the access pattern. This scheme achieves a sub-linear search time.

- **Type of search query.** The searchable encryption scheme suitable for this scenario with the most expressive queries is that of [125], which supports the following types of queries: conjunctive, disjunctive, polynomial and conjunctive/disjunctive normal formulae. This scheme uses bilinear maps in the construction of the index and the search queries, which are fairly computationally intensive for the user. Recent SSE schemes [42, 84] can support arbitrary disjunctive and Boolean queries, however the scheme of [84] is slightly more efficient in terms of search time as in the worst case its searches are sublinear, whereas for arbitrary disjunctive and Boolean queries using [42] these types of searches are performed in linear time.

- **Available local storage.** In this setting, where only the data owner is reading

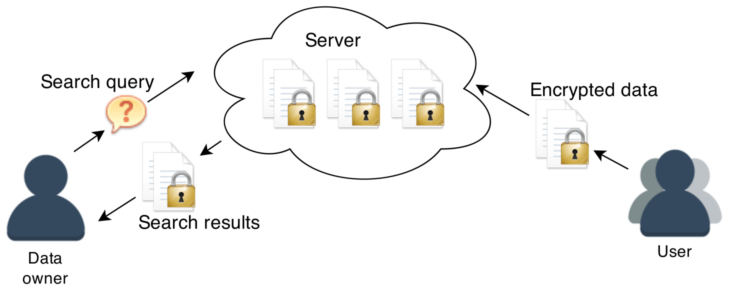

Figure 4.2: Data owner can read data, all users can write data.

and writing data to the remote server, if the data owner has local storage available then an index for the encrypted data could be stored locally, un-encrypted. This would achieve the fastest possible search, as the data owner is searching on un-encrypted data.

4.3.2 Scenario 2: Data owner can read data, all users can write data

In this scenario both the data owner and a number of arbitrary other users are able to write data (Figure 4.2). The data owner, however, is the only party that can read the data.

This scenario is well illustrated by an email routing system. Users send encrypted emails to the data owner. To allow relevant routing of these emails the server receives all the incoming emails and routes them according to the data owner's preferences. In order to ascertain how to route the encrypted emails they will be securely indexed with keywords (these could be all words in the subject of the email, for example). The data owner supplies the server with search queries to enable it to detect which emails contain a specific keyword and then route them according to the data owner's instructions without revealing the plaintext email to the server.

As there is more than one party that can write data, SE schemes for this scenario need to be able to support dynamic data as encrypted data will most likely be added to the server at different times by different parties. In terms of encryption, PKE schemes are well suited here. Using PKE allows simple key distribution to the users as encryption can be done using the data owner's public key. The secret key is used by the data owner to generate search queries. This also ensures that only the data owner is able to produce search queries and decrypt the data items.

- **Size and type of data.** As encrypted data items in this scenario are uploaded by different users, the index is necessarily a forward index, having an entry per data item. Subsequently, the search times are linear in the number of encrypted data items, which might not be suitable for practical systems. In addition most of the public-key schemes suitable for this scenario use pairings and computationally intensive group operations in the Search and BuildIndex algorithms, which for large datasets may not be feasible for a practical system. The scheme of [91] does not use pairings in the BuildIndex or Search, which may make it a more practical choice.

- **Frequency of queries.** Due to the computationally intensive primitives used in the algorithms in public-key searchable encryption schemes they might not be suitable for use in scenarios where a high frequency of queries are issued to the server. The schemes of [13, 58, 91] do not use group operations or pairings in their Query algorithms, they use hash functions ([13, 58]) and polynomial evaluation ([91]).

- **Number of additional users.** As the data is encrypted using a public key, no secure channel is needed for key distribution, hence a large number of additional users does not impact the efficiency of the scheme in this sense. However, more users may imply more encrypted data being stored on the server so it could indirectly affect the efficiency of the scheme due to the issues arising from searching over very large datasets (see previous point regarding size and type of data).

- **Sensitivity of data.** Public-key searchable encryption schemes may be vulnerable to the offline keyword guessing attack described in Section 3.4.2. The schemes of [48, 49] are not susceptible to this attack, however [48] involves a user registering a keyword prior to encrypting data, so there is an extra round of communication between the user and the data owner prior to encryption. The scheme of [49] requires an extra server in order to prevent offline keyword guessing attacks, with the assumption that the two servers do not collude.

- **Type of search query.** The most expressive searchable encryption schemes for this scenario are that of [34, 89]. The scheme presented in [34] supports conjunctive, subset and range queries, whereas [89] supports the following types of queries: conjunctive, disjunctive, polynomial and conjunctive/disjunctive normal formulae.

- **Available local storage.** As the encrypted data is sent straight to the server by the users, in order to index the data locally the data owner will either need to download the encrypted data in order to index it or allow the additional users to

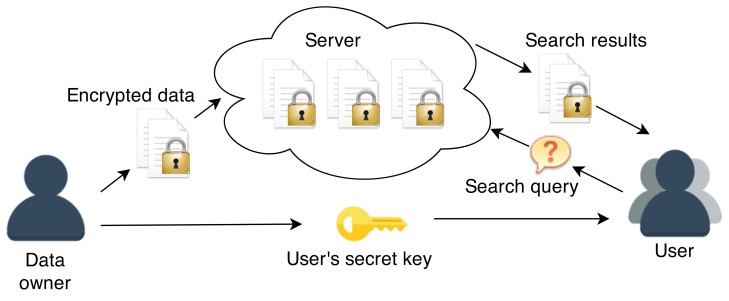

Figure 4.3: Data owner can write data, many users can read data.

upload the index information. This will need to be repeated every time encrypted data is added by a user. Due to the large communication costs associated with this method, we do not consider this a feasible option.

4.3.3 Scenario 3: Data owner can write data, many users can read data

In this scenario, all the encrypted data stored on the server is written by the data owner, who wishes to provide read capabilities to other users, whilst not allowing them the ability to write data (Figure 4.3). As only the data owner is writing data, SKE is a good choice for constructing SE schemes for this scenario and the data could be either static or dynamic.

In practice a corporation may want to outsource some encrypted data to a remote cloud server. In order to allow employees to read the data a SE scheme that supports multiple readers is required. This group of users could also be dynamic meaning that users may get their read capabilities revoked, when an employee is fired for example, or new users maybe enrolled into the system when the corporation hires new employees.

Although this scenario seems to reflect many real-world instances of searching on encrypted data, the literature is not very extensive. In Section 6.3 we investigate generic solutions to multi-user search on encrypted data some of which are applicable here.

- **Size and type of data.** Using broadcast encryption as a method for read-only user addition and revocation was put forward in [55]. Using this method a single user SSE scheme can be converted to a multi-user one, inheriting the properties of the underlying single-user scheme. Choosing a suitable single-user SSE scheme (see Section 4.3.1) and applying a broadcast encryption scheme to it would provide a good solution here, allowing only authorised parties to generate valid search queries.

- **Frequency of queries.** The solution described in the point above also applies here.
- **Number of additional users.** If there is a dynamic user group searching the encrypted data then the user addition and revocation mechanisms will need to be efficient. The scheme of [135] stores a unique key per user on the server which is used together with a search query to search the encrypted data. To revoke a user the user's key is deleted from the server which prevents the server from being able to execute the user's search queries. The revocation in this scheme does not affect the other authorised users.
- **Sensitivity of data.** There are no ORAM solutions specifically for this scenario. One could use a broadcast encryption scheme along with single user ORAM in order to allow multiple users to query the encrypted data.
- **Type of search query.** Using broadcast encryption extends a single-user SSE scheme to multi-user one, hence the single-user schemes detailed in Section 4.3.1 are also applicable here when combined with a broadcast encryption scheme to handle user revocation and addition.
- **Available local storage.** If users have available local storage then each one could store an index locally, if trusted by the data owner. This would work most efficiently when using a static dataset, as any updates to the data would require the data owner to update all the user indexes. If the additional users do not have equal access rights then it would be possible using this method to have a different index for each user, so the user can only learn information regarding data items they are authorized to search.

.

4.3.4 Scenario 4: Many users can read and write data

In this scenario many users including the data owner have read and write capabilities (Figure 4.4).

An example of this scenario in the real world is the management of encrypted electronic health records (EHRs). If you take just one hospital, there are numerous doctors, nurses and other staff (which represent the users and data owners) that write information to

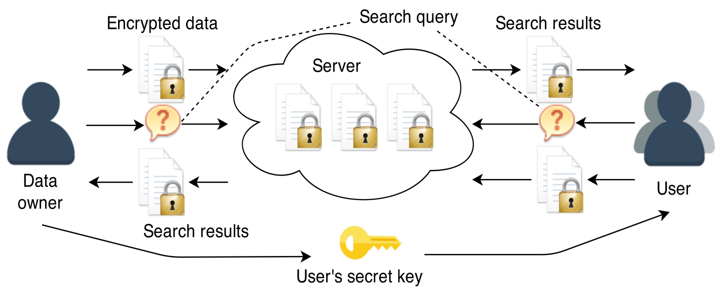

Figure 4.4: Many users can read and write data.

the EHRs. Due to the sensitive nature of the data, access will be restricted to certain users only. For example, a general practitioner may be authorised to read the records of their patients only, whereas a heart specialist may be authorised to read all records relating to heart conditions. All medical staff may need to write data to the records.

As there is more than one entity writing data, as in Section 4.3.2 we need to support dynamic data. Both SKE and PKE are possible in this scenario but produce a subtle difference in the SE schemes' functionality. In a scheme that uses SKE a revoked user will no longer be able to read or write data, whereas when using PKE a revoked user will no longer have read capabilities, however they will still be able to write data to the server as this is done using the public key. We shall look at PKE and SKE based methods for building SE in this scenario.

- **Size and type of data.** For large datasets deterministic encryption [20] could be used along with a proxy server that uses proxy re-encryption to transform user's queries to the correct form. Each user could encrypt data using their own public key. In order to search other user's encrypted data the proxy server can convert search queries to the correct form depending on which user encrypted the data. This could involve the proxy server computing many different search queries in order for a user to search many different users' data, so this method might not scale well to a large number of users.

- **Frequency of queries.** A scheme by Bao et al. [15] allows multiple readers as well as writers using SKE. It employs a trusted authority to generate keys and enrol/revoke users in the system. The method used to add and revoke users is similar to that using BE, however generating a search query and writing a data item to the server requires a valid value which varies from user to user. The server stores information regarding these values and authorized users in a dynamic list.

Using SKE makes the Query and Search algorithms more efficient, hence this method might be suitable for a system with a high volume of queries.

- **Number of additional users.** Multi-receiver PEKS [13] extends the PEKS scheme to a setting that allows multiple readers as well as writers. Each user in the scheme has their own public key. For a large number of additional users this scheme is suitable as user keys can be transferred to the users publicly.

- **Sensitivity of data.** If the data is very sensitive, then ORAM techniques may be applicable. A multi-user ORAM scheme is presented in [81], which uses a chain of proxy servers to protect the data access pattern of each user. Another multi-user ORAM scheme is presented in [29], however, this involves an ORAM per user and only supports a user reading data from their own ORAM, whereas they can write data to other users' ORAMs.

- **Type of search query.** Wang et al. [116] present a conjunctive scheme suitable for this scenario. No other schemes supporting more expressive search queries are detailed in the literature.

- **Available local storage.** As the encrypted data is sent straight to the server by the users, in order to index the data locally the data owner will need to download the encrypted data. This will need to be repeated every time a user submits encrypted data, or otherwise the user could also submit indexing material along with the encrypted data. Due to the large communication costs associated with this method, we do not consider this a feasible option.

4.4 Searchable Encryption in the Real World

In this section we give a brief analysis of some products that claim to support searching on outsourced encrypted data. We try to detail the type of encryption used and the setup and search leakages, although as some of these products are proprietary full details of the encryption method may not be available.

- **CipherLocker**: A product named CipherLocker which has been created by a company called Private Machines Inc. claims to be 'the first platform that lets you securely search through files that are stored encrypted on the server, without having to download the files' [1]. The model used is different to that of standard SE (Section 3.1) as the index is not kept on the server. The data items themselves are encrypted using a semantically secure authenticated encryption scheme before they are transferred to the server for storage. The searches are performed locally by the user hence there is no related leakage from the searches to the server, except for the list of data items retrieved. CipherLocker provides a secure way to locate and retrieve encrypted data items from a server, but the model is very different to that of SE that is discussed in this thesis. We can say that if every SE scheme stored an index and performed the search locally then all schemes would achieve this high level security. In the literature a standard assumption is made that a data owner wishes to outsource sensitive data to a remote server due to local storage restraints, hence CipherLocker may not be a suitable solution to searching on encrypted data for everyone. Internal adversaries may also need to be taken into consideration when using this method.

- **CryptDB**: CryptDB [118] was developed by Popa et al. and is one of the most well-known and documented solutions to searching on encrypted data. The authors are very transparent about the types of encryption used in CryptDB along with the associated leakages, unlike other products discussed in this section. This scheme works on data in SQL databases and allows the evaluation of SQL queries over the encrypted data. Using CryptDB it is possible to encrypt different parts of the database using different types of encryption, depending on what types of queries will be evaluated on that part of the database. This scheme is a good example of the relationship between functionality and security that is often considered when designing a SE scheme.
  - **Random (RND)**: This is the strongest form of encryption used in CryptDB and produces ciphertexts that are IND-CPA secure. It does not allow any computation on parts of the database encrypted with this form of encryption.
  - **Deterministic (DET)**: Deterministic encryption [20] leaks which values are equal in the encrypted data, i.e. equal plaintexts produce equal ciphertexts. Using this type of encryption one can perform equality checks on the encrypted data which enables the user to locate encrypted data that matches a target value. Work by Naveed et al. [105] has shown that the information leakage from ciphertexts encrypted using this form of encryption could be used to uncover the underlying plaintext.
  - **Order preserving encryption (OPE)**: OPE [32] leaks the order relations between ciphertexts. If we have that a plaintext x is less that a plaintext y then if using OPE the resulting ciphertexts c_x, c_y will retain this order relation i.e. c_x is less than c_y. Using OPE to encrypt the data one can perform range queries and determine the maximum and minimum values within the encrypted data. Work by Cash et al. [59] has shown that when

there are multiple correlated columns in a database encrypted using OPE then these columns can be used together to reveal more information about the underlying plaintexts than previous attacks could reveal on each column individually.

  - **Homomorphic encryption (HOM)**: Homomorphic encryption [113] produces ciphertexts that are IND-CPA secure and allow add queries to be performed over the encrypted data. The leakage is limited to size of the ciphertexts. The security for this form of encryption is similar to that of RND.
  - **Word Search (SEARCH)**: The scheme that is used to perform text search is that of [126]. The text is split into individual keywords and the repetitions are removed and the words permuted. Each keyword is encrypted using the scheme as defined in [126]. The ciphertexts are then padded to ensure they are all the same size. The leakage is limited to which data matches the search query. If there exists RND encryptions of the same data then an adversary may be able to detect the number of distinct keywords in the data by comparing the size of the SEARCH ciphertexts with the RND ciphertext.

  CryptDB performs much more than keyword search as we have detailed. It also provides flexibility for the user to assess what kind of security they need on their data and whether they would like to sacrifice leaking some information to the server in exchange for being able to evaluate more expressive queries. It requires a proxy between the user and the server in order to translate the SQL queries to be compatible with the encrypted data, as well as decrypt the query results before transferring them to the user. CryptDB has been used by several companies including Google, SAP, Lincoln Laboratory, Skyhigh and Microsoft [117].

- **CipherCloud**: CipherCloud market a type of encryption called *Searchable Strong Encryption*, which is not the same as the notion of 'SSE' discussed in this thesis. It involves a local index being stored by the user in order to support searching, similarly to Cipherlocker. They also use tokenization to support search on encrypted data. They claim that in some countries which have strict data residency laws, such as Singapore and Luxembourg, companies are not allowed to encrypt their data, hence tokenization is used instead of encryption for clients in these countries to comply with regulations whilst still somewhat protecting their data.

Although any form of encryption on the data is more secure than nothing at all it is obvious that the claims of some companies that supply products that support searchable encryption are slightly misleading for prospective clients.

4.5 Deployment challenges

Despite the existence of seemly practical searchable encryption schemes, there is still scarce evidence of their deployment. Some companies claim that they can achieve searchable encryption but with further investigation into their systems they are usually built using a technology called *tokenization*. Tokenization works by replacing a plaintext string (a keyword for example) by a completely random string (*token*) before uploading it to the server and storing a map of these replacements locally. Although the tokens are chosen randomly, identical plaintext strings are mapped to the same token meaning equality between plaintexts will be leaked. This will provide a fast search (same as searching a plaintext) but requires a lot of data (the mapping) to be stored locally meaning if the entirety of the data is tokenized then there will be just as much data locally stored (excluding repeated keywords) as there is on the server. This method of data confidentiality seems to be more suited to scenarios where only parts of the outsourced data needs to be protected. For example in a database of customer details, only the very sensitive portions of the data such as social security numbers might need protecting. The level of security provided by tokenization is roughly equivalent to that of DE yet it also has the requirement of local storage space, as the user needs to store a dictionary for all the tokenized values. SSE although only achieving a fairly coarse grained search is still able to achieve sub-linear search times and a high level of security so why are companies and consumers not adopting these methods to achieve data confidentiality in the cloud environment? We discuss several reasons that we believe form barriers to the adoption of the cryptographic techniques for searching on encrypted data detailed in this work.

Usability. Searching over data in the clear is an everyday task for most people. Users are accustomed to the efficient and highly functional search facilities found with search engines such as Google. As a result they expect the same levels of usability from an encrypted search. Unfortunately the SE schemes in the literature do not offer this combination of functionality with efficiency; they cannot compete with the search tools that users are accustomed to in this respect. This ties in with the point regarding user education, if users understood the security benefits of using SE compared with other methods, such as tokenization, then it might make SE a more appealing choice.

Lack of prototypes. SE is a relatively new technology, and there exists few prototypes or demonstrations to illustrate the successful adoption of SE in the real world. This makes the implementation of SE into new products or systems over established methods highly risky for the user. Alongside this is the issue of the lack of standardised algorithms and protocols relating to SE which make it harder to implement without specialised knowledge in the area of SE. There are a wide variety of SE schemes available which are well suited to different scenarios yet without any evidence demonstrating the suitability of a scheme to a particular scenario there is no motivation for the user to choose this new technology over established methods. This ties into the next point regarding user education.

User education. Users may not understand how SE can be used to solve their problem or the security benefits it provides over existing methods. Furthermore users may not understand the security issues associated with using an alternative method such as tokenization. Educating users on these issues will help them make a more informed choice and prevent the implementation of a method that is not fit for their purpose.

Legal issues. The Investigatory Powers Bill (IPB) [109] gives new powers to law enforcement allowing them to issue subpoenas to cloud providers storing encrypted data requiring them to break the encryption and release the plaintext data. In the case of SE where the encrypted data stored on a company's server is encrypted and outsourced by a third party and not by the cloud provider itself they would be unable to provide the plaintext data in this case. This could prevent the deployment of SE as the cloud providers would be unable to store third party encrypted data as if requested by law enforcement they would be unable to provide the plaintext data. There are many questions that arise from the passing of the IPB and what power it gives the government to decrypt private data as the wording in some aspects of it is ambiguous. Hence, it is hard to predict the effects it will have on peoples' willingness to encrypt their data. However, we acknowledge that it may affect how people choose to encrypt and store their data.

Compliance. It is obvious that SE would be of great benefit to users or institutions that handle a lot of sensitive data, such as banks. However new technologies such as SE are effectively prevented from being deployed, as they do not comply with these institution's regulations which usually require the use of legacy systems. Institutions also have policies in place regarding how data is to be handled and how their systems are used. Failure to comply with these regulations by an employee would result in disciplinary or legal action against the employee. It is believed that this provides enough of a deterrent to a potential inside adversary and provides a means for keeping sensitive data confidential without the use of encryption. However data breaches are becoming more frequent and severe [119] which motivates the need for provably secure methods of data protection as opposed to ones that rely on trust and user compliance. There are laws in place dictating that access to particular types of sensitive data, such as medical records, should be restricted. The laws do not dictate how to enforce such a restriction however. These laws should be updated to specify how the data should be protected, to eliminate the reliance on compliance for data security.

Lack of communication between academia and industry. In researching for this work we encountered first hand experience of this issue. There is a distinct lack of dialogue between the academics that research in the area of SE and the practitioners that would like to implement the technology. Without a conversation between these two parties, academic researchers do not know which features of SE are most desirable for a practitioner and hence may not focus on these features in their research. In order to facilitate the development of SE schemes that meet the needs of practitioners there needs to be a more open dialogue between academia and industry. Some companies (Skyhigh, for example) have started to address this issue by appointing a cryptographic advisory board consisting of five leading academic researchers. This gives the company access to information regarding cutting edge technologies and specialised advice and on the other hand gives the academic researchers problems and areas of research to focus their work on.

4.6 Summary

In this chapter we defined four scenarios in the real world in which searchable encryption can be applied. We discussed various features of SE schemes and map suitable SE schemes into the four scenarios. We hope that this work will help practitioners design products that implement SE over less secure methods such as tokenization, by highlighting suitable SE solutions and discussing the tradeoffs between security and efficiency within these scenarios.

We discussed some SE products that are available to users along with their pros and cons.

Lastly we discussed some issues we feel are preventing the deployment of SE schemes in the real world.

Chapter 5

Extended Functionality in Verifiable Searchable Encryption

Contents

In this chapter we employ the techniques of VDC (Section 2.4.3) to extend the functionality of VSE (Definition 3.1.7). The scheme we present allows the user to verify that search results are both correct and complete in addition to allowing more complex verifiable computations to be performed on the encrypted data such as averaging and counting functions as well as the standard keyword matching queries. The results of this chapter are published in [6].

5.1 Introduction

The majority of existing work in SE focuses on achieving secure search over encrypted data in the presence of an honest-but-curious adversary. The paradigm of Verifiable Searchable Encryption (VSE) assumes a stronger semi honest-but-curious adversarial model (Definition 3.3.2) where the server might execute only a fraction of the search or return only a fraction of the search results to the user in order to conserve its own computational resources or download bandwidth respectively. A VSE scheme allows the user, who created the search query to verify that the search results it receives in response to their search query from the server are both correct and complete. Our scheme, extended VSE ($e\mathcal{VSE}$), supports multiple users reading data with a single user, the data owner, writing data (Scenario 3, Section 4.3.3). Verifiability of search results is of particular interest in this scenario as users (other than the data owner) would most likely not have seen the data prior to it being encrypted and outsourced to the server as they do not own the data. We use a CPABE scheme (Definition 2.3.5) as the main building block for $e\mathcal{VSE}$: we encode the index for the encrypted data into a CPABE secret key which is stored on the server along with the encrypted data and the search queries are composed of two CPABE ciphertexts. Each user that is authorized to query the data has their own secret key which is used in conjunction with the CPABE public key to create the search queries.

Related work on VSE (as described in Section 5.1.1) does not allow very expressive queries over the encrypted data and only permits the user that initiated the search query to verify the correctness of the search results. Our scheme $e\mathcal{VSE}$ extends the range of queries and computations that a VSE scheme can verifiably perform as well as supporting the public verification of the search results. To summarise, our contributions in this chapter are:

- **More expressive queries**: Our scheme supports search queries such as boolean formulae involving conjunctions, disjunctions and negations, threshold operations, polyomials, arbitrary CNF and DNF formulae (depending on the underlying CPABE scheme used - see discussion in Section 5.3.1).

- **Evaluation of computations**: Our scheme supports some simple computations such as averaging and counting on the encrypted data. This is achieved by indexing the data items with keywords that represent binary data values as well as the standard alphabetic keywords, see Section 5.3.4.
- **Public verifiability of search results**: Any user in the system is able to verify the correctness and completeness of the search results.

5.1.1 Related work

Verifiable searchable encryption

The first VSE scheme was presented by Chai and Gong [44]. It supports verification of search results from a single keyword equality query. This scheme uses a *trie* to index the data items. A trie is a structure that efficiently stores words by reading each one in from root to leaf, hence a word ω of length $|\omega|$ can be searched for in the trie in a maximum of $|\omega|$ steps. The method of verification in this scheme involves producing a proof value for every step in the search process as the server traverses the trie. This concatenation of proof values is sent to the data owner along with the search results to allow the data owner to verify that the server completed the search correctly. The proof consists of a chain of hash values and a set of search results concatenated with a hash of the search results. To check the completeness of the search results, the data owner computes the hash of the received search results and compares it to the received hash value. If these values are equal then the data owner can be sure the set of search results is complete. To check that the correct search was carried out the data owner decrypts the values in the concatenation and compares them with values in the search query. Each proof value contains information regarding the previous node in the trie, hence the path through the trie can be verified by the user. The size of the proof received from the server leaks how many letters matched in the searched keyword. For example, if a server returns a negative search result and the proof has size j, then this tells the data owner that the first $j - 1$ letters matched, but there was a mismatch on the jth letter.

Another verifiable search scheme that supports keyword equality search only is that of [93]. They use a similar technique to [44] for the verification as they include a MAC of the search results (instead of a hash of the search results). The data owner is able to verify if the MAC value included in the search results is correct by computing the MAC of the search results themselves. This scheme is shown to be UC-secure, meaning the scheme can be arbitrarily composed with instances of other schemes, or instances of the same scheme.

A scheme by Kissel et al. [92] extends the construction in [44] to support multiple groups of users searching over the encrypted data. It achieves a form of *multi-level access*, where different users have different access privileges to the data. The users are divided into groups, each one receiving a dictionary of keywords that it is allowed to search for. The user's search queries are restricted to this dictionary of keywords only, hence the users are restricted to search over a portion of the encrypted data.

A scheme by Cheng et al. [51] is the first scheme to apply indistinguishability obfuscation (IO) [16] in the SSE setting. The authors also show an extension to the scheme which supports conjunctive queries. In this scheme the IO circuit is applied to the index. Search queries are then put in to this IO circuit which produces the search results. To verify search results a public verification circuit is created, into which a verification key is embedded. IO is then applied to this verification circuit which allows the data owner to verify the search results. The leakage of this scheme is similar to that of [55].

Another approach in [102] extends a public key encryption with keyword search scheme [33] to support verification of search results from a single keyword equality query, where the indexes are created using a public key.

Several verifiable search schemes have been proposed that use ABE in their construction [102, 129, 137]. The schemes use Bloom filters to perform the searches on the encrypted data. A signed Bloom filter is included in the search results so the data owner is able to verify whether the search was performed correctly or not. The ABE scheme is used to control the access to the encrypted data, i.e. a user can only search a file if they have the correct credentials. The scheme of [129] allows the user to verify the freshness of the search results, which ensures the search results are from the latest version of the data and have not been recycled from previous searches. It uses signed Bloom filters similarly to [102] in order to achieve this.

Another dynamic scheme is presented in [36]. It extends the dynamic SSE scheme of [127] to support verifiability of search results. The scheme uses verifiable hash tables to allow the data owner to verify search results.

Following our work the notion of public verifiability has also been explored further by Azraoui et al. [9]. The authors present a scheme that uses Cuckoo hashing to index the encrypted data, which is authenticated using polynomial-based accumulators and Merkle tree. The root of the Merkle tree is used to verify the search results.

Verifiable computation

Verifiable computation allows a client with limited resources to efficiently outsource a computation to a more powerful server, and to verify the correctness of results. Gennaro et al. [67] considered the use of garbled circuits, whilst Parno et al. [115] introduced *publicly* verifiable computation (PVC) built from key policy attribute based enryption (KP-ABE), where a single client computes an evaluation key for the server and publishes information enabling other clients to outsource computation to the server. *Any* client may verify the correctness of a result. Alderman et al. [5] considered an alternative system model that used ciphertext policy attribute based encryption (CP-ABE) to allow clients to query computations on data held by the server (or initially outsourced by a client) called *Verifiable Delegable Computation* (VDC). This can naturally be applied to problems like querying on remote data, as well as MapReduce. Data remains statically stored on the server and may be embedded in a server's secret key, whilst the computation of many different functions can be requested by creating ciphertexts using *only* public information. Other notable approaches in the realm of querying remote data can be found in [8, 11, 10, 23, 24, 28, 53].

5.1.2 Organisation of chapter

In Section 5.2 we describe the system model and give a formal definition of eVSE, we also describe the types of search queries eVSE supports. Section 5.2.3 defines our notions of security along with the respective security games. In Section 5.3 we give the construction of our eVSE scheme and Section 5.4 details the security proofs to show that eVSE is secure according to our security notions defined in Section 5.2.3. Finally, Section 5.5 summarises and concludes the chapter.

5.2 Extended Verifiable Searchable Encryption

This section describes our system model and gives a formal definition for $e\mathcal{VSE}$.

5.2.1 System model

We consider a system model comprising a *data owner*, a remote storage *server*, and a set of authorised *users* as described in Section 3.1. The architecture of our scheme makes it suitable for use in Scenario 3 (Section 4.3.3). The data owner sets up the system to generate a master secret key and holds a set of data items (e.g. a database or a set of data items) that they wish to encrypt and outsource to the remote server. The data owner creates the index by assigning meta-data in the form of keywords or numeric values to the data items and encoding these values to form the encrypted index. The data owner can also control which users are able to search over the encrypted data by issuing user secret keys to authorized users which are necessary to construct the search queries. The users form the search queries using the same encoding of keywords and numeric values as the data used to construct the encrypted index. Queries can be formulated to locate a set of data items associated with specified keywords as in standard SE schemes, however eVSE also permits computational queries of functions in the class NC^1 which consists of Boolean functions computable by circuits of depth $\mathcal{O}(\log n)$ where each gate has a fan-in of two (where n is the number of data items), over encoded data values. The query is sent to the server who performs the query on the encoded index to generate a result. We allow *any* entity to verify the correctness and completeness of the result (public verifiability, Section 5.2.3). The reason we permit the server to verify correctness is to avoid the rejection problem, where a server may learn some useful information by observing if results are accepted, but we restrict the ability to read the value of the result to only authorised data users (holding a retrieval key). We assume a separation between the server entity and the user and data owners entities as we do not allow the server to issue queries. This prevents the server from trivially learning the encoding used for keywords to create the encrypted index. Users that are authorized to query the encrypted data must know this encoding in order to form their queries.

Users are added and revoked from the system using broadcast encryption. When enrolled each user is given a secret key related to the broadcast encryption scheme which allows them to create valid search queries.

As a practical example consider workgroups within an organisation. The manager or system administrator acts as the data owner for the organisation and outsources a shared database to a remote server. Authorisation is granted by issuing a secret key to each user. This key is used when creating a query token for a particular query Q. As well as each user being able to generate their queries at will using their secret key, the data owner is also able to generate queries and issue them to particular workgroups based on their roles and responsibilities. Each workgroup would then be authorised to query the encrypted data using only the queries generated for them by the data owner. This allows the data owner to restrict what information is released from the encrypted data to that particular workgroup. This is a form of multi-level access (MLA) (see Chapter 6).

5.2.2 Formal Definition

We now formally define our eVSE scheme. We use the following notation. Data to be outsourced is denoted $\mathcal{D}$ and is considered to be a collection of n data items. Prior to outsourcing, the data owner specifies a *pre-index* for $\mathcal{D}$, denoted $\delta(\mathcal{D})$, which assigns a set of descriptive labels to each data item, e.g. keywords contained in the data item or specific data values that may be computed upon. The encoded form of the data, including the descriptive labels, is referred to as the *index* of $\mathcal{D}$, denoted $\mathcal{I}_{\mathcal{D}}$, and is stored by the server. Queries for functions in the class NC^1 are denoted by Q and to make such a query, a user creates a query token QT_Q for Q, a verification key VK_Q which allows *any* entity to blindly verify the result, R, of the query, and a retrieval key RK_Q which is issued to authorised data users to enable the query result r to be learnt.

We consider a broader range of verifiable queries than many prior VSE schemes. In particular, we consider two main types:

- **Keyword matching queries**: Queries of this type have formed the basis of most prior work in SE. Suppose there exists a universe (dictionary) of keywords. Each encrypted data item is associated with an *index* of one or more keywords which describe its contents. Queries are formed over the same universe of keywords. In this work, we permit Boolean formulae over sets of keywords (e.g. $((\mathtt{a} \wedge \mathtt{b}) \vee \mathtt{c})$ where $\mathtt{a}, \mathtt{b}, \mathtt{c}$ are keywords). Thus we can perform very expressive search queries over keywords.

- **Computational queries**: Queries of this type are similar to the operations commonly discussed in the context of outsourced computation. We allow statistical queries over keywords (e.g. counting the number of data items that satisfy a keyword matching query), as well as operations over selected data values that have been encoded using additional portions of the keyword universe. It is possible to encode the entire database in such a way as to enable computations over all data fields, but it would usually be more efficient to select a (small) subset of fields that are most useful or most frequently queried. Clearly, keyword matching queries can be seen as a special case of computational queries where the function operator is equality testing.

- **Mixed queries**: Queries of this type combine both the functionalities of the aforementioned query types (e.g. finding the average of data values contained in all data items associated with a particular keyword).

All types of query are performed in a verifiable manner to ensure that results are correct and complete.

Definition 5.2.1 (Extended Verifiable Searchable Encryption (eVSE))**.** *An eVSE scheme comprises the following algorithms:*

- $(K_O, K_S, PP) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa, \Omega, S)$: Run by the data owner and takes as input the unary representation of the security parameter κ, and a universe of attributes Ω (keywords and data values) along with the server identity, S. It outputs the data owner's secret key K_O that is used for further administrative tasks, the server secret key K_S and public parameters PP.

- $(\mathcal{I}_\mathcal{D}) \overset{\$}{\leftarrow} \mathsf{BuildIndex}(\delta(\mathcal{D}), K_O, PP)$: Run by the data owner and takes as input the pre-index of the data $\delta(\mathcal{D})$ and the data owner's secret key K_O. It outputs a searchable index $\mathcal{I}_\mathcal{D}$ for the data $\mathcal{D}$.

- $(K_u, K_O) \overset{\$}{\leftarrow} \mathsf{AddUser}(u, K_O, PP)$: Run by the data owner to authorise a user with identity u to produce queries by issuing them a secret key K_u and updates the data owner's key K_O.

- $(QT_Q, VK_Q, RK_Q) \overset{\$}{\leftarrow} \mathsf{Query}(Q, K_u, PP)$: Run by a user using its secret key to generate a query token QT_Q for a query Q, a verification key VK_Q and an output retrieval key RK_Q.

- $R \overset{\$}{\leftarrow} \mathsf{Search}(\mathcal{I}_\mathcal{D}, QT_Q, K_S, PP)$: Run by the server to execute a query given in the query token QT_Q on the index $\mathcal{I}_\mathcal{D}$. It generates a result R which can be returned to the querying user or published.

- $RT_Q \leftarrow \mathsf{Verify}(R, VK_Q, PP)$: Run by *any* party to verify the correctness and completeness of the result R. It takes the verification key VK_Q and, if the result is accepted, it outputs a retrieval token RT_Q which can be used to learn the result. Otherwise a distinguished failure symbol $RT_Q = \perp$ is returned.

- $r \leftarrow \mathsf{Retrieve}(VK_Q, RT_Q, RK_Q, PP)$: Run by a data user to read the value of the result. It takes as input the retrieval token RT_Q, the retrieval key RK_Q and the user's secret key. If the user holds a valid retrieval key for Q and the computation was performed correctly, then it returns the actual result $r = Q(\mathcal{I}_\mathcal{D})$, otherwise it returns $r = \perp$.

- $(K_O) \overset{\$}{\leftarrow} \mathsf{RevokeUser}(u, K_O, PP)$: Run by the data owner using its secret key to revoke a user's authorisation to make queries and read results. It does so by updating the server's and data owner's key.

Definition 5.2.2 (Correctness of eVSE)**.** *An eVSE scheme is correct if for all* $\kappa \in \mathbb{N}$*, for all* $(MK, PP) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa, \Omega, S)$*, for all* u *there is a negligible probability that* *verification does not succeed when all algorithms are run honestly. A formal definition follows.*

An Extended Verifiable Searchable Encryption scheme is correct *for a family of queries $\mathcal{Q}$ if for all queries $Q \in \mathcal{Q}$ and for all data sets $\mathcal{D}$:*

$$\begin{aligned}
&\mathbb{P}[(K_O, K_S, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, \Omega, S),\\
&\quad \mathcal{I}_\mathcal{D} \xleftarrow{\$} \mathsf{BuildIndex}(\delta(\mathcal{D}), K_O, PP),\\
&\quad (K_u, K_O) \xleftarrow{\$} \mathsf{AddUser}(u, K_O, PP),\\
&\quad (QT_Q, VK_Q, RK_Q) \xleftarrow{\$} \mathsf{Query}(Q, K_u, PP),\\
&\quad R \xleftarrow{\$} \mathsf{Search}(\mathcal{I}_\mathcal{D}, QT_Q, K_S, PP),\\
&\quad RT_Q \leftarrow \mathsf{Verify}(R, VK_Q, PP),\\
&\quad Q(\mathcal{I}_\mathcal{D}) \leftarrow \mathsf{Retrieve}(VK_Q, RT_Q, RK_Q, PP)]\\
&\quad = 1 - \mathrm{negl}(\kappa)
\end{aligned}$$

5.2.3 Security Model

We now formalise several notions of security for eVSE as a series of cryptographic games. The adversary against each notion is modelled as a probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ run by a challenger, with input parameters chosen to represent the knowledge of a real attacker as well as the security parameter κ. The adversary $\mathcal{A}$ may maintain state and be multi-stage; we refer to each stage as $\mathcal{A}$ for ease of notation. The Public Verifiability game (Game 5.1), the index privacy game (Game 5.2) and the query privacy game (Game 5.3) all have the eVSE scheme eVSE as defined in Section 5.3 as input, along with the following additional inputs: the security parameter 1^κ, the universe of attributes Ω and the server identity S. We denote this additional set of inputs as X. We assume that oracle queries are performed in a logical order such that all required information is generated from previous queries.

Public Verifiability.

In Game 5.1, we capture the notion of *public verifiability* such that a server may not cheat by only executing a fraction of the specified query and/or returning an incorrect or incomplete result without being detected.

The notation $\mathcal{A}^\mathcal{O}$ denotes the adversary being provided with oracle access to the following oracles: $\textsc{BuildIndex}(\cdot, K_O, PP)$, $\textsc{AddUser}(\cdot, K_O, PP)$, $\textsc{Query}(\cdot, \cdot, PP)$ and $\textsc{Search}(\cdot, \cdot, \cdot, PP)$ as follows:

- The $\textsc{BuildIndex}(\cdot, K_O, PP)$ oracle allows the adversary to create indexes of their choosing. It simply runs $\mathsf{BuildIndex}$ on the adversary's chosen input (a pre-index $\delta(\mathcal{D})$ and returns the output to the adversary.

- The $\textsc{AddUser}(\cdot, K_O, PP)$ oracle allows the adversary to request a user be added to the system, and the adversary corrupts this user by receiving their user key. The $\mathsf{AddUser}$ algorithm is run on the adversary's chosen input, a user identity u and the output is returned to the adversary.
- The $\textsc{Query}(\cdot, \cdot, PP)$ oracle allows the adversary to generate search queries. The Query algorithm is run on the adversary's input, a user secret key and a query of their choosing. The output is returned to the adversary.
- The $\textsc{Search}(\cdot, \cdot, \cdot, PP)$ oracle allows the adversary to evaluate search queries over an index of their choosing. The Search algorithm is run on the adversary's chosen inputs and the results returned to the adversary.

This is a selective notion of security where, at the beginning of the game, the adversary chooses the challenge query and set of data items. The challenger runs KeyGen to generate the data owner key, the server key and the public parameters. The challenger then chooses a user identity at random from the universe of user identities $\mathcal{U}$ and authorises this user to search the encrypted data by running $\mathsf{AddUser}$ using the identity u as input. The challenger then encodes the adversary's challenge data item set and runs $\mathsf{BuildIndex}$ and Query on the inputs selected by the adversary at the beginning of the game to generate the index and the query for the adversary. The query is generated using the authorised user identity generated previously. The adversary is then given oracle access along with the outputs from $\mathsf{BuildIndex}$ and Query, which are $\mathcal{I}^*_{\mathcal{D}}$ and $(QT_{Q^*}, VK_{Q^*}, RK_{Q^*})$ respectively, the server key K_S and the public parameters PP. The adversary eventually outputs a result R^* which it believes to be an incorrect result that will, nevertheless, be accepted by the verifier. The challenger runs the Verify algorithm using R^* as input and following this runs $\mathsf{Retrieve}$. The adversary wins if verification succeeds, yet the result, r^* generated by $\mathsf{Retrieve}$, is not $Q(\mathcal{I}_{\mathcal{D}^\star})$.

Definition 5.2.3. *The* advantage *of a PPT adversary $\mathcal{A}$ in the public verifiability game (Game 5.1) is defined as follows:*

$$Adv_{\mathcal{A}}^{\mathbf{PublicVerifiability}}(\mathsf{eVSE}, X) = \left|\mathbb{P}[\mathbf{Exp}_{\mathcal{A}}^{\mathbf{PublicVerifiability}}[\mathsf{eVSE}, X] = 1] - \tfrac{1}{2}\right|.$$

eVSE *is* secure in terms of public verifiability *if for all PPT adversaries $\mathcal{A}$:*

$$Adv_{\mathcal{A}}^{\mathbf{PublicVerifiability}}(\mathsf{eVSE}, X) \leq \operatorname{negl}(\kappa)$$

where negl *is a negligible function.*

```
$\mathbf{Exp}_{\mathcal{A}}^{\mathbf{PublicVerifiability}}[\mathsf{eVSE}, X]$
1 :  $(Q^*, \mathcal{D}^\star) \leftarrow \mathcal{A}(1^\kappa, \Omega)$
2 :  $(K_O, K_S, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, \Omega, S)$
3 :  $u \xleftarrow{\$} \mathcal{U}$
4 :  $(K_u, K_O) \xleftarrow{\$} \mathsf{AddUser}(u, K_O, PP)$
5 :  $\delta(\mathcal{D}^*) \xleftarrow{\$} \mathsf{Encode}(\mathcal{D}^*)$
6 :  $\mathcal{I}_{\mathcal{D}^\star} \xleftarrow{\$} \mathsf{BuildIndex}(\delta(\mathcal{D}^\star), K_O, PP)$
7 :  $(QT_{Q^*}, VK_{Q^*}, RK_{Q^*}) \xleftarrow{\$} \mathsf{Query}(Q^*, K_u, PP)$
8 :  $R^\star \leftarrow \mathcal{A}^{\mathcal{O}}(QT_{Q^*}, VK_{Q^*}, RK_{Q^*}, \mathcal{I}_{\mathcal{D}^\star}, K_S, PP)$
9 :  $RT_{Q^*} \leftarrow \mathsf{Verify}(R^\star, VK_{Q^*}, PP)$
10 : $r^* \leftarrow \mathsf{Retrieve}(VK_{Q^*}, RT_{Q^*}, RK_{Q^*}, PP)$
11 : if $r^* \neq \perp$ and $r^* \neq Q^*(\mathcal{I}_{\mathcal{D}^\star})$ then
12 :   return 1
13 : else
14 :   return 0
```

Game 5.1: Selective public verifiability game for eVSE

Index Privacy.

In Game 5.2, we formalise the notion of index indistinguishability against a selective chosen keyword attack, which ensures no information regarding the keywords is leaked from the index.

The notation $\mathcal{A}^{\mathcal{O}}$ denotes the adversary being provided with oracle access to the following oracles: BuildIndex$(\cdot, K_O, PP)$, Search$(\cdot, \cdot, \cdot, PP)$, Verify$(\cdot, \cdot, \cdot, \cdot, PP)$ and Retrieve$(\cdot, \cdot, \cdot, PP)$ as follows:

- The BuildIndex oracle allows the adversary to create indexes of their choosing. It runs BuildIndex on the adversary's chosen input. As the adversary is modelled as the server in our security model it does not have access to the encoded set of attributes $\tilde{\Omega}$. Hence, this oracle has to execute Encode on the adversary's chosen data set prior to generating the index for that data set. The resulting index generated using the encoded data set as input to the BuildIndex algorithm is returned to the adversary.

- The Query oracle allows the adversary to generate search queries of their choosing. It runs Query on the adversary's input to generate the corresponding query token, QT_Q, verification key, VK_Q and retrieval key, RK_Q. It is required that these search queries produce the same search results when evaluated on either of the challenge indexes i.e. for any Q submitted to Query it is required that if $R_0 \leftarrow \mathsf{Search}(\mathcal{I}_{\mathcal{D}_0}, QT_Q, K_S, PP)$ and $R_1 \leftarrow \mathsf{Search}(\mathcal{I}_{\mathcal{D}_1}, QT_Q, K_S, PP)$ then we need $R_0 = R_1$. If this is not the case then Query outputs 0, otherwise QT_Q, VK_Q and RK_Q are returned to the adversary.

- The Search algorithm evaluates search queries of the adversary's choosing over an index chosen by the adversary. The oracle runs Search on the adversary's chosen inputs and outputs the search results R.
- The Verify oracle allows the adversary to verify search results. The oracle runs Verify on the adversary's inputs and returns the results to the adversary.
- The Retrieve oracle allows the adversary to retrieve search results from the encoded versions output by Verify. The oracle runs Retrieve on the adversary's inputs and returns the results to the adversary.

The game (Game 5.2) begins with the adversary outputting two data sets ($\mathcal{D}_0, \mathcal{D}_1 \subseteq \Omega$) that they wish to be challenged on, with the restriction that $|\mathcal{D}_0| = |\mathcal{D}_1|$ (this is required as the CP-ABE scheme that will be used to produce the index does not conceal the index length). The challenger runs KeyGen to produce the public and secret parameters and selects a bit $b \in \{0, 1\}$ uniformly at random to determine which data set to encode into the index. Before the index is created, the challenger needs to create the pre-index corresponding to the data set $\mathcal{D}_b$. This is done using an Encode mechanism that takes the elements of $\mathcal{D}_b$ as input and outputs the pre-index $\delta(\mathcal{D}_b)$. Encode is not required in our instantiation as the pre-indexes can be chosen directly from $\tilde{\Omega}$ as the user is able to encode the data set using its secret key which the adversary does not have access to. The challenger then runs BuildIndex using $\delta(\mathcal{D}_b)$ to produce the index $\mathcal{I}_{\mathcal{D}_b}$. The challenger then chooses a user identity u from the userspace and authorises that identity to search over the encrypted data by running AddUser with u as input. The adversary is then given the index $\mathcal{I}_{\mathcal{D}_b}$ along with the server key K_S and the public parameters PP and allowed oracle access as specified. After this query phase, $\mathcal{A}$ outputs a guess b' and wins the game if the game returns 1 which indicates that $b' = b$. Hence $\mathcal{A}$ wins the game if they can identify which attribute set ($\mathcal{D}_0$ or $\mathcal{D}_1$) was encoded into the index $\mathcal{I}_{\mathcal{D}_b}$.

Definition 5.2.4. *The* advantage *of a PPT adversary $\mathcal{A}$ in the index privacy game (Game 5.2) is defined as follows:*

$$Adv_{\mathcal{A}}^{\textbf{IndexPrivacy}}(\mathsf{eVSE}, X) = \left|\mathbb{P}[\mathbf{Exp}_{\mathcal{A}}^{\textbf{IndexPrivacy}}[\mathsf{eVSE}, X] = 1] - \tfrac{1}{2}\right|.$$

```
$\mathbf{Exp}_{\mathcal{A}}^{\mathbf{IndexPrivacy}}[\mathsf{eVSE}, X]$
1 : $(\mathcal{D}_0, \mathcal{D}_1) \leftarrow \mathcal{A}(1^\kappa, \Omega)$
2 : if $|\mathcal{D}_0| \neq |\mathcal{D}_1|$
3 :     return 0
4 : $(K_O, K_S, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, \Omega, S)$
5 : $b \xleftarrow{\$} \{0,1\}$
6 : $\delta(\mathcal{D}_b) \xleftarrow{\$} \mathsf{Encode}(\mathcal{D}_b)$
7 : $\mathcal{I}_{\mathcal{D}_b} \xleftarrow{\$} \mathsf{BuildIndex}(\delta(\mathcal{D}_b), K_O, PP)$
8 : $u \xleftarrow{\$} \mathcal{U}$
9 : $(K_u, K_O) \xleftarrow{\$} \mathsf{AddUser}(u, K_O, PP)$
10 : $b' \leftarrow \mathcal{A}^{\mathcal{O}}(\mathcal{I}_{\mathcal{D}_b}, K_S, PP)$
11 : if $b' = b$ then
12 :     return 1
13 : else
14 :     return 0
```

Game 5.2: Selective index privacy game for eVSE

eVSE *is* secure in terms of index privacy *if for all PPT adversaries $\mathcal{A}$:*

$$Adv_{\mathcal{A}}^{\mathbf{IndexPrivacy}}(\mathsf{eVSE}, X) \leq \mathrm{negl}(\kappa)$$

where negl *is a negligible function.*

Query Privacy.

The queries themselves should not leak any information about the corresponding keywords that make up the query. Our construction of the queries leaks the gates in the query (i.e. whether $\vee$, $\wedge$ etc.), but not the keywords themselves. This notion of query indistinguishability against a selective chosen query attack is formalised in Game 5.3. The game runs similarly to that of Game 5.2, subject to the following restrictions: the challenge queries (Q_0, Q_1) must use the same gates. We denote the gate structure of a query Q by $\mathbb{G}_Q$, and hence require that $\mathbb{G}_{Q_0} = \mathbb{G}_{Q_1}$.

The notation $\mathcal{A}^{\mathcal{O}}$ denotes the adversary being provided with oracle access to the following oracles: BuildIndex$(\cdot, K_O, PP)$, Search$(\cdot,\cdot,\cdot, PP)$, Verify$(\cdot,\cdot,\cdot,\cdot, PP)$ and Retrieve$(\cdot,\cdot,\cdot, PP)$ as follows:

- The BuildIndex oracle allows the adversary to create indexes of their choosing. It is required that the two challenge search queries Q_0, Q_1 produce the same search

```
$\mathbf{Exp}_{\mathcal{A}}^{\mathbf{QueryPrivacy}}[\mathsf{eVSE}, X]$
1 :  $(Q_0, Q_1) \leftarrow \mathcal{A}(1^\kappa, \Omega)$
2 :  if $\mathbb{G}_{Q_0} \neq \mathbb{G}_{Q_1}$
3 :    return 0
4 :  $(K_O, K_S, PP) \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa, \Omega, S)$
5 :  $u \xleftarrow{\$} \mathcal{U}$
6 :  $(K_u, K_O) \xleftarrow{\$} \mathsf{AddUser}(u, K_O, PP)$
7 :  $b \xleftarrow{\$} \{0, 1\}$
8 :  $(QT_{Q_b}, VK_{Q_b}, RK_{Q_b}) \xleftarrow{\$} \mathsf{Query}(Q_b, K_u, PP)$
9 :  $b' \leftarrow \mathcal{A}^{\mathcal{O}}(QT_{Q_b}, VK_{Q_b}, RK_{Q_b}, K_S, PP)$
10 : if $b' = b$ then
11 :   return 1
12 : else
13 :   return 0
```

Game 5.3: Selective query privacy game for eVSE

results when evaluated on any index produced by BUILDINDEX i.e. for any $\mathcal{D}$ submitted to BUILDINDEX it is required that if $R_0 \leftarrow \mathsf{Search}(\mathcal{I}_{\mathcal{D}}, QT_{Q_0}, K_S, PP)$ and $R_1 \leftarrow \mathsf{Search}(\mathcal{I}_{\mathcal{D}}, QT_{Q_1}, K_S, PP)$ then we need $R_0 = R_1$. The oracle runs BuildIndex on the adversary's chosen input. As the adversary is modelled as the server in our security model it does not have access to the encoded set of attributes $\tilde{\Omega}$. Hence, this oracle has to execute Encode on the adversary's chosen data set prior to generating the index for that data set. The resulting index generated using the encoded data set as input to the BuildIndex algorithm is returned to the adversary if we have that both challenge queries produce the same search results when evaluated over the index. Otherwise the oracle returns 0.

- The SEARCH, VERIFY and RETRIEVE oracles are run as described for the index privacy game (Game 5.2).

The adversary chooses their challenge queries. The challenger returns 0 if the gates are not equal on these queries, if they are equal then the game proceeds. The challenger runs KeyGen to produce the public and secret parameters and then selects a user identity randomly from the userspace. This user identity is then used as input to AddUser by the challenger to authorise that user identity to search over the encrypted data. The challenger then selects a bit b uniformly at random and creates the challenge query for the adversary by running $\mathsf{Query}(Q_b, K_u, PP)$. This query, Q_b is given to the adversary, along with the server key K_S and the public parameters. The adversary is given access to the oracles as specified. The adversary eventually outputs a bit b' and wins the game if $b' = b$. Hence the adversary wins the game if they can identify which query was given to them by the challenger.

Definition 5.2.5. *The* advantage *of a PPT adversary $\mathcal{A}$ in the query privacy game (Game 5.3) is defined as follows:*

$$Adv_{\mathcal{A}}^{\textbf{QueryPrivacy}}(\mathsf{eVSE}, X) = \left|\mathbb{P}[\mathbf{Exp}_{\mathcal{A}}^{\textbf{QueryPrivacy}}[\mathsf{eVSE}, X] = 1] - \tfrac{1}{2}\right|.$$

eVSE *is* secure in terms of query privacy *if for all PPT adversaries $\mathcal{A}$:*

$$Adv_{\mathcal{A}}^{\textbf{QueryPrivacy}}(\mathsf{eVSE}, X) \leq \text{negl}(\kappa)$$

where negl *is a negligible function.*

5.3 Construction

5.3.1 Overview

We base our instantiation on a CP-ABE scheme. As shown in [6], CP-ABE can be used to verifiably request computations to be performed on data held by a server, referred to as VDC (see Section 2.4.3). In VDC, a trusted Key Distribution Center (KDC) initialises the system and issues a CP-ABE decryption key to the server pertaining to the data it holds. We use a similar technique, but have the data owner act as the KDC (so the data items need not be revealed to an external KDC, as in VDC). The index for a set of data items is a CP-ABE decryption key for a set of attributes encoding the pre-index, and is sent to the server. The method of encoding is described in Section 5.3.4, we require this encoding as CP-ABE is not by definition *attribute hiding* hence we need to encode the keywords and attributes used into the pre-index first before using them to generate the CP-ABE secret key (index). This ensures the privacy of the index.

A query Q is represented as a Boolean function of encoded keywords and computational data points. We consider the family $\mathcal{B}$ of Boolean functions closed under complement – that is, if $F \in \mathcal{B}$ then $\overline{F}$, where $\overline{F}(x) = F(x) \oplus 1$, is also in $\mathcal{B}$. A function $F : \{0,1\}^n \to \{0,1\}$ is *monotonic* (specifically *montonic increasing*) if $x \leqslant y$ implies $F(x) \leqslant F(y)$, where $x = (x_1, \ldots, x_n) \leq y = (y_1, \ldots, y_n)$ if and only if $x_i \leqslant y_i$ for all i (for a *monotonic decreasing* function we have that $x \leqslant y$ implies $F(x) \geqslant F(y)$). In the context of CP-ABE, this means that if a user has a set of attributes X that satisfies F (i.e. $F(X) = 1$) then any user with an attribute set $Y : X \subseteq Y$ will also satisfy F.

For a monotonic function F, the set $\mathbb{A}_F = \{x : F(x) = 1\}$ defines a monotonic access structure (the set of all sets of attributes that satisfy F).

If a monotonic CP-ABE scheme is used then queries can be comprised of `AND` and `OR` gates (and negation can inefficiently be handled by including both a positively and negatively labelled attribute in the universe and requiring the presence of exactly one of them). To see why a CP-ABE scheme with a monotonic access structure cannot support queries comprising of `NOT` gates is shown by the following example: consider three users with attribute sets $A = \{a_1, a_2, a_3, a_4\}$, $B = \{a_1, a_2, a_3\}$ and $C = \{a_1\}$. Let the Boolean function $F = (a_2 \wedge \neg a_4)$; we have that $B \subseteq A$ and $F(B) = 1$ and $F(A) = 0$ i.e. $F(B) \geq F(A)$ however $C \subseteq B$ but $F(C) = 0$ and $F(B) = 1$ hence $F(C) \leq F(B)$. This shows that the function F, which contains a negation, is not monotonic.

A non-monotonic CP-ABE scheme enables queries formed from `AND`, `OR` and `NOT` gates, which is a universal set of gates, as they permit the queries to be formed from non-monotonic functions. We can achieve all functions in the class NC^1, which includes common arithmetic and comparison operators useful in queries. An n-bit result can be formed by performing n Boolean queries, each of which returns the i^{th} bit of the output, see Section 5.3.4.

The query token for a Boolean function $Q \in \mathcal{B}$ comprises two CP-ABE ciphertexts for access structures representing Q and $\overline{Q} \in \mathcal{B}$ respectively. The user chooses two random plaintexts m_0 and m_1 from the plaintext space which will act as verification tokens. The user also chooses a random bit b which is used to permute the ciphertexts. The plaintexts m_b and m_{1-b} are encrypted under the queries $Q \in \mathcal{B}$ and $\overline{Q} \in \mathcal{B}$ respectively and the resulting ciphertexts are sent to the server as the query token QT_Q. The user also generates a verification key VK_Q by applying a one-way function g to each plaintext, and the bit b forms an output retrieval key RK_Q. To perform the search, the server attempts to decrypt each ciphertext under the secret key (which forms the index). Now, by definition, the data attributes in the key will satisfy precisely *one* of Q or $\overline{Q}$. Thus, precisely one plaintext will be correctly decrypted and the other decryption will return $\perp$. This pair of decryptions is returned as the encoded result R.

Any entity may perform the verification operation using the verification key VK_Q. To determine correctness of the result, the same one-way function g is applied to the returned results and compared to those values contained in VK_Q. If either matches, then this value is output as the retrieval token RT_Q and the result is accepted; otherwise the verifier rejects the result.

Finally, an authorised entity that has been granted access to the retrieval key RK_Q may apply this to the token RT_Q to determine the permutation of the ciphertexts and thus whether the returned plaintext corresponds to Q or to $\overline{Q}$. If m_0 is the returned plaintext then the result r is set to 1, and $r = 0$ otherwise.

5.3.2 Choosing a Broadcast Encryption scheme

Our scheme, eVSE (also our multi-level symmetric searchable encryption scheme detailed in Chapter 6), uses a BE scheme as a black box. There are many types of BE schemes described in the literature, in this section we discuss which types of BE schemes are most suited to our applications.

We require a BE scheme which supports just one entity, the data owner in our case, who is able to create the BE ciphertexts. The data owner updates the value of st_O (the data owner state) each time a user is revoked from the system and then encrypts it using BE to create the server state, st_S. When choosing a BE scheme that is suitable for this purpose we have to consider variables such as the number of users (both authorised and revoked), bandwidth required to transfer the BE ciphertexts and the storage required by a user.

BE schemes allow encrypted data to be delivered to a large set of users, so that only a particular subset of privileged users can decrypt it. There are two main types of BE scheme: stateless and stateful. A stateful BE scheme provides users with keys that may be updated when a user is either revoked or added to the system. These schemes require users to be online in order to receive update messages. A stateless BE scheme provides users with long term keys that do not change over the lifespan of the system, a user can be revoked without having to change or update the keys of currently authorised users.

BE schemes can also be either public key or symmetric key. Public key BE schemes are more suited to scenarios where more than one user is creating the BE ciphertexts, as the encryption key distribution is simplified by the use of public keys. Symmetric key BE schemes are more suited to scenarios where just one entity is creating the BE ciphertexts.

In our scenario users may not be online at all times, so a stateless BE scheme seems more practical, and as there is just one entity creating the BE ciphertexts, a symmetric key BE scheme would be well-suited.

Stateless BE was first investigated by Naor, Naor and Lotspiech. The complete subtree (CS) method for achieving BE was proposed by Naor et al. [104]. It models each user as a leaf of a binary tree (hence it is required that the total amount of users is a power of 2). Every node in this tree is associated with a distinct key, and a user receives each key associated with each node from the root to their leaf node. In order to encrypt a message for only authorised users, several ciphertexts need to be created. To determine the keys needed to create these ciphertexts the minimal cover of authorised users' leaves in the binary tree needs to be calculated. The keys associated with these nodes that provide the minimum cover are used to create the ciphertexts.

Another method of BE known is known as the Subset Difference (SD) method, which was also presented by Naor et al. [104]. In this method the data owner (more generally known as the sender in BE) generates independent keys for each node in the tree (as in the CS method). Subsets of these keys are then distributed to the users. Instead of using the minimal cover to determine which keys to use to create the BE ciphertexts, the number of subsets is increased significantly to $\mathcal{O}(n^2)$, where n is the total number of users. The set of subsets is defined as follows:

$$S = \{S_{i,j} | v_i \text{ is an ancestor of } v_j\},$$

where:

$$S_{i,j} = \{\text{Descendants of } v_i\} \setminus \{\text{Descendants of } v_j\}.$$

Defining the subsets in this way allows a message to be encrypted for $n \backslash r$ authorised users (where r is the number of revoked users) with at most $2r - 1$ subsets (with the average number of subsets required being 1.25).

The SD method was improved by [74] which presents another method for achieving BE called the Least Subset Difference (LSD) method. This method is also tree-based and reduces the amount of keys that each user needs to store by nearly a square root factor. The BE schemes detailed in [27, 72] also improve on the work of [104] with Goodrich et al. [72] producing a scheme that reduces the storage required by the users, however this is at the cost of an increase in computation for the user.

There are two main types of tradeoff in BE schemes: reduced message length usually implies an increase in user storage and reduced user storage usually implies an increased message length. The type of scenario our scheme is implemented in will affect the choice of BE scheme. If the users have resource constrained devices then schemes that minimize the user storage might be preferable such as [72, 74]. Whereas for applications where the users' storage space is not limited, choosing a scheme with a smaller message length may be preferable, such as [133].

BE can be achieved trivially by encrypting the plaintext a number of times so there is a different BE ciphertext for each authorised user. These ciphertexts are then broadcast to all users, who each have a distinct key to decrypt their respective ciphertext. The aim of most BE schemes is improve on the efficiency of this trivial scheme. However, in some scenarios when the number of users is low, using the trivial BE scheme might be feasible. In eVSE (also our multi-level symmetric searchable

Scheme	Average message length	Storage per user	Computation per user, prior to decryption
-	$(n \setminus r)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
[104]	$r\log(\frac{n}{r})$	$\log n$	$\mathcal{O}(\log\log n)$
[104]	$1.38r$	$\frac{1}{2}\log^2 n + \frac{1}{2}\log n + 1$	$\mathcal{O}(\log n)$
[74]	2r	$\mathcal{O}(\log^{1+\epsilon} n)$	$\mathcal{O}(\log n)$
[27]	$2r - 1$	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(\log n)$
[72]	$1.25r$	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$
[133]	$< 1.38r$	$\mathcal{O}(2^\lambda \log n)$	$\mathcal{O}(\log n)$
[80]	$\mathcal{O}(\frac{r}{p} + \frac{n-r}{c})$	$\mathcal{O}(c^p)$	$\mathcal{O}(c-1)$

Table 5.1: Comparison of Broadcast Encryption Schemes

encryption scheme detailed in Chapter 6) it is only required that one $k-$bit value (st_O) is encrypted in the BE ciphertext, meaning the BE ciphertexts are not very large. Hence, requiring a number of BE ciphertexts linear in n (where n is the number of authorised users) might be feasible in a scenario where n is small.

5.3.3 Choosing a CP-ABE scheme

Our scheme, eVSE, uses a CP-ABE scheme as a black box. We have discussed a few requirements of the CP-ABE scheme previously, however here we discuss which CP-ABE schemes would be well suited for use with eVSE. We also discuss the features of our scheme that effect the choice of CP-ABE scheme used.

Due to the way the attribute universe is defined eVSE requires a large universe CP-ABE scheme. Large universe CP-ABE was first explored by Okamoto and Takashima [111]. This scheme is based on the dual pairing vector spaces framework and each attribute can take on several values chosen from a space of exponential size. There is a bound on the number of times each space can be used in a policy which is chosen when setting up the system and remains fixed throughout the life span of the system. Rouselakis and Waters [122] claim that as this bound increases the scheme becomes less efficient. In this same paper [122], Rouselakis and Waters present a large universe CP-ABE scheme which has no restriction on the attributes used in the policies and has constant size public parameters.The authors extended their scheme to a *multi-authority* one in [123]. This construction uses bilinear groups of prime order which optimizes the computation of the pairings used in the construction [73].

Other work [107] which uses techniques from [122], details a large universe CP-ABE scheme with white-box traceability.

Scheme	Encryption	Decryption	Storage per user (key size)	Monotonic/Non-monotonic
[122]	$(5\|\ell\|+2)E$	$\|I\|E+(3\|I\|+1)P$	$\mathcal{O}(\|k\|)$	Yes
[107]	$(3+5\|\ell\|)E$	$(2+\|I\|)E+(3\|I\|+1)P$	$\mathcal{O}(\|k\|)$	Yes

Table 5.2: Comparison of Large Universe CP-ABE schemes

When choosing a CP-ABE scheme to implement eVSE we consider the following to be important for optimizing the efficiency of our scheme: fast search query generation, search time and verification, and minimal storage per user. These features are all related to the user and streamlining their experience. We look at the following large universe CP-ABE schemes [107, 122] as discussed above and consider how they perform when considering these features. We omit the scheme of [111] due to the issues discussed in [122].

Looking at the Query algorithm in eVSE we can see that it requires the generation of n pairs of CP-ABE ciphertexts (where n is the number of encrypted data items). In order to optimize the query generation for the user a CP-ABE with the most efficient encryption algorithm would be preferable. In the Search algorithm the main bottleneck is the n pairs of CP-ABE decryptions, hence a CP-ABE scheme with the most efficient decryption algorithm would be preferable. Each user has to store their secret user key which consists of a CP-ABE user key (along with a secret key for a symmetric encryption scheme and a PRF and its key). Hence, choosing a CP-ABE with the shortest user keys would be recommended. The CP-ABE scheme is not used in the verification steps in eVSE, hence this can be omitted from our analysis.

Overall, the scheme of [122] is more efficient in the encryption and decryption algorithms, and both schemes have the same storage requirements for the user, hence this scheme would be recommended for use with eVSE, as it is also a large universe CP-ABE.

eVSE supports both monotonic and non-monotonic functions in the policies. If a monotonic CP-ABE scheme is used then negative attributes need to be included in the attribute universe, effectively doubling its size. An open area of research here would be to investigate possible large universe, non-montonic CP-ABE schemes.

5.3.4 Data Encoding

Defining the Index.

Suppose the data $\mathcal{D}$ to be outsourced comprises n data items. We now discuss how to form a *pre-index* $\delta(\mathcal{D})$, which represents the keywords and data fields that may be queried over.

Let Δ be a dictionary of keywords that describe the data items. Δ alone suffices for keyword matching queries but for computational queries, we also need to be able to encode data values such that they can be input to queries represented as access structures encoding Boolean functions.

For each data field x that may be input to a computational query, let the maximum size of the data value be m_x bits. We define m_x additional attributes $A_{x,1}, A_{x,2}, \dots, A_{x,m_x}$, and define the universe $\mathcal{C} = \bigcup_{x \in \mathcal{D}} \cup_{i=1}^{m_x} A_{x,i}$ to be the union of these attributes over all data fields. Let y be a value stored in the data field x and let the binary representation of y be $y_1, \dots, y_{m_x}$. We view y as a *characteristic tuple* of an attribute set $A_y \subseteq \mathcal{C}$, where $A_y = \{A_{x,i} : y_i = 1\}$ – we include an attribute for position i in the set if and only if the i^{th} bit of y is 1.

Finally, to enable the index for all n data items to be encoded within a single CP-ABE key (and hence for computations to be performed simultaneously on all data items), and to ensure that the correct index data is used for each query, we must encode a labelling of the data item that each attribute pertains to. We define our attribute universe Ω for the CP-ABE scheme to be $\Omega = \{\Delta \cup \mathcal{C}\} \times [n]$. That is, we take n copies of Δ and $\mathcal{C}$. Each element of Ω describes a particular keyword or data value, and each copy relates to a different data item in $\mathcal{D}$: if we index each copy of an attribute $w \in \Omega$ as $\{w_i\}_{i=1}^{n}$, then w_i denotes the presence of w in data item i. In practice, it may be desirable to use a 'large universe' CP-ABE scheme, wherein arbitrary textual strings are mapped to attributes (group elements), e.g. using a hash function H. Thus, for a keyword or data value w in data item i, the attribute could be defined as $\mathsf{H}(w||i)$.[1]

The pre-index of the data $\mathcal{D}$ is a set of attributes $\delta(\mathcal{D}) \subseteq \Omega$ and represent the sets of keywords and data values associated with the set of data items $\mathcal{D}$. We define an algorithm $\delta(\mathcal{D}) \leftarrow \mathsf{Encode}(\mathcal{D})$ that encodes a data set $\mathcal{D}$ in to the pre-index $\delta(\mathcal{D})$. The index that is outsourced will be a CP-ABE key generated over this attribute set in the pre-index.

Here we provide a simple example that shows how we define the index. Suppose we have three data items with the following characteristics:

[1]In this case, it may be possible to avoid the use of symmetric encryption in our construction by letting the secret k be the key for this cryptographic hash function.

- **Document 1:** *Keywords:* Male, Vaccinated. *Data:* Age $= 7 = 111_2$.
- **Document 2:** *Keywords:* Female. *Data:* Age $= 4 = 100_2$.
- **Document 3:** *Keywords:* Male, Vaccinated. *Data:* -

Then we define the index for these three data items as follows:

$$\begin{aligned}
\Delta =&\{\mathtt{Male}, \mathtt{Female}, \mathtt{Vaccinated}\},\\
\mathcal{C} =&\{\mathtt{A}_{\mathtt{Age},1}, \mathtt{A}_{\mathtt{Age},2}, \mathtt{A}_{\mathtt{Age},3}\},\\
\Omega =&\{\mathtt{Male}_{\mathtt{Doc1}}, \mathtt{Male}_{\mathtt{Doc2}}, \mathtt{Male}_{\mathtt{Doc3}},\\
&\mathtt{Female}_{\mathtt{Doc1}}, \mathtt{Female}_{\mathtt{Doc2}}, \mathtt{Female}_{\mathtt{Doc3}},\\
&\mathtt{Vaccinated}_{\mathtt{Doc1}}, \mathtt{Vaccinated}_{\mathtt{Doc2}}, \mathtt{Vaccinated}_{\mathtt{Doc3}},\\
&\mathtt{A}_{(\mathtt{Age},1),\mathtt{Doc1}}, \mathtt{A}_{(\mathtt{Age},1),\mathtt{Doc2}}, \mathtt{A}_{(\mathtt{Age},1),\mathtt{Doc3}},\\
&\mathtt{A}_{(\mathtt{Age},2),\mathtt{Doc1}}, \mathtt{A}_{(\mathtt{Age},2),\mathtt{Doc2}}, \mathtt{A}_{(\mathtt{Age},2),\mathtt{Doc3}},\\
&\mathtt{A}_{(\mathtt{Age},3),\mathtt{Doc1}}, \mathtt{A}_{(\mathtt{Age},3),\mathtt{Doc2}}, \mathtt{A}_{(\mathtt{Age},3),\mathtt{Doc3}}\},\\
\delta(\mathcal{D}) =&\{\mathtt{Male}_{\mathtt{Doc1}}, \mathtt{Vaccinated}_{\mathtt{Doc1}}, \mathtt{A}_{(\mathtt{Age},1),\mathtt{Doc1}}, \mathtt{A}_{(\mathtt{Age},2),\mathtt{Doc1}}, \mathtt{A}_{(\mathtt{Age},3),\mathtt{Doc1}},\\
&\mathtt{Female}_{\mathtt{Doc2}}, \mathtt{A}_{(\mathtt{Age},3),\mathtt{Doc2}},\\
&\mathtt{Male}_{\mathtt{Doc3}}, \mathtt{Vaccinated}_{\mathtt{Doc3}}\}.
\end{aligned}$$

Hiding the Index.

In general, CP-ABE schemes do not hide the attributes within the decryption key. This is usually expected behaviour since CP-ABE is often used to cryptographically enforce access control policies and it is natural to assume that an entity is aware of their access rights.

However, in this setting we are using CP-ABE not to protect objects from unauthorised access, but instead to prove the outcome of a function evaluation. The keys in our setting are formed over attributes encoding the index of outsourced data, as opposed to encoding access rights. Since the server should not learn any information about the data items, *including* the index, we must implement a mechanism by which the CP-ABE decryption key hides the attributes associated with it.

In many CP-ABE schemes, and in particular in that proposed by Waters et al. [134], the public parameters comprise an ordered set of group elements, each associated with an attribute from the universe; that is, $\forall i \in \Omega$, choose $t_i \xleftarrow{\$} \mathbb{Z}_p$, then form the encoded attribute set $\{g^{t_i}\}_{i \in \Omega}$. Thus, given a key (or ciphertext) that comprises g^{t_i}, it is possible, based on the ordering of this set, to determine the attribute $i \in \Omega$ it relates to. In addition, the attributes may be listed in the clear, and attached to keys and ciphertexts to indicate which group elements should be applied at each point. Clearly, this is unsuitable for our requirement for a hidden index.

To this end, we first apply a random permutation to Ω such that the position of the group elements within the ordered set does not reveal the attribute string (unless the permutation is known). We then use a symmetric encryption scheme to encrypt each attribute $x \in \Omega$ under a key k, and then instantiate the CP-ABE scheme on this universe of *encrypted* attributes. Thus, without knowledge of the key k, the server should be unable to determine the attribute x that a given group element corresponds to. We assume that only the keywords or data items being computed over are considered sensitive, and not the logical makeup of the Boolean function (in terms of gates).

5.3.5 Formal Details

The data owner initialises the system and encodes the data as an index which is pushed to the server. Each (authorised) user will be issued with a personalised secret key enabling them to form queries. Suppose that to make a query Q, a user chooses a random plaintext from the plaintext space $\mathcal{M}$ to act as a verification token, and encrypts this using the CP-ABE scheme under the access structure encoding Q. The server attempts to decrypt the ciphertext and recovers the chosen plaintext if and only if $Q(\mathcal{I}_{\mathcal{D}}) = 1$. By the indistinguishability security of the CP-ABE scheme, the server learns nothing about the plaintext if $Q(\mathcal{I}_{\mathcal{D}}) = 0$ since this corresponds to an access structure not being satisfied. Thus, if a server returns the correct plaintext, the user is assured that the query evaluated to 1 on the data. If, however, $Q(\mathcal{I}_{\mathcal{D}}) = 0$, then decryption will return $\perp$. This is insufficient for verification purposes since the server can return $\perp$ to convince a user of a false negative query result. Thus, the user must, in fact, produce two CP-ABE ciphertexts. As described in Section 5.3.1, one corresponds to the function Q, whilst the other corresponds to $\overline{Q}$, the complement query of Q. Hence, the CP-ABE secret key that forms the index will decrypt *exactly one* ciphertext and the returned plaintext will distinguish whether Q or $\overline{Q}$ was satisfied, and therefore the value of $Q(\mathcal{I}_{\mathcal{D}})$. A well-formed response (d_0, d_1) from a server, therefore, satisfies the following:

$$(d_0, d_1) = \begin{cases} (m_b, \perp), & \text{if } Q(\mathcal{I}_{\mathcal{D}}) = 1 \\ (\perp, m_{1-b}), & \text{if } Q(\mathcal{I}_{\mathcal{D}}) = 0. \end{cases} \tag{5.1}$$

Public Verifiability is achieved by publishing a token comprising a one-way function g applied to both plaintexts. Any entity can apply g to the server's response and compare with this token to check correctness. To achieve blind verification, a random bit b permutes the order of the ciphertexts. Thus, verifiers that do not know b cannot determine whether a plaintext is associated with Q or $\overline{Q}$.

Our adversarial model can allow the adversary (and hence server in our system) to hold more than one key (for multiple datasets); if this is the case we must ensure that a key cannot produce a valid looking response to a query on a different index. We can achieve this by labelling each pre-index with a label $l(\delta(\mathcal{D}))$ and define an attribute for each label. Then, for a pre-index $\delta(\mathcal{D})$, the decryption key is formed over the attribute set $(\delta(\mathcal{D}) \cup l(\delta(\mathcal{D})))$. Recall that encoded data stored in the server side index is a collection of n sets of attributes, each one corresponding to a data item, which we label $D_1, \ldots, D_n$. When making a query $Q(\mathcal{I}_{\mathcal{D}})$, a sub-query Q_i may be formed for each attribute set (e.g. to check if a given keyword is contained in each data item). In this case, the encryption algorithm takes the access structure encoding of the conjunction $(D_i \wedge l(\delta(\mathcal{D})))$ for $i \in [n]$. A valid result can only be formed by applying the sub-query to the specified attribute set. Decryption succeeds if and only if the function is satisfied *and* the label $l(\delta(\mathcal{D}))$ is matched in the key and ciphertext. Note that a key for a different pre-index will not include the correct label. For ease of notation we omit the labels from our construction, but we note that our scheme can be extended in this way to support a server that holds multiple datasets.

5.3.6 Instantiation Details

Let $\mathsf{CPABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Encrypt}, \mathsf{ABE.Decrypt})$ define a CP-ABE encryption scheme over the universe Ω. Let $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Encrypt}, \mathsf{SKE.Decrypt})$ be an authenticated symmetric encryption scheme secure in the sense of IND-CPA. Let $\mathsf{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Encrypt}, \mathsf{BE.Add}, \mathsf{BE.Decrypt})$ be a broadcast encryption scheme that retains IND-CPA security against a coalition of revoked users. Finally, let g be a one-way function and let Π and ϕ be pseudo-random permutations (PRPs) (with keys k_Π and st_O respectively). Then Algorithms 5.1–5.8 define an eVSE scheme for a class of queries $\mathcal{Q}$.

The KeyGen algorithm sets up the system and produces the secret key for the data owner and the public parameters. The data owner's secret key consists of seven parts: a secret key for CPABE, a secret key for BE, a secret key for SKE, a PRP Π, a key for Π, the data owner's state and the authorised user group. KeyGen starts by generating the initial two parts of the data owner's secret key by running $\mathsf{BE.KeyGen}$ and $\mathsf{SKE.KeyGen}$ to generate k_{BE} and k_{SKE} respectively. The algorithm then encrypts each element in the attribute universe using SKE, this encrypted set of attributes is denoted Ω'. A key for the PRP Π is generated and Ω' is then permuted using Π to generate the set $\tilde{\Omega}$. The key generation algorithm, $\mathsf{ABE.KeyGen}$, is then run on input $\tilde{\Omega}$ to produce the public and secret key pair for CPABE, $(k_{\mathsf{ABE}}, PP_{\mathsf{CPABE}})$. The secret

```
$(K_O, K_S, PP) \leftarrow_\$ \mathsf{KeyGen}(1^\kappa, \Omega, S)$

1 :  $(k_{\mathsf{BE}}, PP_{\mathsf{BE}}) \leftarrow_\$ \mathsf{BE.KeyGen}(1^\kappa, |\mathcal{U}|)$
2 :  $k_{\mathsf{SKE}} \leftarrow_\$ \mathsf{SKE.KeyGen}(1^\kappa)$
3 :  for $i \in \Omega$
4 :    $\omega_i \leftarrow_\$ \mathsf{SKE.Encrypt}(i, k_{\mathsf{SKE}})$
5 :  endfor
6 :  $\Omega' \leftarrow \{\omega_i\}_{i \in [1, |\Omega|]}$
7 :  $k_\Pi \leftarrow_\$ \{0,1\}^\kappa$
8 :  $\tilde{\Omega} \leftarrow \Pi_{k_\Pi}(\Omega')$
9 :  $(k_{\mathsf{ABE}}, PP_{\mathsf{ABE}}) \leftarrow_\$ \mathsf{ABE.KeyGen}(1^\kappa, \tilde{\Omega})$
10 : $st_O \leftarrow_\$ \{0,1\}^\kappa$
11 : $k_S \leftarrow_\$ \mathsf{BE.Add}(k_{\mathsf{BE}}, S)$
12 : $\mathcal{G} \leftarrow \{S\}$
13 : $st_S \leftarrow_\$ \mathsf{BE.Encrypt}(st_O, \mathcal{G}, k_{\mathsf{BE}})$
14 : return
15 : $K_S \leftarrow (k_S, st_S)$
16 : $K_O \leftarrow (k_{\mathsf{ABE}}, k_{\mathsf{BE}}, k_{\mathsf{SKE}}, \Pi, k_\Pi, st_O, \mathcal{G})$
17 : $PP \leftarrow (PP_{\mathsf{ABE}}, PP_{\mathsf{BE}}, \tilde{\Omega})$
```

Algorithm 5.1: KeyGen algorithm for eVSE

key for CPABE along with the PRP Π and its key form the next parts of the data owner's secret key. The data owner's state is generated by choosing a $\kappa-$bit binary string string uniformly at random, st_O. The server's secret key is then generated by running BE.Add using the server's identity as input. The authorised user group $\mathcal{G}$ is initialised as a set containing the server's identity. Finally the server state is generated by encrypting the data owner's state, st_O using BE.Encrypt. The two states, st_O and st_S are used for adding and revoking user's querying rights. The authorised user group along with st_O form the final parts of the data owner's secret key which is output as $K_O = (k_{\mathsf{CPABE}}, k_{\mathsf{BE}}, k_{\mathsf{SKE}}, \Pi, st_O, \mathcal{G})$. The server's secret key is made up of the BE user key generated using the server's identity, along with the server state: $K_S = (k_S, st_S)$. The public parameters are: $PP = (PP_{\mathsf{ABE}}, PP_{\mathsf{BE}}, \tilde{\Omega})$.

```
$\mathcal{I}_\mathcal{D} \leftarrow_\$ \mathsf{Buildindex}(\delta(\mathcal{D}), K_O, PP)$

1 :  $\mathcal{ID} \leftarrow_\$ \mathsf{ABE.KeyGen}(\delta(\mathcal{D}), k_{\mathsf{ABE}}, PP_{\mathsf{ABE}})$
```

Algorithm 5.2: Buildindex algorithm for eVSE

The BuildIndex algorithm generates the index for the encrypted data The index, $\mathcal{I}_\mathcal{D}$ is generated using the ABE.KeyGen algorithm with the preindex $\delta(\mathcal{D})$ as input along with the public and secret key pair for CPABE. The $\delta(\mathcal{D})$ corresponds to the attribute set that is input into into ABE.KeyGen algorithm in the standard definition (see Definition 2.3.5). The output of the ABE.KeyGen algorithm is the CPABE secret key corresponding to the set of attributes in $\delta(\mathcal{D})$ and is referred to as the index $\mathcal{I}_{\mathcal{D}}$.

```
$(K_u, K_O) \leftarrow_{\$} \mathsf{AddUser}(u, K_O, PP)$
1 :  $\mathcal{G} \leftarrow \mathcal{G} \cup \{u\}$
2 :  $k_u \leftarrow \mathsf{BE.Add}(u, k_{\mathsf{BE}})$
3 :  $st'_O \leftarrow_{\$} \{0,1\}^{\kappa}$
4 :  $st_O \leftarrow st'_O$
5 :  $st'_S \leftarrow_{\$} \mathsf{BE.Encrypt}(st_O, \mathcal{G}, k_{\mathsf{BE}})$
6 :  $st_S \leftarrow st'_S$
7 :  Update server key with current value of $st_S$
8 :  return
9 :  $K_u \leftarrow (k_u, k_{\mathsf{SKE}}, \Pi)$
10 : $K_O \leftarrow (k_{\mathsf{ABE}}, k_{\mathsf{BE}}, k_{\mathsf{SKE}}, \Pi, st_O, \mathcal{G})$
```

Algorithm 5.3: AddUser algorithm for eVSE

The AddUser algorithm authorizes users new users to allow them to query the encrypted data. The new user's identity u is added to the authorised user group $\mathcal{G}$ and a user key is generated using BE.Add using the new user's identity as input. A new data owner state is then generated and the old value of the data owner's state is updated. The new server state is created by encrypting the new data owner state using BE.Encrypt with the updated authorised user group as input. The server's key is then updated so it holds the new value for st_S. The new user's secret key is output as: $K_u = (k_u, k_{\mathsf{SKE}}, \Pi)$. The key k_{SKE} and the permutation Π are given to the user to allow them to read the encoded attributes and generate search queries.

The Query algorithm generates the search queries which users can send to the server. Initially the user retrieves the current server state, st_S from the server and decrypts it using their user key. Only users that were authorised to search on the encrypted data when the server state was created will be able to decrypt the server state. If the user was not a member of the authorised user group then decryption will fail and the algorithm will return $\perp$. If the user is authorised to search on the encrypted data, the query is formed by encrypting pairs of plaintexts using CPABE, with the search query and the complement of the search query as input respectively. A bit b is chosen prior to encryption and used to permute the plaintexts.The query token is formed by the ciphertext pairs concatenated and then used as input to the permutation ϕ, using st_O as the key. The verification token is created by applying the one-way function g to each plaintext and the retrieval token is the bit b.

The Search algorithm evaluates the search query over the encrypted data in the

```
$(QT_Q, VK_Q, RK_Q) \leftarrow \mathsf{Query}(Q = \{Q_i\}, K_u, PP)$
1 :  Retrieve $st_S$ from server
2 :  $st'_O \leftarrow \mathsf{BE.Decrypt}(st_S, k_u)$
3 :  if $st'_O = \perp$ then
4 :    return $\perp$
5 :  else
6 :    $st_O \leftarrow st'_O$
7 :    for $i = 1, ..., |Q|$ do
8 :      $(m_{0_i}, m_{1_i}) \xleftarrow{\$} \mathcal{M} \times \mathcal{M}$
9 :      $b_i \xleftarrow{\$} \{0, 1\}$
10 :     $c_{b_i} \leftarrow \mathsf{ABE.Encrypt}(m_{b_i}, Q_i, PP_{\mathsf{ABE}})$
11 :     $c_{1-b_i} \leftarrow \mathsf{ABE.Encrypt}(m_{1-b_i}, \overline{Q_i}, PP_{\mathsf{ABE}})$
12 :     $QT_{Q_i} \leftarrow (c_{b_i}, c_{1-b_i})$
13 :     $\gamma_i \leftarrow \phi_{st_O}(c_{b_i} \| c_{1-b_i})$
14 :     $VK_{Q_i} \leftarrow (g(m_{0_i}), g(m_{1_i}))$
15 :     $RK_{Q_i} \leftarrow b_i$
16 :   endfor
17 :  $QT_Q \leftarrow \{\gamma_i\}, VK_Q \leftarrow \{VKQ_i\}, RK_Q \leftarrow \{RK_{Q_i}\}$
```

Algorithm 5.4: Query algorithm for eVSE

```
$R \leftarrow \mathsf{Search}(\mathcal{I_D}, QT_Q = \{\gamma_i\}, K_S, PP)$
1 :  $st'_O \leftarrow \mathsf{BE.Decrypt}(st_S, k_S)$
2 :  for $i = 1, ..., |Q|$ do
3 :    $(c_{b_i} \| c_{1-b_i}) \leftarrow \phi^{-1}_{st'_O}(\gamma_i)$
4 :    $d_{b_i} \leftarrow \mathsf{ABE.Decrypt}(c_{b_i}, \mathcal{I_D}, PP_{\mathsf{ABE}})$
5 :    $d_{1-b_i} \leftarrow \mathsf{ABE.Decrypt}(c_{1-b_i}, \mathcal{I_D}, PP_{\mathsf{ABE}})$
6 :    $R_i = (d_{b_i}, d_{1-b_i})$
7 :  endfor
8 :  $R = \{R_i\}$
```

Algorithm 5.5: Search algorithm for eVSE

index $\mathcal{I_D}$. It outputs a result R which can then be verified by the user. The server initially decrypts the current value of the server state and uses this as the key for the inversion of the PRP ϕ to reveal the pairs of CPABE ciphertexts that make up the query token. The index is then used to decrypt these pairs of ciphertexts. As one of each pair was encrypted using the search query and the other using the complement of the query exactly one of each pair will decrypt successfully to the original plaintext. These pairs of decryptions form the search results R.

```
$RT_Q \leftarrow \mathsf{Verify}(R = \{(d_i, d'_i)\}, VK_Q = \{(VK_i, VK'_i)\}, PP)$
1 :  for $i = 1, ..., |Q|$ do
2 :    if $VK_i = g(d_i)$ then
3 :      $RT_{Q_i} = d_i$
4 :    elseif $VK'_i = g(d'_i)$ then
5 :      $RT_{Q_i} = d'_i$
6 :    else $RT_{Q_i} = \perp$
7 :  endfor
8 :  $RT_Q = \{RT_{Q_i}\}$
```

Algorithm 5.6: Verify algorithm for eVSE

The Verify algorithm checks the correctness and completeness of the search result, R, using the verification key, VK_Q. The one-way function g is applied to the results which are then compared to the relevant parts in the verification key VK_Q. One of these pairs from R should match the verification key, this part is added to the retrieval token RT_Q. If neither match then this tells the verifier that the search has not been performed honestly and the relevant part of the retrieval token is set to $\perp$.

```
$R \leftarrow \mathsf{Retrieve}(VK_Q = \{(g(m_{b_i}), g(m_{1-b_i}))\}, RT_Q = \{RT_{Q_i}\}, RK_Q = \{b_i\}, PP)$
1 :  for $i = 1, ..., |Q|$ do
2 :    if $g(RT_{Q_i}) = g(m_0)$ then
3 :      $r_i = 1$
4 :    elseif $g(RT_{Q_i}) = g(m_1)$ then
5 :      $r_i = 0$
6 :    else $r_i = \perp$
7 :  endfor
8 :  $r = \{r_i\}$
```

Algorithm 5.7: Verify algorithm for eVSE

The Retrieve algorithm reveals the final search results using the retrieval key RK_Q. The retrieval key enables the user to determine whether the ciphertext encrypted with the search query or the ciphertext encrypted with the complement of the search query decrypted correctly and hence, for each data item the user can tell whether it satisfies the search query or not. The final search results are output as a bit string of length n, with an entry corresponding to each data item. An entry of 1 implies that the data item satisfied the search query and an entry of 0 tells the user that the data item did not satisfy the search query.

The RevokeUser algorithm removes a user's capability to produce search queries,

```
$K_O \leftarrow \mathsf{RevokeUser}(u, K_O, PP)$
$\mathcal{G} \leftarrow \mathcal{G} \setminus \{u\}$
$st'_O \leftarrow_{\$} \{0,1\}^{\kappa}$
$st_O \leftarrow st'_O$
$st'_S \leftarrow_{\$} \mathsf{BE.Encrypt}(st_O, \mathcal{G}, k_{\mathsf{BE}})$
$st_S \leftarrow st'_S$
Update server key with current value of $st_S$
return
$K_O \leftarrow (k_{\mathsf{ABE}}, k_{\mathsf{BE}}, k_{\mathsf{SKE}}, \Pi, st_O, \mathcal{G})$
```

Algorithm 5.8: RevokeUser algorithm for eVSE

and hence search the encrypted data. The revoked user's identity is first removed from the authorised user set $\mathcal{G}$. A new data owner state is then created and encrypted using the BE scheme with the updated authorised user set as input. The resulting BE ciphertext is the updated server state, st_S. The server key is updated with this new value for st_S and the data owner's key is updated with the new value of the data owner state, st_O.

Theorem 5.3.1. *Given a selective* IND-CPA *secure CP-ABE scheme, a symmetric encryption scheme and a broadcast encryption scheme, both secure in the sense of* IND-CPA*, pseudo-random permutations Π and ϕ, and a one-way function g, let* eVSE *be the extended verifiable searchable encryption scheme defined in Algorithms 5.1–5.8. Then* eVSE *is secure in the sense of Public Verifiability, Index Privacy and Query Privacy.*

Note that we can add additional contextual access control following Alderman et al. [5] by replacing ϕ with a *key assignment scheme.*

5.4 Proofs of Security

5.4.1 Public verifiability

Lemma 5.4.1. *An eVSE scheme* eVSE *as defined in Algorithms 5.1–5.8 is secure against Public Verifiability (Game 5.1) under the same assumptions as in Theorem 5.3.1.*

Proof. Suppose $\mathcal{A}_{\mathsf{eVSE}}$ is an adversary with non-negligible advantage against the selective Public Verifiability game (Game 5.1) when instantiated with Algorithms 5.1–5.8. We begin by defining the following three games:

- **Game 0.** This is the selective Public Verifiability game as defined in Game 5.1.

- **Game 1.** This is the same as **Game 0** with the modification that in Query, we no longer return an encryption of m_0 and m_1.
  Instead, we choose another random plaintext $m' \neq m_0, m_1$ and, if $Q(\mathcal{I}_{\mathcal{D}*}) = 1$, we replace c_1 by $\mathsf{ABE.Encrypt}(\overline{Q}, m', PP_{\mathsf{ABE}})$. Otherwise, we replace c_0 by $\mathsf{ABE.Encrypt}(Q, m', PP_{\mathsf{ABE}})$. In other words, we replace the ciphertext associated with the unsatisfied function with the encryption of a separate random plaintext unrelated to the other system parameters, and in particular to the verification keys.

- **Game 2.** This is the same as **Game 1** with the exception that instead of choosing a random plaintext m', we implicitly set m' to be the challenge input w in the one-way function game.

We show that an adversary with non-negligible advantage against the selective Public Verifiability game can be used to construct an adversary that may invert the one-way function g.

Game 0 to Game 1. We begin by showing that there is a negligible distinguishing advantage between **Game 0** and **Game 1**.

Suppose otherwise, that $\mathcal{A}_{\mathsf{eVSE}}$ can distinguish the two games with non-negligible advantage δ. We then construct an adversary $\mathcal{A}_{\mathsf{ABE}}$ that uses $\mathcal{A}_{\mathsf{eVSE}}$ as a sub-routine to break the selective IND-CPA security of the CP-ABE scheme. We consider a challenger $\mathcal{C}$ playing the IND-CPA game with $\mathcal{A}_{\mathsf{ABE}}$, who in turn acts as a challenger in the public verifiability game (Game 5.1) for $\mathcal{A}_{\mathsf{eVSE}}$:

1. $\mathcal{A}_{\mathsf{eVSE}}$ is given the security parameter and Ω by the environment, and declares (to $\mathcal{A}_{\mathsf{ABE}}$) its choice of data set $D^{\star}$ and the query Q^{*}.

2. $\mathcal{A}_{\mathsf{ABE}}$ uses oracle calls to $\mathcal{C}$ in order to obtain the encrypted-then-permuted universe $\tilde{\Omega}$. $\mathcal{C}$ runs $k_{\mathsf{SKE}} \leftarrow \mathsf{SKE.KeyGen}(1^{\kappa})$ and uses the symmetric key k to symmetrically encrypt each element of Ω and therefore obtain Ω'. $\mathcal{C}$ then applies the permutation Π to the set Ω' that results in the permuted set $\tilde{\Omega}$. Furthermore $\mathcal{C}$ runs $k_{\mathsf{BE}} \leftarrow \mathsf{BE.KeyGen}(1^{\kappa})$ from the broadcast encryption scheme to produce $(k_{\mathsf{BE}}, PP_{\mathsf{BE}})$ and $\mathsf{ABE.KeyGen}(1^{\kappa}, \tilde{\Omega})$ from the ABE scheme to produce $(k_{\mathsf{ABE}}, PP_{\mathsf{ABE}})$. It then chooses an authorised user group $\mathcal{G} \in \mathcal{U}$, a data owner state $st_O \xleftarrow{\$} \{0,1\}^{\kappa}$ and encrypts it to produce st_S. $\mathcal{C}$ also generates components of the server key by running $\mathsf{BE.Add}$ using the server identity S as input.

3. $\mathcal{C}$ sets the public parameters to be $PP = (PP_{\mathsf{ABE}}, PP_{\mathsf{BE}}, \tilde{\Omega})$ and gives them to $\mathcal{A}_{\mathsf{ABE}}$, who shares them with $\mathcal{A}_{\mathsf{eVSE}}$.

4. $\mathcal{C}$ sets the master key to be $K_O = (k_{\mathsf{ABE}}, k_{\mathsf{BE}}, k_{\mathsf{SKE}}, \Pi, st_O, \mathcal{G})$ and keeps it private, except for $k_{\mathsf{BE}}, st_O, \Pi, \mathcal{G}$ that it shares with $\mathcal{A}_{\mathsf{ABE}}$.

5. The server key is set as $K_S = (k_s, st_S)$ and shared with $\mathcal{A}_{\mathsf{ABE}}$.

6. $\mathcal{A}_{\mathsf{ABE}}$ chooses a random user u authorised user group $\mathcal{G}$ which is then used as input for $\mathsf{AddUser}$, which generates the secret key for user u. $\mathcal{A}_{\mathsf{ABE}}$ possesses all parameters to run this algorithm. It sends the updated values of st_O and $\mathcal{G}$ to $\mathcal{C}$.

7. $\mathcal{A}_{\mathsf{ABE}}$ proceeds to create the index. It does so by sending the challenge data set $\mathcal{D}^*$ to $\mathcal{C}$ who then runs Encode to produce the pre-index $\delta(\mathcal{D}^*)$. It then runs $\mathsf{BuildIndex}$ using the pre-index as input to produce $\mathcal{I}_{\mathcal{D}^*}$.

8. $\mathcal{A}_{\mathsf{ABE}}$ must send a challenge access structure to the challenger. It first computes $r* = Q^*(\mathcal{I}_{\mathcal{D}^\star})$, that is, the outcome of the challenge query $Q*$ applied to the challenge index. If $r* = 1$, $\mathcal{A}_{\mathsf{ABE}}$ sets $\mathbb{A}^\star = \overline{F_{Q^\star}}$, where $F_{Q^\star}$ is the query $Q*$ represented as a function. Else, $r = 0$ and $\mathbb{A}^\star = F_{Q^\star}$. The challenge access structure, $\mathbb{A}^*$ is sent to $\mathcal{C}$.

9. $\mathcal{A}_{\mathsf{ABE}}$ sends st_S to $\mathcal{C}$ who runs $\widetilde{j} \leftarrow \mathsf{BE.Decrypt}(\mathrm{st_S}, \mathrm{uk_u})$ and returns $\widetilde{j}$ to $\mathcal{A}_{\mathsf{ABE}}$ in the Query stage. It checks whether $\widetilde{j} = j$. If so it proceeds with the next step, otherwise the game aborts.

10. Each query Q can comprise multiple subqueries Q_i. Therefore the following is done for all $i \in [|Q|] = \{1, \dots, |Q|\}$:

    - To generate the challenge input, $\mathcal{A}_{\mathsf{ABE}}$ begins by choosing a random bit b_i, three random plaintexts m_{0_i}, m_{1_i} and m'_i from the plaintext space, and another random bit t_i.

      $\mathcal{A}_{\mathsf{ABE}}$ sends the plaintexts m_{0_i} and m_{1_i} to $\mathcal{C}$ as the challenge plaintexts for the CP-ABE game. $\mathcal{C}$ chooses a random bit c_i and returns $CT^\star \leftarrow \mathsf{Encrypt}(m_{c_i}, \mathbb{A}^\star, PP_{\mathsf{ABE}})$.

      – If $r = 1$, $\mathcal{A}_{\mathsf{ABE}}$ generates $c_{b_i} \leftarrow \mathsf{Encrypt}(Q_i^*, m'_i, PP_{\mathsf{ABE}})$ and sets $c_{1-b_i} = CT^\star$ (formed over $\mathbb{A}^\star$ by $\mathcal{C}$). It also sets $VK_{b_i} = g(m'_i)$ and $VK_{1-b_i} = g(m_{t_i})$.

      – Else $r = 0$, and $\mathcal{A}_{\mathsf{ABE}}$ sets $c_{b_i} = CT^\star$ and computes $c_{1-b_i} \leftarrow \mathsf{Encrypt}(\overline{Q_i^*}, m'_i, PP_{\mathsf{ABE}})$. It sets $VK_{b_i} = g(m_{t_i})$ and $VK_{1-b_i} = g(m'_i)$.

      $\mathcal{A}_{\mathsf{ABE}}$ sets $QT_{Q_i^*} = (c_{b_i}, c_{1-b_i})$, $VK_{Q_i^*} = (VK_{b_i}, VK_{1-b_i})$ and $RK_{Q_i^*} = b_i$. Finally, $\mathcal{A}_{\mathsf{ABE}}$ computes $\gamma_i \leftarrow \phi_{st_O}(c_{b_i} \| c_{1-b_i})$ and sets for all $i \in [|Q^*|]$ $QT_Q^* = \{\gamma_i\}$, $VK_Q = \{VK_{Q_i^*}\}$, and $RK_Q^* = \{RK_{Q_i^*}\}$.

11. $\mathcal{A}_{\mathsf{ABE}}$ sends the output from Query along with the public information to $\mathcal{A}_{\mathsf{eVSE}}$, who is also given oracle access to which $\mathcal{A}_{\mathsf{ABE}}$ responds as follows:

    - $\mathsf{BuildIndex}(\cdot, K_O, PP)$: To generate the evaluation key for the queried pre-index $\delta(\mathcal{D})$, $\mathcal{A}_{\mathsf{ABE}}$ makes use of the KeyGen oracle in the CP-ABE game. $\mathcal{A}_{\mathsf{ABE}}$ then makes an oracle query to $\mathcal{C}$ for $\mathcal{O}^{\mathsf{KeyGen}}(\delta(\mathcal{D}), k_{\mathsf{ABE}}, PP_{\mathsf{ABE}})$. $\mathcal{C}$ shall generate a CP-ABE decryption key for $\delta(\mathcal{D})$ if and only if $\delta(\mathcal{D}) \notin \mathbb{A}^\star$.
    - All other oracles are run as described in Section 5.2.3.

12. Eventually, $\mathcal{A}_{\mathsf{eVSE}}$ outputs $R^\star$ which it believes is a valid forgery (i.e. that it will be accepted yet does not correspond to the correct value of $Q(\mathcal{I}_{\mathcal{D}^\star})$).

13. $\mathcal{A}_{\mathsf{ABE}}$ parses $R^\star$ as $\{R_i^\star\}$ for all $i \in [|Q^*|]$. For each $i \in [|Q^*|]$ $\mathcal{A}_{\mathsf{ABE}}$ does the following: $\mathcal{A}_{\mathsf{ABE}}$ parses $R_i^\star$ as (d_{b_i}, d_{1-b_i}) and using the retrieval key $RK_{Q_i} = b_i$, finds d_{0_i} and d_{1_i}. One of d_{0_i} and d_{1_i} will be $\perp$ (by construction) and we denote the other value by Y_i.

    Observe that, since $\mathcal{A}_{\mathsf{eVSE}}$ is assumed to be a successful adversary against selective public verifiability, the non-$\perp$ value, Y_i, that it will return will be the plaintext m_{c_i} since the challenge access structure was always set to be unsatisfied on the challenge input.

    Thus, if $g(Y_i) = g(m_{t_i})$, $\mathcal{A}_{\mathsf{ABE}}$ outputs a guess $c_i' = t_i$ and otherwise guesses $c_i' = (1 - t_i)$.

    If $t_i = c_i$ (the challenge bit chosen by $\mathcal{C}$), we observe that the above corresponds to **Game 0** (since the verification key comprises $g(m_i')$ where m_i' is the plaintext a legitimate server could recover, and $g(m_{c_i})$ where m_{c_i} is the other plaintext). Alternatively, $t_i = 1 - c_i$ and the distribution of the above experiment is identical to **Game 1** (since the verification key comprises the legitimate plaintext and a random plaintext m_{1-c_i} that is unrelated to the ciphertext).

    Now, we consider the advantage of this constructed adversary $\mathcal{A}_{\mathsf{ABE}}$ playing the selective IND-CPA game for CP-ABE. Recall that by assumption, $\mathcal{A}_{\mathsf{eVSE}}$ has a non-negligible advantage δ in distinguishing between **Game 0** and **Game 1**, that is:

$$|\mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 0}}[e\mathcal{VSE}, X]) - \mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 1}}[e\mathcal{VSE}, X])| \geqslant \delta,$$

    The probability of $\mathcal{A}_{\mathsf{ABE}}$ guessing c_i correctly is:

$$\begin{aligned}
\mathbb{P}(c_i' = c_i) &= \mathbb{P}(t_i = c_i)\mathbb{P}(c_i' = c_i | t_i = c_i) + \mathbb{P}(t_i \neq c_i)\mathbb{P}(c_i' = c_i | t_i \neq c_i) \\
&= \frac{1}{2}\mathbb{P}(g(Y_i) = g(m_{t_i}) | t_i = c_i) + \frac{1}{2}\mathbb{P}(g(Y_i) \neq g(m_{t_i}) | t_i \neq c_i) \\
&= \frac{1}{2}\mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 0}}[e\mathcal{VSE}, X]) + \frac{1}{2}\left(1 - \mathbb{P}(g(Y_i) = g(m_{t_i}) | t_i \neq c_i)\right) \\
&= \frac{1}{2}\mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 0}}[e\mathcal{VSE}, X]) + \frac{1}{2}\left(1 - \mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 1}}[e\mathcal{VSE}, X])\right) \\
&= \frac{1}{2}\left(\mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 0}}[e\mathcal{VSE}, X]) - \mathbb{P}(1 \leftarrow \mathbf{Exp}_{\mathcal{A}_{\mathsf{eVSE}}}^{\mathbf{Game\ 1}}[e\mathcal{VSE}, X]) + 1\right) \\
&\geqslant \frac{1}{2}(\delta + 1)
\end{aligned}$$

Hence,

$$\begin{aligned}
Adv_{\mathcal{A}_{\mathsf{ABE}}} &\geqslant \left|\mathbb{P}(c_i = c_i') - \frac{1}{2}\right| \\
&\geqslant \left|\frac{1}{2}(\delta + 1) - \frac{1}{2}\right| \\
&= \frac{\delta}{2}
\end{aligned}$$

14. Overall the above is done for all $i \in [|Q^*|]$ and therefore we have $n \cdot Adv_{\mathcal{A}_{\mathsf{ABE}}} \geqslant n \cdot \frac{\delta}{2}$.

Hence, if $\mathcal{A}_{\mathsf{eVSE}}$ has advantage δ at distinguishing these games then $\mathcal{A}_{\mathsf{ABE}}$ can win the selective IND-CPA game for CP-ABE with non-negligible probability. Thus since we assumed the CP-ABE scheme to be secure, we conclude that $\mathcal{A}_{\mathsf{eVSE}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability.

Game 1 to Game 2. The transition from **Game 1** to **Game 2** is simply to set the value of m_i' to no longer be random but instead to correspond to the challenge w in the one-way function inversion game. The game basically formalizes that it is infeasible for any probabilistic polynomial-time algorithm to invert the one-way function g, i.e. to find a pre-image of a given value z. We argue that the adversary has no distinguishing advantage between these games since the new value is independent of anything else in the system bar the verification key $g(w)$ and hence looks random to an adversary with no additional information (in particular, $\mathcal{A}_{\mathsf{eVSE}}$ does not see the challenge for the one-way function as this is played between $\mathcal{C}$ and $\mathcal{A}_{\mathsf{ABE}}$).

Final Proof. We now show that using $\mathcal{A}_{\mathsf{eVSE}}$ in **Game 2**, $\mathcal{A}_{\mathsf{ABE}}$ can invert the one-way function g. That is, given a challenge $z = g(w)$ we can recover w. (The following can be done analogous for all $i \in [|Q^*|]$.) Specifically wlog, during Query, we choose the plaintexts as follows:

- if $Q^*(\mathcal{I}_{\mathcal{D}^*}) = 1$, we implicitly set m_{1-b_i} to be w and set the verification key component $VK_{1-b_i} = z$. We choose m_{b_i} and VK_{b_i} randomly as usual.
- if $Q^*(\mathcal{I}_{\mathcal{D}^*}) = 0$, we implicitly set m_{b_i} to be w and set the verification key component $VK_{b_i} = z$. We choose m_{1-b_i} and VK_{1-b_i} randomly as usual.

Now, since $\mathcal{A}_{\mathsf{eVSE}}$ is assumed to be successful, it will output a forgery comprising the plaintext encrypted under the unsatisfied query (Q^* or $\overline{Q^*}$). By construction, this will be w (and the adversary's view is consistent since the verification key is simulated correctly using z). $\mathcal{A}_{\mathsf{ABE}}$ can therefore forward this result to $\mathcal{C}$ in order to invert the one-way function with the same non-negligible probability that $\mathcal{A}_{\mathsf{eVSE}}$ has against the public verifiability game.

We conclude that if the CPABE scheme is selectively IND-CPA secure and the one-way function is hard-to-invert, then eVSE as defined by Algorithms 5.1–5.8 is secure in the sense of selective Public Verifiability.

□

5.4.2 Index privacy

Lemma 5.4.2. eVSE *as defined in Algorithms 5.1–5.8 is secure against Index Privacy (Game 5.2) under the same assumptions as in Theorem 5.3.1.*

Proof. Suppose $\mathcal{A}_{\mathsf{eVSE}}$ is an adversary with non-negligible advantage against the Index Privacy game (Game 5.2) when instantiated with Algorithms 5.1–5.8. We begin by defining the following two games:

- **Game 0**: Fix an adversary $\mathcal{A}_{\mathsf{eVSE}}$ we define **Game 0** to be the selective Index Privacy game as defined in Game 5.2.
- **Game 1**: This is the same as **Game 0** with the modification that we use a random permutation in Algorithm 5.1 to construct $\tilde{\Omega}$. We re-label $\tilde{\Omega}$ as $\tilde{\Omega}'$ to differentiate it from the $\tilde{\Omega}$ generated using a PRP in **Game 0**. This means that in **Game 1**, PP will contain a randomly permuted set $\tilde{\Omega}'$ instead of one generated using a PRP, as in **Game 0**.

We show that an adversary with non-negligible advantage against the selective Index Privacy game can be used to construct an adversary $\mathcal{A}_{\mathsf{SKE}}$ which may break the IND-CPA security (Definition 2.6.3) of a symmetric key encryption scheme SKE = (SKE.KeyGen, SKE.Encrypt, SKE.Decrypt).

Game 0 to Game 1. The PRP-assumption [22] is formulated as follows. Let $\mathcal{D}$ be a distinguisher algorithm that takes as input a permutation (which is either pseudorandom or truly random) and outputs a bit, b. In the following, let Π be a random permutation, Π' be a PRP and S be a set. We define the PRP-advantage of $\mathcal{D}$ to be:

$$\left|\mathbb{P}[\tilde{S} \xleftarrow{\$} \Pi(S) : \mathcal{D}(\tilde{S}) = 1] - \mathbb{P}[\tilde{S} \leftarrow \Pi'(S) : \mathcal{D}(\tilde{S}) = 1]\right|.$$

The PRP-assumption states that for any efficient algorithm $\mathcal{D}$, the PRP-advantage is negligible.

- Fix an adversary $\mathcal{A}_{\mathsf{eVSE}}$ that is able to distinguish between **Game 0** and **Game 1** with non-negligible advantage δ. That is, $\mathcal{A}_{\mathsf{eVSE}}$'s probability of success in one game is non-negligibly different to their probability of success in the other game (suppose the probability of success in **Game 0** is higher than that of **Game 1**, without loss of generality). We can build a distinguisher $\mathcal{D}$ that is able to distinguish whether a permutation is either truly random or pseudorandom with non-negligible probability, by using $\mathcal{A}_{\mathsf{eVSE}}$ as a subroutine (hence has non-negligible PRP-advantage), leading to a contradiction.

- Given a permutation π, $\mathcal{D}$ instantiates $\mathcal{A}_{\mathsf{eVSE}}$ in Game 5.2 using π to permute the universe of attributes instead of Π. Note that if π is a PRP then $\mathcal{A}_{\mathsf{eVSE}}$ is acting in **Game 0** and if π is a truly random permutation then $\mathcal{A}_{\mathsf{eVSE}}$ is acting in **Game 1**.

- $\mathcal{A}_{\mathsf{eVSE}}$ outputs a bit b' at the end of their game.

- If $\mathcal{A}_{\mathsf{eVSE}}$ wins their game then $\mathcal{D}$ outputs 1, indicating that they believe π is a PRP and 0 otherwise.

- $\mathcal{D}$ will be able to distinguish π with exactly $\mathcal{A}_{\mathsf{eVSE}}$'s (non-negligible) advantage, δ contradicting the PRP-assumption. Thus we conclude that $\mathcal{A}_{\mathsf{eVSE}}$ cannot distinguish **Game 0** from **Game 1** with non-negligible probability. Hence we continue the proof using **Game 1** and assume that $\mathcal{A}_{\mathsf{eVSE}}$ can only gain at most an extra negligible advantage γ from hopping from **Game 0 to Game 1.**

Reduction to IND-CPA. Now let $\mathcal{A}_{\mathsf{eVSE}}$ be an adversary with non-negligible advantage δ against **Game 1**. We now show that using $\mathcal{A}_{\mathsf{eVSE}}$ as a subroutine in **Game 1**, $\mathcal{A}_{\mathsf{SKE}}$ is able to break the IND-CPA security of SKE. That is, given a challenge ciphertext c which is an encryption m_b where $b \xleftarrow{\$} \{0,1\}$, $\mathcal{A}_{\mathsf{SKE}}$ can distinguish whether c is an encryption of m_0 or m_1.

Fix an adversary $\mathcal{A}_{\mathsf{eVSE}}$ that is able to break the index privacy of eVSE with non-negligible advantage γ. Let $\mathcal{C}$ be the challenger for$\mathcal{A}_{\mathsf{SKE}}$ in the proof, $\mathcal{A}_{\mathsf{SKE}}$ will act as the challenger for $\mathcal{A}_{\mathsf{eVSE}}$.

1. $\mathcal{A}_{\mathsf{eVSE}}$ chooses their challenge sets: $\mathcal{D}_0 = (d_{0,1}, d_{0,2}, ..., d_{0,q})$ and $\mathcal{D}_1 = (d_{1,1}, d_{1,2}, ..., d_{1,q}) \subseteq \mathcal{U}$ such that $|\mathcal{D}_0| = |\mathcal{D}_1|$. Note that a rational adversary will always choose $\mathcal{D}_0 \neq \mathcal{D}_1$ i.e. $\mathcal{D}_0$ and $\mathcal{D}_1$ differ by at least one element. $\mathcal{A}_{\mathsf{eVSE}}$ submits these challenge sets to $\mathcal{A}_{\mathsf{SKE}}$.

2. The challenger $\mathcal{C}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$.

3. KeyGen is run between $\mathcal{C}$ and $\mathcal{A}_{\mathsf{SKE}}$:

   - $\mathcal{A}_{\mathsf{SKE}}$ generates the secret key for the broadcast encryption scheme $\mathsf{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Encrypt}, \mathsf{BE.Add}, \mathsf{BE.Decrypt})$:

     $$k_{\mathsf{BE}} \leftarrow \mathsf{BE.KeyGen}(1^\kappa, |\mathcal{U}|),$$

     which is retained by $\mathcal{A}_{\mathsf{SKE}}$ and shared with $\mathcal{C}$.

   - $\mathcal{C}$ generates the secret key for the symmetric encryption scheme $\mathsf{SKE} = (\mathsf{SKE.KeyGen}, \mathsf{SKE.Encrypt}, \mathsf{SKE.Decrypt})$:

     $$k_{\mathsf{SKE}} \leftarrow \mathsf{SKE.KeyGen}(1^\kappa),$$

     which is retained by $\mathcal{C}$.

   - $\mathcal{C}$ provides $\mathcal{A}_{\mathsf{SKE}}$ with access to the following oracles $\text{SKE.E}\textsc{ncrypt}(\cdot, k)$: which takes as input a plaintext m and returns its symmetric encryption under k_{SKE} and $\text{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ which takes as input two plaintexts m_0, m_1 and a bit b and outputs the symmetric encryption of m_b under k_{SKE}.

   - For all $d_{0,j} \neq d_{1,j}$, $\mathcal{A}_{\mathsf{SKE}}$ submits the pair $(d_{0,j}, d_{1,j})$ (where such pairs exist as the challenge sets are not equal) to $\text{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ and receives challenge ciphertexts, c_j, in return.

   - These challenge ciphertexts are included in Ω'. To compute the rest of Ω' $\mathcal{A}_{\mathsf{SKE}}$ submits every other pair $(d_{0,i}, d_{1,i}) \in \{\mathcal{D}_0, \mathcal{D}_1\} : d_{0,i} = d_{1,i}$, to $\text{SKE.E}\textsc{ncrypt}(\cdot, k_{\mathsf{SKE}})$ and includes the output in Ω'. To ensure Ω' contains an encryption of each attribute in Ω, $\mathcal{A}_{\mathsf{SKE}}$ submits the remaining attributes of Ω: $a_n \notin (\mathcal{D}_0 \cup \mathcal{D}_1)$ to $\text{SKE.E}\textsc{ncrypt}(\cdot, k_{\mathsf{SKE}})$ and includes the output in Ω'. Ω' now contains an encryption of every element in Ω. To map elements from Ω to Ω' (which is required to run Encode), every attribute in Ω needs to have a corresponding ciphertext in Ω' which is defined as follows: for each pair of attributes that were submitted to $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ we let the output denote the corresponding element in Ω' for the attribute on the left hand side of the submitted pair. For all attributes in Ω submitted to SKE.ENCRYPT$(\cdot, k_{\mathsf{SKE}})$ the output is the corresponding element in Ω' for that attribute. This defines a bijective mapping of attributes in Ω to elements in Ω' as required.

- $\mathcal{A}_{\mathsf{SKE}}$ defines the set $\tilde{\Omega} \leftarrow \Pi(\Omega')$, where Π is a random permutation and runs:
$$(k_{\mathsf{ABE}}, PP_{\mathsf{ABE}}) \leftarrow \mathsf{ABE.KeyGen}(1^\kappa, \tilde{\Omega}).$$
- $\mathcal{A}_{\mathsf{SKE}}$ chooses the data owner state uniformly at random: $st_O \xleftarrow{\$} \{0,1\}^\kappa$ and then generates the first part of the sever key:
$$k_{\mathsf{S}} \xleftarrow{\$} \mathsf{BE.Add}(k_{\mathsf{BE}}, S).$$
- The authorised user group $\mathcal{G}$ is initialised as the set containing the server identity and the server state, st_S is generated:
$$st_S \xleftarrow{\$} \mathsf{BE.Encrypt}(st_O, \mathcal{G}, k_{\mathsf{BE}}).$$
- The server key is set as $K_S = (k_s, st_S)$, and given to $\mathcal{A}_{\mathsf{eVSE}}$.
- $\mathcal{A}_{\mathsf{SKE}}$ retains k_{ABE}, k_{BE} and st_O and sets $PP = (PP_{\mathsf{ABE}}, PP_{\mathsf{BE}}, \tilde{\Omega})$ which is given to $\mathcal{A}_{\mathsf{eVSE}}$.

4. The challenge bit for $\mathcal{A}_{\mathsf{eVSE}}$, $\hat{b}$, is set to b.

5. $\mathcal{A}_{\mathsf{SKE}}$ creates a pre-index $\delta(\mathcal{D}_b)$ to encode into the challenge index for $\mathcal{A}_{\mathsf{eVSE}}$. This is done using Encode, which takes as input the elements from the challenge set $\mathcal{D}_b$ and maps them to elements in Ω'. Due to the way the mapping is defined using $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$, this will produce a preindex containing encryptions of elements from the challenge set $\mathcal{D}_b$. If $b = 0$ then $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ would have been used for encryption hence the attributes in $\mathcal{D}_0$ would have been encrypted and be included in the preindex, wheres if $b = 1$ then $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ would have been used for encryption hence the attributes in $\mathcal{D}_1$ would have been encrypted and be included in the preindex.

6. $\mathcal{A}_{\mathsf{SKE}}$ runs $\mathsf{BuildIndex}$ using the pre-index created in step 5 to create the challenge index $\mathcal{I}_{\mathcal{D}_b}$ for $\mathcal{A}_{\mathsf{eVSE}}$. Note that although $\mathsf{BuildIndex}$ takes K_O as input, it does

not require k_{SKE} or Π. The only part of K_O that is required is k_{ABE} which is generated and retained by $\mathcal{A}_{\mathsf{SKE}}$ in step 3.

7. $\mathcal{A}_{\mathsf{SKE}}$ choses a user from the userspace and authorises the user identity by running $\mathsf{AddUser}$. They then generate a series of queries that produce the same results when evaluated over the challenge index and return these queries to $\mathcal{A}_{\mathsf{eVSE}}$.

8. $\mathcal{A}_{\mathsf{eVSE}}$ is given oracle access to $\textsc{BuildIndex}(\cdot, K_O, PP)$, $\textsc{Search}(\cdot,\cdot, K_S, PP)$, $\textsc{Verify}(\cdot,\cdot, PP)$ and $\textsc{Retrieve}(\cdot,\cdot,\cdot, PP)$. To query $\textsc{BuildIndex}$ $\mathcal{A}_{\mathsf{eVSE}}$ submits a set of attributes (with the restriction that no attributes from either challenge set are used) to $\mathcal{A}_{\mathsf{SKE}}$. $\mathcal{A}_{\mathsf{SKE}}$ responds to $\textsc{BuildIndex}(\cdot, K_O, PP)$ queries using Encode to produce the pre-index from the set of attributes then runs $\mathsf{ABE.KeyGen}$ to produce the relevant index. To query $\textsc{Search}(\cdot,\cdot, K_S, PP)$ $\mathcal{A}_{\mathsf{eVSE}}$ submits their challenge index $\mathcal{I}_{\mathcal{D}_b}$ and QT_Q to $\mathcal{A}_{\mathsf{SKE}}$ which runs Search using these values along with K_S and PP to produce search results R which are returned to $\mathcal{A}_{\mathsf{eVSE}}$. To allow verification of search results $\mathcal{A}_{\mathsf{eVSE}}$ can query $\textsc{Verify}(\cdot,\cdot, PP)$ which is run by $\mathcal{A}_{\mathsf{SKE}}$. $\mathcal{A}_{\mathsf{eVSE}}$ submits the result R along with the verification token VK_Q to $\mathcal{A}_{\mathsf{SKE}}$ which runs Algorithm 5.6 and returns RT_Q to $\mathcal{A}_{\mathsf{eVSE}}$. Calls to $\textsc{Retrieve}(\cdot,\cdot,\cdot, PP)$ are answered by $\mathcal{A}_{\mathsf{SKE}}$ by running Algorithm 5.7 and returning the output to $\mathcal{A}_{\mathsf{eVSE}}$.

9. $\mathcal{A}_{\mathsf{eVSE}}$ outputs their guess $\hat{b}'$ for $\hat{b}$.

10. $\mathcal{A}_{\mathit{IND\text{-}CPA}}$ outputs their guess $b' = \hat{b}'$ as their guess for b.

11. As we have assumed that $\mathcal{A}_{\mathsf{eVSE}}$ has a non-negligible advantage, δ, in the Index Privacy game:

$$Adv_{\mathcal{A}_{\mathsf{eVSE}}} = \delta,$$

we have that:

$$\mathbb{P}[\hat{b}' = \hat{b}] = \delta + \frac{1}{2}.$$

Using these assumptions we now calculate the advantage of $\mathcal{A}_{\mathsf{SKE}}$ against the challenger, $\mathcal{C}$, in their game. We start by calculating the probability of $\mathcal{A}_{\mathsf{SKE}}$ winning their game, $\mathbb{P}[b' = b]$, as follows:

$$\begin{aligned}
\mathbb{P}[b' = b] &= \mathbb{P}[b' = b | \hat{b}' = \hat{b}]\mathbb{P}[\hat{b}' = \hat{b}] + \mathbb{P}[b' = b | \hat{b}' \neq \hat{b}]\mathbb{P}[\hat{b}' \neq \hat{b}] \\
&= 1 \cdot \mathbb{P}[\hat{b}' = \hat{b}] + 0 \cdot \mathbb{P}[\hat{b}' \neq \hat{b}] \\
&= \delta + \frac{1}{2}.
\end{aligned}$$

Hence,

$$\begin{aligned} Adv_{\mathcal{A}_{\mathsf{SKE}}} &= \left|\mathbb{P}[b' = b] - \frac{1}{2}\right| \\ &= \left|(\delta + \frac{1}{2}) - \frac{1}{2}\right| \\ &= \delta \end{aligned}$$

As we fixed δ to be non-negligible we have that the advantage of $\mathcal{A}_{\mathsf{SKE}}$ is also non-negligible which contradicts the IND-CPA security of the symmetric encryption scheme SKE. We conclude that if the symmetric encryption scheme SKE is IND-CPA secure then eVSE as defined in Algorithms 5.1-5.8 is secure in the sense of Index Privacy.

□

5.4.3 Query privacy

Lemma 5.4.3. eVSE *as defined in Algorithms 5.1–5.8 is secure against Query Privacy (Game 5.3) under the same assumptions as in Theorem 5.3.1.*

Proof. Suppose $\mathcal{A}_{\mathsf{eVSE}}$ is an adversary with non-negligible advantage against the Query Privacy game (Game 5.3) when instantiated with Algorithms 5.1–5.8. We begin by defining two games:

- **Game 0**: This is the selective Query Privacy game as defined in Game 5.2.
- **Game 1**: This is the same as **Game 0** with the modification that we use a pseudorandom permutation in Algorithm 5.1 to construct $\tilde{\Omega}$.

We show that an adversary with non-negligible advantage against the selective Query Privacy game can be used to construct an adversary, $\mathcal{A}_{\mathsf{SKE}}$, which may break the IND-CPA security of a symmetric key encryption scheme SKE = ($\mathsf{SKE.KeyGen}$, $\mathsf{SKE.Encrypt}$, $\mathsf{SKE.Decrypt}$).

Game 0 to Game 1. See **Game 0** *to* **Game 1** in proof of Lemma 5.4.2 for details.

Reduction to IND-CPA. We now show that using $\mathcal{A}_{\mathsf{eVSE}}$ in **Game 1**, $\mathcal{A}_{\mathsf{SKE}}$ can break the IND-CPA security of a symmetric key encryption scheme SKE. That is, given a challenge ciphertext c which is an encryption m_b where $b \xleftarrow{\$} \{0,1\}$, $\mathcal{A}_{\mathsf{SKE}}$ can distinguish whether c is an encryption of m_0 or m_1.

This proof follows in the spirit of the proof of Lemma 5.4.2. Let $\mathcal{C}$ be the challenger for $\mathcal{A}_{\mathsf{SKE}}$ and $\mathcal{A}_{\mathsf{SKE}}$ will act as the challenger for $\mathcal{A}_{\mathsf{eVSE}}$.

1. $\mathcal{A}_{\mathsf{eVSE}}$ chooses their challenge queries: Q_0, Q_1 with the restriction that all gates match in both queries (the gates are denoted by $\mathbb{G}_{Q_0}$ for Q_0 and $\mathbb{G}_{Q_1}$ for Q_1). Note that a rational adversary will always choose queries where the input attributes differ in at least one position. We denote the sets of attributes contained in the queries as $(q_{0,1}, q_{0,2}, ..., q_{0,t})$, $(q_{1,1}, q_{1,2}, ..., q_{1,t}) \subseteq \ \Omega$ for Q_0 and Q_1 respectively (note that if all the (binary) gates in each query match the two sets of attributes will be the same size). $\mathcal{A}_{\mathsf{eVSE}}$ submits Q_0 and Q_1 to $\mathcal{A}_{\mathsf{SKE}}$.
2. The challenger $\mathcal{C}$ chooses a bit $b \xleftarrow{\$} \{0,1\}$.
3. KeyGen is run between $\mathcal{C}$ and $\mathcal{A}_{\mathsf{SKE}}$:
   - $\mathcal{A}_{\mathsf{SKE}}$ generates the secret key for the broadcast encryption scheme $\mathsf{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Encrypt}, \mathsf{BE.Add}, \mathsf{BE.Decrypt})$:
   $$k_{\mathsf{BE}} \leftarrow \mathsf{BE.KeyGen}(1^{\kappa}, |\mathcal{U}|).$$
   - $\mathcal{C}$ generates the secret key for the symmetric encryption scheme SKE:
   $$k_{\mathsf{SKE}} \leftarrow \mathsf{SKE.KeyGen}(1^{\kappa}).$$
   - $\mathcal{C}$ provides $\mathcal{A}_{\mathsf{SKE}}$ with access to the following oracles $\text{SKE.}\textsc{Encrypt}(\cdot, k_{\mathsf{SKE}})$: which takes as input a plaintext m and returns its symmetric encryption under k_{SKE} and $\text{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ which takes as input two plaintexts m_0, m_1 and a bit b and outputs the symmetric encryption of m_b under k_{SKE}.
   - For all $d_{0,j} \neq d_{1,j}$, $\mathcal{A}_{\mathsf{SKE}}$ submits the pair $(d_{0,j}, d_{1,j})$ (where such pairs exist as the challenge sets are not equal) to $\text{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ and receives challenge ciphertexts, c_j, in return.
   - $\mathcal{A}_{\mathsf{SKE}}$ now computes Ω'. For every pair $(q_{0,i}, q_{1,i}) : q_{0,i} \neq q_{1,i}$, $\mathcal{A}_{\mathsf{SKE}}$ submits the pairs $(q_{0,i}, q_{1,i})$ and $(q_{1,i}, q_{0,i})$ to $\text{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$. For all other pairs $(q_{0,t}, q_{1,t})$ we have that $q_{0,t} = q_{1,t}$, hence $\mathcal{A}_{\mathsf{SKE}}$ submits $q_{0,t}$ to $\text{SKE.}\textsc{Encrypt}(\cdot, k_{\mathsf{SKE}})$ and includes the output in Ω'. To ensure Ω' contains an encryption of each attribute in Ω, $\mathcal{A}_{\mathsf{SKE}}$ submits the remaining attributes of Ω: $a_n \notin (Q_0 \cup Q_1)$ to $\text{SKE.}\textsc{Encrypt}(\cdot, k_{\mathsf{SKE}})$ and includes the output in Ω'. Ω' now contains an encryption of every element in Ω. In order to map

elements from Ω to Ω' (which is required to run Encode), every attribute in Ω needs to have a corresponding ciphertext in Ω' which is defined as follows: for each pair of attibutes that were submitted to $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$ we let the output denote the corresponding element in Ω' for the attribute on the left hand side of the submitted pair. For all attributes in Ω submitted to $\mathrm{SKE.E{\scriptsize NCRYPT}}(\cdot, k_{\mathsf{SKE}})$ the output is the corresponding element in Ω' for that attribute. This defines a bijective mapping of attributes in Ω to elements in Ω' as required.

- $\mathcal{A}_{\mathsf{SKE}}$ defines the set $\tilde{\Omega} \leftarrow \Pi(\Omega')$, where Π is a pseudorandom function, and runs:
$$(k_{\mathsf{ABE}}, PP_{\mathsf{ABE}}) \leftarrow \mathsf{ABE.KeyGen}(1^{\kappa}, \tilde{\Omega}).$$
- $\mathcal{A}_{\mathsf{SKE}}$ selects the data owner state: $st_O \xleftarrow{\$} \{0,1\}^{\kappa}$ and then runs:
$$k_S \xleftarrow{\$} \mathsf{BE.Add}(k_{\mathsf{BE}}, S),$$
to produce the first part of the server key.
- The authorised user group is initialised as: $\mathcal{G} \leftarrow \{S\}$.
- $\mathcal{A}_{\mathsf{SKE}}$ generates the server state by encrypting the data owner state using the BE scheme:
$$st_S \xleftarrow{\$} \mathsf{BE.Encrypt}(st_O, \mathcal{G}, k_{\mathsf{BE}}).$$
- The public parameters are set to: $PP = (PP_{\mathsf{ABE}}, PP_{\mathsf{BE}}, \tilde{\Omega})$ and the server key is set to $K_S = (k_S, st_S)$. These are both given to $\mathcal{A}_{\mathsf{eVSE}}$.

4. A user, u, is selected at random by $\mathcal{A}_{\mathsf{SKE}}$ and enrolled as an authorised user:
$$k_u \xleftarrow{\$} \mathsf{BE.Add}(k_{\mathsf{BE}}, u).$$

5. The challenge bit for $\mathcal{A}_{\mathsf{eVSE}}$, $\hat{b}$, is set to b.

6. $\mathcal{A}_{\mathsf{SKE}}$ creates a pre-query $\tilde{Q}_b$ to encode into the challenge query for $\mathcal{A}_{\mathsf{eVSE}}$. This is done using Encode, which takes as input the elements from set of attributes corresponding to the challenge query Q_b and maps them to elements in Ω'. Due to the way the mapping is defined using $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, b)$, this will produce a pre-query containing encryptions of attributes corresponding to the challenge query Q_b. If $b = 0$ then then $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, 0)$ would have been used for encryption hence the attributes in Q_0 would have been encrypted and be included in the pre-query, wheres if $b = 1$ then $\mathrm{LR}(\cdot, \cdot, k_{\mathsf{SKE}}, 1)$ would have been used for encryption hence

the attributes corresponding to Q_1 would have been encrypted and be included in the pre-query.

7. $\mathcal{A}_{\mathsf{SKE}}$ runs Query using the pre-query created in step 4 to create the challenge query $(QT_{Q_b}, VK_{Q_b}, RK_{Q_b})$ for $\mathcal{A}_{\mathsf{eVSE}}$.

8. $\mathcal{A}_{\mathsf{eVSE}}$ is given oracle access to $\textsc{BuildIndex}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\textsc{Search}(\cdot,\cdot,\cdot,PP)$, $\textsc{Verify}(\cdot,\cdot,\cdot,\cdot,PP)$ and $\textsc{Retrieve}(\cdot,\cdot,\cdot,PP)$. To query $\textsc{BuildIndex}(\cdot, \mathrm{MK}, \mathrm{PP})$, $\mathcal{A}_{\mathsf{eVSE}}$ submits a set of attributes to $\mathcal{A}_{\mathsf{SKE}}$ (with the restriction that the search results produced when searching the corresponding index cannot be used to distinguish the two challenge sets, i.e. the search results have to be the same). $\mathcal{A}_{\mathsf{SKE}}$ responds to $\textsc{BuildIndex}$ queries using Encode to produce the pre-query from the set of attributes then runs $\mathsf{BuildIndex}$ to produce the relevant index. $\mathcal{A}_{\mathsf{SKE}}$ responds to queries to $\textsc{Search}, \textsc{Verify}, \textsc{Retrieve}$ by running the relevant algorithms using the adversary's input.

9. $\mathcal{A}_{\mathsf{eVSE}}$ outputs their guess $\hat{b}'$ for $\hat{b}$.

10. $\mathcal{A}_{\mathsf{SKE}}$ outputs their guess $b' = \hat{b}'$ as it's guess for b.

11. As we have assumed that $\mathcal{A}_{\mathsf{eVSE}}$ has a non-negligible advantage, δ, in the Query Privacy game:
$$Adv_{\mathcal{A}_{\mathsf{eVSE}}} = \delta,$$
we have that:
$$\mathbb{P}[\hat{b}' = \hat{b}] = \delta + \frac{1}{2}.$$
Using these assumptions we now calculate the advantage of $\mathcal{A}_{\mathsf{SKE}}$ against the challenger, $\mathcal{C}$, in their game. We start by calculating the probability of $\mathcal{A}_{\mathsf{SKE}}$ winning their game, $\mathbb{P}[b' = b]$, as follows:
$$\begin{aligned}\mathbb{P}[b' = b] &= \mathbb{P}[b' = b|\hat{b}' = \hat{b}]\mathbb{P}[\hat{b}' = \hat{b}] + \mathbb{P}[b' = b|\hat{b}' \neq \hat{b}]\mathbb{P}[\hat{b}' \neq \hat{b}] \\ &= 1 \cdot \mathbb{P}[\hat{b}' = \hat{b}] + 0 \cdot \mathbb{P}[\hat{b}' \neq \hat{b}] \\ &= \delta + \frac{1}{2}.\end{aligned}$$

Hence,

$$\begin{aligned} Adv_{\mathcal{A}_{\mathsf{SKE}}} &= \left|\mathbb{P}[b' = b] - \frac{1}{2}\right| \\ &= \left|(\delta + \frac{1}{2}) - \frac{1}{2}\right| \\ &= \delta \end{aligned}$$

As we fixed δ to be non-negligible we have that the advantage of $\mathcal{A}_{\mathsf{SKE}}$ is also non-negligible which contradicts the security of the IND-CPA security of the symmetric encryption scheme SKE. We conclude that if the symmetric encryption scheme SKE is IND-CPA secure then eVSE as defined in Algorithms 5.1-5.8 is secure in the sense of Query Privacy.

□

5.5 Summary

Our scheme extends the expressiveness of queries that can be achieved in VSE. No other VSE schemes to our knowledge are able to perform the range of search queries or include negation of keywords in their search queries. Additionally our scheme leaks neither the access nor the search pattern to the server whilst executing a search. Our combination of search queries with computational queries is also a novel functionality in the field of VSE.

The search time, and size of the queries and search results are linear in n (the number of data items stored on the remote server). Due to this eVSE may be more suited to smaller databases to prevent these features from being prohibitively expensive. The VSE scheme of [44] has a search time that is linear in the number of letters in the queried keyword (which is usually much smaller than n). This faster search is achieved using a tree-based index, however only a single keyword equality search can be performed. Another scheme that uses ABE primitives in the construction, as we do, is that of [137]. This scheme is able to achieve multi-level access, where users can be restricted to searching only certain parts of the database. Keywords are grouped with respect to their access control policies, and the search time is linear in the number of groups. However, this scheme also only achieves a single keyword equality search. The scheme of Wang et al. [132] achieves verifiable fuzzy keyword search with a search time that is linear in the size of the fuzzy keyword set (which varies depending on the level of fuzziness required i.e. searching for data items that contain keywords of edit distance two will require a larger fuzzy keyword set than searching for keywords with an edit distance of one from the queried keyword [97]). The scheme by Kurosawa et al. [94] uses an SSE scheme along with a MAC on the search results to achieve verification and adding a new data item requires time linear in the number of keywords associated with the new data item. In [127], a scheme by Stefanov et al., adding a new data item takes time $\mathcal{O}(\log^2 N)$ and this scheme, similarly to [94], also uses a MAC to verify the search results. Both [94, 127] only achieve single keyword equality search in the symmetric key setting. A dynamic scheme that can support conjunctive queries (the most expressive query type of all the dynamic schemes in Table 5.3) is that of [128] which is constructed using public key primitives, namely ABE, to create the indexes. Each user in the system has a separate public key to create their indexes which can be added to the server at anytime. This scheme uses a combination of Bloom filters and signatures to achieve verifiability of search results. Cheng et al. [51] construct a VSSE from indistinguishability obfuscation that can handle boolean and conjunctive queries. Their scheme is able to achieve public verifiability by implementing a specific public verification circuit.

In terms of the number of rounds of communication required per search, our scheme is optimal requiring only one round of communication. The size of the search results in our scheme is also linear in n. Most VSE schemes in the literature return results of a size that is linear in the number of data items that match the query, however this method leaks the access pattern which in turn may leak information about the query. Our scheme hides the access pattern as all search results are of the same form, regardless of what query was submitted.

In terms of security, as illustrated in our security games, our scheme achieves public verifiability, index privacy and query privacy (in terms of the keywords searched for), which is comparable to other VSE schemes that have been discussed. Overall, our scheme sacrifices efficiency when compared to existing VSE schemes, but gains much increased functionality and query expressiveness. Furthermore, our scheme currently only supports static data and future work will be needed to extend our scheme to support a dynamic data set as in the schemes in [94, 127, 128].

Table 5.3 gives a brief comparison between our scheme and those in the literature as discussed above and throughout the chapter. The abbreviation AP stands for *access pattern* and SP stands for *search pattern*.

Table 5.3: Comparison of Verifiable Searchable Encryption Schemes

Scheme	Data type	Query type	Publicly Verifiable	Leakage	Computations
[130]	Static	Ranked equality	No	AP,SP	No
[94]	Dynamic	Equality	No	AP	No
[128]	Static	Conjunctive, Disjunctive	No	AP	No
[129]	Dynamic	Equality	No	AP	No
[127]	Dynamic	Equality	No	AP, SP	No
[137]	Static	Equality	No	AP	No
[131]	Static	Fuzzy	No	AP, SP	No
[65]	Static	Semantic	No	AP, SP	No
[44]	Static	Equality	No	AP, SP	No
[51]	Static	Conjunctive	Yes	AP, SP	No
Our scheme	Static	Conjunctive, Disjunctive, Arbitrary CNF/DNF formulae, NC^1	Yes	None	Yes

Chapter 6

Multi-level Searchable Encryption

Contents

In this chapter we extend the paradigm of searchable symmetric encryption to support multiple users with different access levels. We call this multi-level access. We look at general methods that can take any single-user searchable symmetric encryption scheme as a black box and extend it to support multi-level access as well as a specific searchable symmetric scheme that supports multi-level access. The results of this chapter have appeared in the proceedings of Financial Cryptography and Data Security 2017.

6.1 Introduction

Many searchable symmetric encryption (SSE) schemes, single user and multi-user, in the literature only consider the scenario where a querier is either authorised to search over the entirety of the encrypted data or not at all [33, 39, 55, 126], in which case (ideally) the querier should learn nothing about the outsourced data. In practice, however, outsourced data sets could be large and the access control requirements of such data sets are likely to be more fine-grained than the binary 'all or nothing' approach above; hence existing schemes do not suffice.

We study the problem of enforcing a multi-level access control policy (MLA), or information flow policy, in the context of searchable symmetric encryption, where the data items are classified at different levels (which may or may not be hierarchal) and users are assigned access rights corresponding to the data items they are authorised to view. A notable example of this type of data classification is government data. In the UK the government uses three levels of classification for its data: official, secret and top secret [110]. Employees are assigned an access level that determines which data items they are authorised to view. In our model, a user with 'secret' clearance, should be unable to learn any information about data items classified as 'top secret', such as whether they contain searched keywords or not. This is an example of an information flow policy with a total order of security labels [18]. The SE model in Section 3.1 needs to be adapted so that users can only receive search results corresponding to encrypted data items that they are authorised to view, to prevent any leakage of information to unauthorised users. We provide a rigorous definitional framework for Multi-level Searchable Symmetric Encryption (MLSSE), introduce appropriate security notions and present several generic solutions to MLSSE using SSE as a black box as well as a provably secure construction.

We focus on SKE solutions to MLA in SE to produce schemes that are easy to deploy and efficient to use in real world scenarios. This branch of SE with access control has, up to now, received little to no attention in the literature.

6.1.1 Related work

Kissel et al. [92] present a scheme supporting multiple groups of users searching over encrypted data where each group has a specified dictionary of keywords they are allowed to search over. These groups are arranged hierarchically so that users at one level in the hierarchy are able to search for all keywords in dictionaries assigned to groups at lower levels in the hierarchy. Although this scheme presents a form of hierarchal access in SSE, the user is still able to search over the entire data set. In most access control scenarios, we are concerned with protecting a data item (i.e. the complete content of a data item), not just a single keyword describing the data item. Furthermore, we believe that expressing an access control policy in terms of authorised keywords can potentially be difficult to administer correctly. Sensitive data items may well gain their classification level due to semantic meaning regarding their contents (for example, the subject to which they pertain), which may not trivially be captured through the associated keywords. For example, consider two data items that contain information about company spending; one data item contains a public report of company-wide spending, whilst the other pertains specifically to the research department. Clearly, both data items may be labelled by a keyword such as 'finance', but detailed knowledge of the research spending alone may be more sensitive than a generalised report of the entire company. Thus, simply authorising users to search for keywords, such as 'finance', does not suffice in this instance as not all users that can search the public report should be able to view the specific report. The access control policy in this case must be managed carefully — perhaps additional, more granular, keywords must be defined e.g. 'finance-public' (leading to an increase in the size of the searchable encryption index and a subsequent loss of efficiency) or a (less efficient) SE scheme that supports 'conjunctive keyword-only access control' would be required such that one can be authorised to search for ('finance' AND 'public') and only data items with *both* keywords would be returned. In this work, we consider the problem of fine-grained classification of data items *directly* and gain a more efficient solution.

There are solutions to *multi-level access (MLA)* in the area of PKE. In particular functional encryption, has been used to achieve MLA in SE [25, 83, 99, 136]. PKE, in general, is computationally more intensive and harder to implement than SKE which is often built using much simpler operations, perhaps making PKE less suitable for systems required for real world deployment.

The notion of multi-user SSE was introduced by Curtmola et al. [55]. In this work they combine a single-user SSE scheme with a broadcast encryption scheme that is used to add and revoke users in the system. They define the security of their multi-user SSE scheme similarly to that of the single-user SSE scheme with the additional security notion of *revocation*, which requires that a revoked user is no longer able to perform searches on the index stored on the server. In order to produce a search query a user needs to be authorised by the data owner. This is done by adding the user to an authorised user set. Initially a random value is chosen by the data owner, which is referred to as the data owner state. This value is then encrypted using the broadcast encryption scheme, so that only the authorized users are able to decrypt it, and broadcast to all users. The data owner state is then used as a key for a PRP that is applied to the search query before it is sent to the server. During the search phase the server checks if the search query was correctly formed by applying the inverse of the PRP. If it is not, then the server outputs a failure symbol $\perp$ and the search is not performed. As only authorised users have access to the data owner state this means that only authorised users are able to form valid search queries. Each time a user is revoked the data owner state is renewed and re-encrypted according to the updated authorised user set. The access control to the data in this scheme is very coarse grained, meaning either a user is authorised to search over all of the index or none at all.

Work by Yang [135] introduce the concept of query accountability within multi-user SSE, by issuing a unique query key to each authorised user. For each of these unique query keys the server holds a corresponding *helper* key which is related to the user's query key: the helper key is a group element g whose exponent contains the inverse of the user's query key. Their scheme uses bilinear maps to mask information in the index and compute the search. The index entries and queries are constructed in such a way that when the search query is combined with the helper key we get a cancellation in the exponent of a bilinear map and can use the hash of the resulting group element to check whether a particular index entry contains the keyword associated with the search query. The major drawback of this scheme is that each data item is only indexed by one searchable keyword, which does not support a very fine grained search. In order to retrieve relevant search results using only one keyword in the index for each data item may require the user to know some information a priori regarding the data set. This might not be realistic in this multi-user scenario (Scenario 3) as the users searching the data items do not own the data. An extension to the main scheme is detailed that permits a form of multi-level access where each user is granted a set of keywords they are authorised to search for. This is different to our definition of multi-level access where a user's search is restricted to the set of data items that they are authorized to view. Only restricting the set of keywords a user is authorised to search for still permits the user to search all the data items. The scheme of [15], which the scheme of [135] is based on, supports users writing data to the server as well as just reading it (Scenario 4). The revocation of users in both [15] and [135] is accomplished by instructing the server to delete the helper key that they posses for the revoked user. This method of revocation relies on the server being at most honest-but-curious (Definition 3.3.1) in order to guarantee that the helper keys are actually erased by the server. A scheme by Kissel et al. [92] also achieves a form of multi-level search by restricting the set of keywords that a user can search for similarly to [135]. This scheme uses key regression [64] and broadcast encryption to add and revoke users from the system. The data owner state as described above in relation to [55] is produced using the key regression scheme and renewed using the same process when a new user is enrolled or revoked. The new values are distributed to the users using a broadcast channel. In terms of revocation the security notion guarantees that revoked users are not able to issue successful queries after they are revoked, as in [55].

A paper by Cao et al. [40] presents a multi-user scheme where the users can perform ranked searches, where the search results are ranked by the server according to their relevance to the search query. They do not explicitly detail the user addition and revocation procedures but state that they use broadcast encryption to achieve these tasks as in [55].

There are a few solutions to multi-level search in the public key setting where users are able to both read and write data (Scenario 4). Hattori et al. [75] present a scheme which supports a hierarchy of users that can perform conjunctive keyword searches on the index. They define the new notion of ciphertext-policy hidden vector encryption (CP-HVE). Hidden vector encryption (HVE) was previously only defined in the key policy setting (KP-HVE) where a secret key is associated with a vector that defines a policy and a ciphertext associated with a vector of attributes. It was first defined for searchable encryption by Katz et al. in their seminal paper [89]. The secret keys that define policies are used as search tokens and the index entries are ciphertexts associated with a vector of attributes, this method of SE is known as predicate-based encryption (Definition 2.3.7). It is very similar to the notion of ABE (Definition 2.3.5) which associates a ciphertext with a policy and a secret key with attributes but PBE requires that the policies are hidden. The new notion of CP-HVE is the dual concept of KP-HVE where the secret keys are associated with hidden policies and the ciphertexts associated with attributes. In this scheme a search result is only returned if the policy on the ciphertext is fulfilled by the attributes associated with the user that generated the search query and the keyword(s) in the search query are contained in the data item. There is no revocation mechanism for users defined in this scheme. The scheme of [76] presents a public key conjunctive keyword search scheme that supports multiple users, but also does not consider the revocation of users. An extension is discussed that allows the data owner to choose which users can search their data items when creating an index, however this involves making multiple indexes if the data owner wishes their data to be searchable by multiple users. It also requires the data owner to know the public key of each user they wish to authorise.

The majority of multi-level search schemes that support multiple users searching the data but not writing data (Scenario 3) involve some from of attribute based encryption (ABE) in the construction. The scheme of [83] is based on the linked list construction of Curtmola et al. [55], but the concept behind their construction could be applied to most single user SSE schemes. It is an example of one of our generic solutions to multi-level search which is defined in Section 6.3.2, and achieves multi-level search by encrypting the search results with ABE. Usually in an SE scheme, the search results consist of identifiers of data items that satisfy the search query, the user does not need to process these search results any further. Encrypting the search results with ABE ensures that the user can only view search results that they are authorised to by the policy defined on their ABE secret key. However this scheme reveals the total number of search results, even when the user is not authorised to decrypt all of them. To prevent this leakage, they propose another scheme using a trusted third party, another one of our generic methods outlined in Section 6.3.3, that intercepts the search results and sorts them according the user's access rights before sending the final set of search results to the user. The work of [52] points out the high computational overheads of ABE and presents a scheme that outsources the key generation and decryption processes associated with the ABE scheme. A solution that uses the techniques of [52] for multi-level search in Scenario 3 is that of [62]. To reduce the computational overheads of ABE in this scheme the data owner and users are able to outsource their ABE computations to a proxy server. Another extra entity is also introduced into the system model: a trusted authority (TA) which is used to setup the system and revoke users. The user revocation mechanism is similar in this scheme to that of [15, 135] as a helper key is stored on the server for each user. When a user is revoked from the system the TA instructs the server to delete the relevant helper keys. The search query generation is a two step process in this scheme as a user needs to send a search query request to the proxy server before they are able to generate a search query. The search is also a two step process as the server first verifies that the user's attribute set associated with the search query is genuine. The server does this by creating a CP-ABE ciphertext from a random plaintext m with the policy of the ciphertext being the intersection of all the users attributes. The user then attempts to decrypt this ciphertext using their secret key (which is associated with their attributes) and sends the resulting plaintext back to the server. If this plaintext matches the original plaintext m that the server encrypted then this verifies that the user does in fact possess the attributes that it claims to have.

As the attributes in this scheme are not encrypted the server identifies which set of data items the user is authorised to search and performs the search over those data items only. The scheme of Ye et al. [98] adds another server into the system model and considers the additional security notion of a revoked user colluding with one of the servers. The index is duplicated across the two servers. Each server stores helper keys similarly to other schemes discussed above, which are then deleted by the servers when a user is revoked. This scheme provides a verification mechanism for the search which compares the two sets of search results, if they are equal then the search is verified as correct and if not then the search results are rejected. The fine-grained access control is implemented as a threshold check on the users attributes before search. If the user's attribute set contains a fixed amount (or greater) of matching attributes to that of the data item, then they are allowed to search that data item.

6.1.2 Organisation of chapter

In Section 6.2 we outline the notion of SSE along with descriptions of the type of indexes that are used in SSE and the type of leakage that can occur. In Section 6.3 we describe some methods of achieving multi-level access using a single-user SSE scheme as a black box. In Section 6.4 we define the system and security models of our multi-level SSE scheme, and our construction is defined in Section 6.5. Our security proofs are given in Section 6.6. We conclude the chapter in Section 6.8 by summarising the findings of the chapter.

6.2 Searchable Symmetric Encryption

This section gives an overview of Searchable Symmetric Encryption (SSE). The definition of SSE can be found in Definition 3.1.1, here we discuss some finer details of SSE such as the different types of index used and the leakage associated with SSE.

6.2.1 Types of index

There are three main types of index used in SSE schemes in the literature: *forward index, inverted index* and *tree based index*. Each of these types of indexes have advantages and disadvantages and the choice of index will depend on the data that needs to be indexed e.g. dynamic or static data and any restrictions imposed by the data owner, such as a fast search time.

1. **Forward Index** A forward index contains an entry per data item and lists the set of keywords (or attributes) associated with each data item. It is relatively

straightforward to accommodate dynamic data sets using a forward index; when adding an encrypted data item to the server one can just create a new entry in the index for that data item. Furthermore, deleting or amending a data item on the server only affects the part of the index associated with that data item and does not require changes to any other part of the index. The search time on a forward index will be linear in the number of data items, so this type of index may not be suitable for large data sets. However, this process could be parallelized to improve the search time, say we use p processors then the search time complexity is reduced to $\mathcal{O}(\frac{n}{p})$. Some examples of SSE schemes employing a forward index are [46, 68, 71].

2. **Inverted index** An inverted index (also referred to as a postings list) contains an entry per keyword (or attribute) and lists the set of data items that contain each keyword. An inverted index can produce a search time linear in the number of data items that match the search query which is optimal since the amount of work to retrieve the encrypted data items is linear in the number of data items that match the search query, hence this is the minimal amount of work that needs to be done. However, these indexes are not so well suited for use with dynamic data sets as the update process has been shown to be very complex and leak extra information from the indexes [85]. The scheme of [85] also shows that an SSE scheme that uses an inverted index may not permit a parallel search. Some examples of SSE schemes employing an inverted index are [55, 86].

3. **Tree-based index** In a tree-based index the characters of the keywords in the data items are inserted into a tree sequentially from root to leaf and the identifiers for the data items containing that keyword are stored in the leaves of the tree at the end of this path. A tree-based index provides a sub-linear search time and can also be parallelized. This type of index can also handle updates efficiently, however has a higher space complexity then the other two types of index. An example of SSE schemes employing a tree-based index is [85].

6.2.2 Classifying leakage

Almost all schemes designed for efficient search over encrypted data leak some form of information from the encrypted data items stored on the server. Until recently the possible attacks on the queries and the index that could be mounted using the available leakage in a SE had not been investigated, however there have been several recent papers that investigate this [41, 59, 79, 102]. Classifying the leakage in SE is an important feature of the security definition of a SE scheme as it tells the user what information will be available to a potential adversary. This will help the user to decide which SE scheme is most fit for purpose when implementing a scheme. The information leakage in a SSE scheme can be sorted into four categories, each one relating to a specific phase in the scheme (the add and delete leakage only corresponds to dynamic SSE schemes):

1. **Setup leakage** The setup leakage (see Definition 3.2.3) is the information leaked to the server from the index. This will include information such as the size of the index. Depending on how the information in the index is encrypted and padded this information may also include the total number of data items indexed or the total number of distinct keywords in all the data items.

2. **Search leakage** The search leakage (see Definition 3.2.4) is the information leaked as the product of a search query being evaluated over the index. This will include information such as the access pattern and the search pattern (Definition 3.2.1 and Definition 3.2.2 respectively).

3. **Add leakage** This is the information leaked to the server when a new data item is added to the index. This could include information such as the encryptions of the keywords contained in the new data item (these are usually deterministic) and the number of keywords associated with the new data item. If the search is sequential (as in [86]) then information such as the location of the previous or next data item in the sequence could be revealed.

4. **Delete leakage** This is the information leaked to the server when a data item is deleted from the index. The information here is similar to that leaked when adding a data item. Note that depending on the type of index then it is possible to not leak any information when adding or deleting a data item. If you consider a forward index then you will simply need to add or remove the row in the index that corresponds to that data item.

6.3 Generic solutions for MLSSE using SSE as a black box

In this section we describe methods of achieving multi-level access in searchable symmetric encryption using any generic single user SSE scheme in a black box manner. These methods can all be applied to any single user SSE scheme. Whether the scheme is able to accommodate dynamic data or not will be determined by the underlying SSE scheme.

For each method we look at two scenarios: one where the access levels of the data items are hidden and one where they are not. We analyse the following factors in each scenario and compare them to those of the underlying SSE scheme:

- How much storage space is needed on the server.
- Time required for index generation and search.
- Computation required for index generation and search.
- Leakage during setup and search.

In terms of leakage, each scheme leaks the access pattern, which is standard in most SSE schemes. One can assume an IND-CPA secure symmetric encryption scheme is used to encrypt the data items themselves, so no information is leaked from the ciphertexts, except their size (the size can also be hidden if required by padding each ciphertexts to the same size). However, we consider a *structure only* searchable encryption scheme, which does not detail the retrieval and decryption of the encrypted data items. We also assume that in the "non-hidden access levels" method the access levels of both the user and the data items are revealed to the server. Hence, in relation to this variant we only discuss the leakages associated with the search pattern and index information.

6.3.1 Augmented Index

- **Non-hidden access levels**: Multi-level access can be supported by augmenting the index in an SSE scheme. Users in each access level will share a secret key that is used to generate search queries. A separate index, labelled with the relevant access level will be created for each access level using the associated secret key. A user can derive their search queries using their secret key and send them to the server along with their access level to allow the server to determine which index to search. The data items only need to be encrypted once and stored on the server, along with the set of indexes. This method relies on the server to enforce access control, so is only suitable for honest and honest-but-curious adversarial models.

- **Hidden access levels**: This works similarly to the non-hidden access levels case, however this method is only compatible with SSE schemes that use either forward or inverted indexes. The indexes are generated as above using the relevant secret keys and concatenated to create the rows of the indexes and then randomly shuffled using a pseudo-random permutation, to create one large index. The data items only need to be encrypted once and stored on the server, along with the (set of) indexes. Each secret key will only be able to produce valid search queries for the rows of the index that were generated using that secret key (due to the robustness of the encryption), hence users will only be able to search over and retrieve search results corresponding to their access level. Due to the permutation of the rows in the indexes the server will not know whether a row in the index produced $\perp$ in response to the search query not being satisfied or if the search query was not valid for that row's access level. This conceals the access level of the user from the server.

Storage

The storage required for both hidden and non-hidden access level schemes is increased by a factor of ℓ (where ℓ is the number of access levels) compared with the underlying SSE scheme. If we have hierarchal access levels there will be repetitions in the indexes (for example if a data item can be viewed by more than one access level, which is typical in hierarchal access), this will require more entries in the indexes if using forward indexes that have entries per data item.

Time

The time required for the index generation phase will be greater by a factor of l than that of the underlying SSE scheme as multiple indexes need to be produced. The search time for the non-hidden access level method will be the same as that of the underlying SSE scheme as the server only searches one of the indexes, whereas the hidden access levels method requires the server to traverse the entirety of the augmented index in order to compute the complete search, increasing the search time by a factor of ℓ. The efficiency of this method could be enhanced by searching the index in parallel. If there were ℓ processors then the search could be performed in the same time as using one processor to search one index.

Computation

The computation required for the index generation phase will be greater by a factor of ℓ than that of the underlying SSE scheme as multiple indexes need to be produced. The computation required during the search phase in the non-hidden access levels method will be the same as the underlying SSE scheme as only one index is searched, whereas the search computation will increase by a factor of ℓ in the hidden access levels method.

Leakage

Both methods achieve search pattern privacy across access levels, that is only repeated queries for a keyword at the same access level is detectable, search queries for the keyword at a different access level will be indistinguishable. However it might be possible to link search queries together using the information in the search results. For example if one set of search results is the subset of another set of search results then this may indicate that the search queries that generated these search results are for the same keyword, just at different access levels. There is no additional index information leakage than that of the underlying SSE scheme. There is no other additional leakage (other than the assumed leakage as stated at the beginning of the section).

6.3.2 Encrypted/labelled Search Results

- **Non-hidden access levels**: The scheme is set up analogously to the single user SSE scheme with one secret key used to create the index and shared between users. Each identifier in the set of search results is labelled with the relevant access level. A user submits their access level along with their search query (computed using the secret key). The server returns only the search results that satisfy the search query and that the user is authorised to view according to the labels on the search results. This method requires that the server computes the results honestly so is only suitable for an honest or honest-but-curious adversarial model.

- **Hidden access levels**: The scheme is set up analogously to the single user SSE scheme in addition to a secret key being generated for each user (not necessarily distinct). The search results could be encrypted using CP-ABE and each user would have a CP-ABE secret key corresponding to their access level policy. Alternatively the search results can be encrypted using symmetric encryption with a different key per access level and a key assignment scheme could also be used so the user is able to derive all keys at lower access levels to enable them to decrypt all relevant search results. This could also be done without using a key assignment scheme, however if a search result is viewable by more than one access level then it would have to be copied and encrypted several times according to the access levels that are able to access it. The server will return all encrypted search results that satisfy the user's search query, however the user will only be able to decrypt search results corresponding to their access level. This will prevent any user that is not authorised to view a particular data item from seeing a search result relating to that data item.

Storage

The size of the indexes themselves is the same as the underlying SSE scheme however the data stored in the index might increase in size according to the type of encryption used.

Time

In the hidden access levels method more time will be required by the data owner to encrypt the search results and the search becomes a two step process for the user as they need to decrypt the search results they receive from the server.

Computation

There will be extra computation needed in order to encrypt and label the search results when compared with the costs of the underlying SSE scheme. In the case of hidden access levels, the user will also be required to decrypt the search results, which increases the computation required. If a key assignment scheme is used a user will be require to derive additional decryption keys from their own secret key in order to be able to view all relevant search results.

Leakage

In the hidden access levels method there is index information leakage as the server and the user see a full set of encrypted search results (as opposed to only seeing results corresponding to the user's access level). If the sets of encrypted search results are padded to be of equal size then a user will not be able to tell whether any results they cannot decrypt are beyond their access level or just padding. This requires extra computation in the index generation phase and higher communication costs when transferring the search results to the user, but will leak less information to the user regarding the size of the search results they are unauthorised to view. In the non-hidden access levels method there is also index information leakage as the server sees a full set of search results. The search pattern is revealed to the server in both cases.

6.3.3 Using a trusted third party (TTP)

- **Non-hidden access levels**: The sorting of the search results is done by the TTP (see the hidden access levels method) hence there is no benefit to revealing the access levels to the server.

- **Hidden access levels**: This is set up in the same way as the single user SSE scheme with the exception of a TTP that intercepts the search query and the search results between the user and the server. The secret key used to generate the index is distributed to each user and used to generate the search tokens. A user submits their search queries along with their access level to the TTP who keeps the access level information and transfers the search query to the server. The server computes the search results as per the single user SSE scheme and then sends the results to the TTP. The TTP holds information regarding the access levels of the search results and sorts them according to the user's access level. All search results that the user is authorised to view are then returned to the user.

Storage

No extra storage on the server is needed.

Time

The search time for the user will increase as the results need to filtered by the TTP before they can be returned to the user. Setting up the TTP will require extra time when initialising the scheme.

Computation

No extra computation is required for the data owner or server. Extra computation will be required to set up and maintain the TTP.

Leakage

There is index information leakage in the hidden access levels method as a full set of search results will be revealed to the server. The access levels and full search results are revealed to the TTP, however this entity is assumed to be fully trusted. The search pattern is also revealed to the server and the TTP.

6.3.4 Expanding the Keyword Dictionary

- **Non-hidden access levels**: The method described below, in the hidden access levels method, of incorporating the access level into the keywords themselves conceals the access level, hence this does not apply to the non-hidden access level setting.

- **Hidden access levels**: The set of keywords that can be queried over (keyword dictionary) in a single user SSE can be expanded to include a codeword per access level for each distinct keyword in the original dictionary. This could be done by concatenating the access level to the keyword and using this codeword in the search query, for example. The access level would form part of the user's secret key which would prevent users from searching at access levels other than their own. The index will contain the relevant codewords depending on the data item's access level, hence a data item will only satisfy a search query if the access level and the keyword match those of the search query. The search queries and index are generated using the same secret key which is distributed to all users. Each user also receives an access level which they can use to generate the codewords to compute the search queries, this should be kept secret to each user.

Storage

If a data item can be accessed by several access levels then the number of keywords used to index that data item will be increased by a factor of l, hence more storage will be required for the index on the server compared with the single user SSE scheme. If using an inverted index the index will have more entries as this type of index has an entry per keyword.

Time

The search time will not be affected by using more keywords, however the time needed to generate the index will increase by a factor of l.

Computation

Additional computation is needed to create the index due to the additional keywords that need to be indexed. The user will be required to compute the codeword for each search query by concatenating their access level with the search keyword before creating the search query.

Leakage

This method will achieve search pattern privacy between access levels as search queries for the same keyword at different access levels will be indistinguishable. However as in Section 6.3.1 the search results may reveal which search queries correspond to the same keyword. There is no other additional leakage.

6.4 Multi-level Searchable Symmetric Encryption

In this section we describe our system model and give a formal definition for MLSSE. Our new construction is based on the ground-breaking work of Curtmola et al. [55] that uses linked lists to store the identifiers of the data items in an array. Our scheme does not use an SSE scheme as a black box as in the generic solutions of Section 6.3. The index in this construction is an inverted index as it stores a linked list per keyword. Each linked list stores the identifiers of all data items that contain a specific keyword in an array where each node contains an identifier, a pointer to the next node in the linked list and a key that is used to decrypt the next node in the list. The server uses the search query along with a lookup table to locate the starting node of the linked list in the array and the key used to decrypt this node. This construction lends itself well to the multi-level access setting as the array does not need to be augmented. There only needs to be one linked list per keyword, as in the single user scenario, however during a search the lookup table points the server to the node in the linked list depending on the access level of the user that submitted the search query. For example a user at a higher access level will be pointed to a node nearer the start of the linked list, whereas a user at a lower access level will be pointed to a node further down the linked list. The nodes at the higher access level cannot be accessed by the server using a search query at the lower access level as the list can only be traversed in one direction.

6.4.1 System Model

We consider a system model comprising a *data owner* O, a remote *server* S, and a set of m data *users* $\mathcal{U} = (u_1, ..., u_m)$, where m is arbitrarily fixed. The data owner possesses a set of n data items $\mathcal{D} = (d_1, ..., d_n)$ which they wish to encrypt and outsource to S whilst authorising other users to search over some data items within $\mathcal{D}$. We present a *structure only* multi-level SE system model meaning we only consider the data structure (index) and we do not encrypt any associated data items. The encryption of the data items can be carried out separately using a suitable SKE or PKE scheme. The search results in our model consist of a set of data item identifiers which fulfil the query; these identifiers can be used to locate the encrypted data items themselves.

To support searching over the encrypted data items, O uploads some encrypted meta data to the server along with the ciphertexts. It defines a dictionary of keywords, denoted $\Delta = \{\omega_1, ..., \omega_{|\Delta|}\}$. Each data item can be associated with one or more keywords in Δ over which searches can be performed; we denote the plaintext set of keywords as $\delta_{\mathcal{D}} = (\delta_{d_1}, ..., \delta_{d_n})$ where $\delta_{d_i} \subseteq \Delta$ is the set of keywords associated with the data item d_i. The data owner produces an encrypted *index* $\mathcal{I}_{\mathcal{D}}$ based on $\delta_{\mathcal{D}}$, over which keyword searches are performed by S. Some schemes such as [55] use all distinct keywords within a word document as the set of keywords associated with the document, however we present a more generic scheme where the objects to encrypt and search over are not necessarily word documents but could be an image or video for example, we refer to these objects as *data items*. The set of keywords associated with each data item can be arbitrarily chosen by O and do not necessarily correspond to the actual words in a word document hence this minimises the risk of a statistical attack on the index as the frequency of a certain word in a document is not necessarily reflected in the set of keywords chosen to index the data item.

To prevent authorized users from receiving search results relating to *all* documents, O defines an information flow policy $\mathcal{P}$ with a labelling function λ mapping each user u and data item d to an access level, denoted $\lambda(u)$ and $\lambda(d)$ respectively, in the totally ordered set $\mathbb{L} = \{a_1, ..., a_l\}$. A user with clearance $\lambda(u_i)$ is authorised to search over a data item with classification $\lambda(d_j)$ if and only if $\lambda(d_j) \leq \lambda(u_i)$. Note that the access control in our model is enforced at data item level, meaning that users are restricted in which data items they are authorised to search over as opposed to restricting the keywords a user can search for [92]. This is an extension of the standard MSSE system model [55] where each user is able to search over and receive search results from the entire set of data items.

Each user that has been authorised by O can submit a *search query* to the server. All authorised users are added to the authorised user group $\mathcal{G}$, which is input into the key generation algorithm initially as an empty set. The group $\mathcal{G}$ forms part of the data owner's secret key. Note that O can also be enrolled as a 'user' in the system. Suppose user u_i with clearance $\lambda(u)$ wants to search for the keyword $\omega \in \Delta$; we denote the generated search query by $T_{\omega,\lambda(u)}$. Let us define $\mathcal{D}_\omega$ to be the set of identifiers of all data items that contain the keyword ω, and denote by $\mathcal{D}_{\omega,\lambda(u)} \subseteq \mathcal{D}_\omega$ the search results that u is authorised to view — that is, the set of identifiers of all data items id_d that contain ω and where $\lambda(d) \leq \lambda(u)$.

The addition and revocation of users in our model is handled using *broadcast encryption* (Definition 2.3.9). A user is only able to produce a valid search query if they are an authorized user in terms of the BE scheme.

To ease notation, let us define the tuple ${d_i}^{aug} = (d_i, id_i, \delta_{d_i}, \lambda(d_i))$ to completely describe a data item $d_i \in \mathcal{D}$ (being the data itself, the identifier, the associated keywords and the security classification). We denote the information regarding all data items by $\mathcal{D}^{aug} = \{{d_1}^{aug}, ..., {d_n}^{aug}\}$.

6.4.2 Formal Definition

Definition 6.4.1. *An MLSSE scheme consists of six algorithms defined as follows:*

- $(K_O, K_S, PP) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\kappa, S, \mathcal{P}, \mathcal{U}, \mathcal{G})$*: A probabilistic algorithm run by the data owner* O *that takes the security parameter* κ*, server identity* S *and information flow policy* $\mathcal{P}$*. It produces* O*'s secret key* K_O*, a server key* K_S *and public parameters* PP*.*

- $\mathcal{I}_\mathcal{D} \overset{\$}{\leftarrow} \mathsf{BuildIndex}(\mathcal{D}^{aug}, K_O, PP)$*: A probabilistic algorithm run by* O*. It takes the set* $\mathcal{D}^{aug}$ *and the data owner's secret key, and outputs the index* $\mathcal{I}_\mathcal{D}$*.*

- $K_{u_i} \overset{\$}{\leftarrow} \mathsf{AddUser}(u_i, \lambda(u_i), K_O, PP)$*: A probabilistic algorithm run by* O *to enrol a new user into the system. It takes the new user's identity and access level, and outputs a secret key for the new user.*

- $T_{\omega,\lambda(u_i)} \leftarrow \mathsf{Query}(\omega, K_{u_i})$*: A deterministic algorithm run by a user with clearance* $\lambda(u_i)$ *to generate a search query. It takes as input a keyword* $\omega \in \Omega$ *(where* Ω *is the set of keywords) and the user's secret key and outputs a search query* $T_{\omega,\lambda(u_i)}$*.*

- $\mathcal{R}_{\omega,\lambda(u_i)} \leftarrow \mathsf{Search}(T_{\omega,\lambda(u_i)}, \mathcal{I}_\mathcal{D}, K_S)$*: A deterministic algorithm run by* S *to search the index for data items containing a keyword* ω*. It takes as input the index and a search query submitted by a user and returns the search results* $\mathcal{R}_{\omega,\lambda(u_i)}$ *which consists of either a set of identifiers of ciphertexts* $\mathcal{D}_{\omega,\lambda(u_i)}$ *containing* ω *such that* $\lambda(d_j) \leq \lambda(u_i)$ *(where* $\lambda(u_i)$ *is the access level of the user that submitted the search query) or* $\perp$*.*

- $(K_O, K_S) \overset{\$}{\leftarrow} \mathsf{RevokeUser}(u_i, K_O, PP)$*: A probabilistic algorithm run by* O *to revoke a user from the system. It takes the user's id, the data owner's and server's secret keys, and outputs updated owner and server keys.*

An MLSSE scheme is correct if for all $\kappa \in \mathbb{N}$*, for all policies in* $\mathcal{P}$*, for all* K_O, K_S *output by* $\mathsf{KeyGen}(1^\kappa, \mathcal{P})$*, for all* $\mathcal{D}^{aug}$*, for all* $\mathcal{I}_\mathcal{D}$ *output by* $\mathsf{Buildindex}(\mathcal{D}^{aug}, K_O)$*, for all* $\omega \in \Delta$*, for all* $u \in \mathcal{U}$*, for all* K_u *output by* $\mathsf{AddUser}(K_O, u, \lambda(u), PP)$*, we have that* $Search(\mathcal{I}_\mathcal{D}, T_{\omega,\lambda(u)}) = \mathcal{D}_{\omega,\lambda(u)}$*.*

If $\mathcal{G} = \emptyset$ *along with the restrictions above then we have correctness if:*

$$Search(\mathcal{I}_\mathcal{D}, T_{\omega,\lambda(u_m)}) = \perp .$$

6.4.3 Security Model

In terms of security for SSE schemes it is ideally required that given the outsourced metadata $\mathcal{I}_{\mathcal{D}}$, an adversary playing the role of the server cannot learn any information regarding the data set $\mathcal{D}$ (i.e. a curious server cannot learn information about the data it stores). However, most SSE schemes in the literature leak some information in a trade off for increased efficiency. For example, for a set of search queries $\{T_1, ..., T_q\}$, the outcomes of these searches $\{R_1, ..., R_q\}$ could be leaked to an adversary, which is the server in our case; this is referred to as the *access pattern* (Definition 3.2.1). The access pattern defines the link between a search query and the search results it produces and can be thought of as a database where each row stores two values: a search query and a corresponding data item identifier of a data item that satisfies the search query.

As well as the access pattern most efficient SSE schemes also leak the *search pattern* (Definition 3.2.2) which reveals repetitions in the set of search queries made to the server. In most single user SSE schemes, [46, 47, 55, 68, 85, 86], search queries are formed deterministically which enables the server to ascertain whether a particular search query has been used previously.

In our scheme search queries for the same keyword that are produced by users with different access levels are indistinguishable from one another. That is, a search query for a keyword ω from a user u_i with access level $\lambda(u_i)$ is indistinguishable from a search query for ω from a user u_j with access level $\lambda(u_j)$ for $\lambda(u_i) \neq \lambda(u_j)$. This means that from the queries alone an adversary is unable to deduce how many times a certain keyword has been searched for overall, it can only deduce how many times the same keyword has been searched for within each access level. We alter the definition of search pattern to reflect this:

Definition 6.4.2. *(Search pattern (multi-level access)) For a sequence of q search queries $\{T_{\omega_1,\lambda(u_1)}, ..., T_{\omega_1,\lambda(u_q)}\}$ where ω_i and ω_j or $\lambda(u_i)$ and $\lambda(u_j)$ are not necessarily distinct for $i \neq j$, and $\lambda(u_i)$ is the access level of the user submitting the ith query, along with an index $\mathcal{I}_{\mathcal{D}}$, the search pattern is defined as a $q \times q$ symmetric binary matrix $SP(\mathcal{I}_{\mathcal{D}}, q)$ such that for $1 \leq i, j \leq q$:*

$$SP(\mathcal{I}_{\mathcal{D}}, q)_{i,j} = 1 \iff T_{\omega_i,\lambda(u_i)} = T_{\omega_j,\lambda(u_j)}.$$

Intuitively, the search pattern reveals when the ith and jth queries are the same, which happens when queries are issued for the same keyword by users with the same access level.

In terms of access pattern we also reduce the amount of information leakage compared with standard single user or multi-user SSE schemes such as [46, 47, 55, 68, 85, 86]. In particular we do not reveal whether a data item contains the keyword ω_i associated with a search query unless the access level of that data item is less than or equal to that of the user u_i that generated the search query, meaning that an adversary cannot see a full set of search results. We alter the definition of access pattern to reflect this:

Definition 6.4.3. *(Access pattern (multi-level access)) For a sequence of q search queries $\{T_{\omega_1,\lambda(u_1)}, ..., T_{\omega_q,\lambda(u_q)}\}$ where ω_i and ω_j or a_i and a_j are not necessarily distinct for $i \neq j$, and $\lambda(u_i)$ is the access level of the user submitting the ith query, the access pattern is:*

$$AP = \{\mathcal{R}_1, ..., \mathcal{R}_q\}.$$

Where $\mathcal{R}_i = \perp$ if $\lambda(u_i) < \lambda(d_j)$ for all $d_j \in \mathcal{D}$, or $\mathcal{R}_i = \mathcal{R}_{\omega_i,\lambda(u_i)}$ otherwise. The access pattern is the set of search results produced by the sequence of q queries.

However when a search query is paired with the search results it generates (the access pattern) then an adversary may be able to correlate which search queries are for the same keyword by looking at the intersections of the search results. For example if one set of search results is a subset of another set of search results then this may imply that the two search queries used to generate these results are for the same keyword. An adversary may eventually be able to build up a complete set of search results for a particular keyword, which is equivalent to the leakage produced by a search query in a single user SSE scheme.

The hierarchal relationships between the data item identifiers i.e. which identifiers represent data items at higher access level than others could also be leaked in the same way. If an adversary has ascertained that two sets of search results $\mathcal{R}_{\omega,a_i} \subset \mathcal{R}_{\omega,a_j}$ represent searches for the same keyword ω, then an adversary will be able to conclude that identifiers in the set $\mathcal{R}_{\omega,a_j} \setminus \mathcal{R}_{\omega,a_i}$ are at a higher access level than those in $\mathcal{R}_{\omega,a_i}$.

We note that unless the search results are padded in some way this leakage is inevitable. Padding search results is not standard in SSE schemes as it requires post-processing of the search results by the user. Hence we do not pad the search results in our system model in order to maintain an efficient scheme.

From this we can see that initially our scheme leaks less information about the search pattern and access pattern than a single user SSE scheme, however over time as more queries are generated the information leakage tends to that of a single user SSE scheme. The information leakage relating to a keyword ω, i.e. the access patterns for search queries corresponding to ω, only reaches that of a single user SSE scheme once a search query has been generated at each possible access level, and our leakage remains lower up until this point.

Leakage functions

In MLSSE we also need to consider the leakage of information from data items during a search that do not correspond to the user's access level associated with the search query, in particular we do not reveal whether a data item contains the keyword associated with a search query unless the access level of that data item is less than or equal to that of the user that generated the search query. We define the following two leakage functions to precisely capture the leakage of information in MLSSE, from the output of the setup phase (KeyGen and BuildIndex) and the search phase (Query and Search):

1. $\mathcal{L}^{MLSSE}_{Setup}(\mathcal{I}_{\mathcal{D}}) = \{|\mathcal{I}_{\mathcal{D}}|, id(\omega, a)\}$
2. $\mathcal{L}^{MLSSE}_{Search}(\mathcal{I}_{\mathcal{D}}, T_{\omega_i,\lambda(u_i)}) = \{id(\omega_i, \lambda(u_i)), AP(\mathcal{I}_{\mathcal{D}}, T_{\omega_i,\lambda(u_i)}, SP(\mathcal{I}_{\mathcal{D}}, T_{\omega_i,\lambda(u_i)})\}$

During the setup phase the only information we expect to be leaked is the size of the index and a *codeword* for each keyword and access level pair, which we denote $id(\omega, a)$. This codeword is a ciphertext that is unique for each each keyword and access level pair. During the search phase we have additional leakage in the form of the access pattern and search pattern (Definition 6.4.3 and Definition 6.4.2 respectively) along with the codeword associated with each search query.

We define an additional notion of search leakage called the *maximal search leakage* with respect to an index $\mathcal{I}_{\mathcal{D}}$, which we use to define the notion of multi-level security. This is the leakage resulting from every possible keyword search on $\mathcal{I}_{\mathcal{D}}$ with a given access level λ^{max}, as follows:

Definition 6.4.4 (Maximal search leakage)**.** *Consider an access level λ^{max}. The maximal search leakage on an index $\mathcal{I}_{\mathcal{D}}$ associated with access level λ^{max} is:*

$$\mathcal{L}^{\lambda^{max}}_{Search}(\mathcal{I}_{\mathcal{D}}) = \mathcal{L}_{Search}(\mathcal{I}_{\mathcal{D}}, \{T_{\omega_i,\lambda_{max}}\}_{\forall \omega_i \in \Delta}).$$

We now formalise the notions of security we require in MLSSE. We consider security against chosen keyword attacks and secure revocation of users such that the ability to perform searches can be removed. We use cryptographic games to formalise our notions of security. The adversary $\mathcal{A}$ for each game is run by a challenger $\mathcal{C}$. The adversary is modelled as a probabilistic polynomial time (PPT) entity whose inputs are chosen to reflect the information available to a realistic adversary, this information is defined by the leakage functions.

The multi-level access game (Game 6.1) and the revocation game (Game 6.2) have the following inputs: the multi-level access scheme defined in Definition 6.4.1, which we denote by MLSSE along with a set of other inputs: the security parameter 1^κ, the universe of user identities $\mathcal{U}$, the access policy $\mathcal{P}$, the function λ and Δ (all as defined in Section 6.4.1). We denote this set of inputs as X. The set $\mathcal{D}^{aug}$ represents the set of meta-data used to construct the index as defined in Section 6.4.1 and $\mathcal{L}_{Search}^{\lambda^{max}}(\mathcal{I}_{\mathcal{D}})$ represents the maximal search leakage as defined in Definition 6.4.4.

Multi-level Access

Our first security notion, in Game 6.1, is that of *multi-level access* which requires that a user, u, cannot receive search results or learn information relating to data items d_i such that $\lambda(u) < \lambda(d_i)$. More specifically, a server colluding with several users cannot learn anything about the index beyond the specified leakage according to the corrupt users' access rights. This notion is similar to the benchmark notion of security called IND2-CKA which was established by Curtmola et al. in [55].

The challenger sets up the system, including instantiating several global variables (which the challenger can use in the main game and in oracle functions, but which the adversary cannot see): λ^{max} is the maximum access level of any corrupted user and chall is a Boolean flag to show whether the challenge parameters have been generated yet. The adversary is given the security parameter, access control policy, server key and the public parameters, as well as access to the following oracles, where the parameters represented by $\cdot$ are provided by the adversary, and the adversary is given the resulting user keys and search results:

- The $\textsc{AddUser}(\cdot, \lambda(\cdot), K_O, PP)$ oracle allows the adversary to request that a user is added to the system, and the adversary corrupts this user by receiving the user key. If the challenge has not yet been generated, then the challenger adds the requested user, checks if λ^{max} needs updating and runs the $\mathsf{AddUser}$ algorithm. Otherwise, if the challenge has been generated, the above procedure is carried out only if the new user's access level is less than or equal to λ^{max}. This oracle is described in Game 6.1.

- The $\textsc{RevokeUser}(\cdot, K_O, PP)$ oracle allows the adversary to revoke a user from the system. The oracle simply runs the $\mathsf{RevokeUser}$ algorithm on the inputs supplied by the adversary. We note that querying $\mathsf{RevokeUser}$ could potentially alter the value of λ^{max} in Game 6.1. We do not consider this option as revoking a user would reduce the amount of access that the adversary has, hence a rational adversary would have no interest in doing this.

- The $\textsc{BuildIndex}(\cdot, K_O, PP)$ oracle allows the adversary to create indexes of their choosing. It simply runs $\mathsf{BuildIndex}$ on the adversary's input and returns the output to the adversary.

We denote the adversary $\mathcal{A}$ as having access to oracles $\textsc{AddUser}(\cdot, \lambda(\cdot), K_O, PP)$, $\textsc{RevokeUser}(\cdot, K_O, PP)$ and $\textsc{BuildIndex}(\cdot, K_O, PP)$ as $\mathcal{A}^{\mathcal{O}}$.

After a polynomial number of queries, the adversary outputs two data sets $(\mathcal{D}_0^{aug}, \mathcal{D}_1^{aug})$ which must have identical maximal search leakage for the maximal access level, λ^{max}, of any corrupted user — the adversary cannot choose data sets where a user that it has corrupted could make any query that legitimately distinguishes the data sets as this would count as a trivial win. Whilst this may appear to be a strong assumption, we believe it to be the minimal assumption necessary to avoid trivial wins in the multi-user setting. The main issue is that, unlike Curtmola et al. [55], in the multi-user setting, it is necessary to consider the server colluding with a set of users (but not the data owner); as such, the adversary is able to perform the roles of the server and of an authorised user and therefore may produce arbitrary queries and perform searches himself. Thus, the challenger in the game is unable to monitor which searches have been performed and hence cannot determine whether the traces of the *actual* queries on both data sets are equal, and instead must rely on the stronger assumption that no possible authorised query can distinguish the data sets. Note that Van Rompay et al. [112] deal with the multi-user case without this assumption since they deal with single word indexes and have a proxy through which all queries are made.

The challenger sets the challenge flag to `true` and chooses a random bit b which determines the data set used to form an index. The adversary is given the index and oracle access $\mathcal{A}^{\mathcal{O}}$ as defined above and must determine which data set was used.

Definition 6.4.5. *(Multi-level Access (MLA)) Let* MLSSE *be a multi-level searchable symmetric encryption scheme where* $\kappa \in \mathbb{N}$ *is the security parameter, for an information flow policy* $\mathcal{P}$*, and* $\mathcal{A}$ *a PPT adversary. The advantage of* $\mathcal{A}$ *is:*

$$Adv_{\mathcal{A}}^{\mathbf{MLA}}(\mathsf{MLSSE}, X) = |\mathbb{P}[\mathbf{Exp}_{\mathcal{A}}^{\mathbf{MLA}}[\mathsf{MLSSE}, X] = 1] - \frac{1}{2}|,$$

where $\boldsymbol{Exp}_{\mathcal{A}}^{\mathbf{MLA}}[X]$ *is defined in Game 6.1. We say that* MLSSE *is secure against non-adaptive chosen keyword attacks in the sense of Game 6.1 if for all* $\mathcal{A}$*, all* $k \in \mathbb{N}$ *and all* $\mathcal{P}$*:*

$$Adv_{\mathcal{A}}^{\mathbf{MLA}}(\mathsf{MLSSE}, X) \leq \text{negl}(k),$$

for a negligible function negl.

Revocation

In MLSSE, as with other multi-user SSE schemes, we need to consider user *revocation* to remove a user's ability to submit valid search queries to the server, and hence receive

```
$\mathbf{Exp}^{\mathbf{MLA}}_{\mathcal{A}}[\mathsf{MLSSE}, X]$:
1 :  $\lambda^{max} \leftarrow \perp$
2 :  $\mathsf{chall} \leftarrow \mathbf{false}$
3 :  $(K_O, K_S, PP) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\kappa, S, \mathcal{P}, \mathcal{U}, \mathbb{L})$
4 :  $(\mathcal{D}_0^{aug}, \mathcal{D}_1^{aug}, st) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(X, K_S, PP)$
5 :  $\mathcal{I}_{\mathcal{D}_0} \leftarrow\!\!\$\ \mathsf{BuildIndex}(\mathcal{D}_0^{aug}, K_O, PP)$
6 :  $\mathcal{I}_{\mathcal{D}_1} \leftarrow\!\!\$\ \mathsf{BuildIndex}(\mathcal{D}_1^{aug}, K_O, PP)$
7 :  if $\mathcal{L}^{\lambda^{max}}_{Search}(\mathcal{I}_{\mathcal{D}_0}) \neq \mathcal{L}^{\lambda^{max}}_{Search}(\mathcal{I}_{\mathcal{D}_1})$
8 :    return $0$
9 :  $\mathsf{chall} \leftarrow \mathbf{true}$
10 :  $b \leftarrow\!\!\$\ \{0,1\}$
11 :  $b' \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(\mathcal{I}_{\mathcal{D}_b}, st)$
12 :  if $b' = b$ return $1$
13 :  else return $0$

Oracle $\textsc{AddUser}(u, \lambda(u), K_O, PP)$
1 :  if $\mathsf{chall} = \mathbf{false}$
2 :    if $\lambda(u) > \lambda^{max}$
3 :      $\lambda^{max} \leftarrow \lambda(u)$
4 :    return $\mathsf{AddUser}(u, \lambda(u), K_O, PP)$
5 :  else if $\lambda(u) > \lambda_{max}$
6 :    return $\perp$
7 :  else return $\mathsf{AddUser}(u, \lambda(u), K_O, PP)$
```

Game 6.1: Adaptive multi-level access game for MLSSE

search results. We capture this in Game 6.2. The challenger sets up the system, by running KeyGen to produce the data owner's key, the server key and the public parameters. The adversary is given the input X along with the public parameters and the server key K_S and selects a data set (along with associated access levels, keywords and identifiers). The challenger then creates the index which is transferred to the adversary. The adversary is given access to the following oracles where the parameters represented by $\cdot$ are provided by the adversary, and the adversary is given the resulting user keys and search results.

- The $\textsc{AddUser}(\cdot, \lambda(\cdot), K_O, PP)$ oracle allows the adversary to add a user to the system. It runs AddUser on the adversary's inputs u, $\lambda(u)$ and returns either the output of $\mathsf{AddUser}(u, \lambda(U), K_O, PP)$ if user u is not already an authorised user or $\perp$ if the user is already authorised. The oracle is run as depicted in Game 6.2.

- The $\textsc{Search}(\cdot, \mathcal{I}_{\mathcal{D}}, K_S)$ oracle allows the adversary to search the index for a particular keyword. The adversary computes the search query for the particular keyword using Query and sends it to the oracle as input.

- The $\textsc{RevokeUser}(\cdot, K_O, PP)$ allows the adversary to revoke a user from the system. The oracle runs the RevokeUser algorithm on the input u supplied by the adversary if u is an authorised user; otherwise it outputs $\perp$. The oracle is run as depicted in Game 6.2.

- The $\textsc{BuildIndex}(\cdot, K_O, PP)$ oracle allows the adversary to create indexes of their choosing. It simply runs BuildIndex on the adversary's input and returns the output to the adversary.

```
$\mathbf{Exp}^{\mathbf{Revoke}}_{\mathcal{A}}[\mathsf{MLSSE}, X]$:
1 :  $(K_O, K_S, PP) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^k, S, \mathcal{P}, \mathcal{U}, \mathbb{L})$
2 :  $(\mathcal{D}^{aug}, st) \leftarrow \mathcal{A}^{\mathcal{O}}(X, PP)$
3 :  $\mathcal{I}_{\mathcal{D}} \leftarrow\!\!\$\ \mathsf{BuildIndex}(\mathcal{D}^{aug}, K_O, PP)$
4 :  $st \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(st)$
5 :  for $u \in \mathcal{G}$
6 :    $(K_O, PP) \leftarrow\!\!\$\ \mathsf{RevokeUser}(u, K_O, PP)$
7 :  $T_\omega \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}}(st, PP)$
8 :  $\mathcal{R} \leftarrow \mathsf{Search}(T_\omega, \mathcal{I}_{\mathcal{D}}, K_S)$
9 :  if $\mathcal{R} \neq \perp$
10 :    return 1
11 :  else
12 :    return 0
```

```
Oracle AddUser$(u, \lambda(u), K_O, PP)$
1 :  if $u \in \mathcal{G}$
2 :    return $\perp$
3 :  else
4 :    $\mathcal{G} \leftarrow \mathcal{G} \cup u$
5 :    return
6 :      $\mathsf{AddUser}(u, \lambda(u), K_O, PP)$
```

```
Oracle RevokeUser$(u, K_O, PP)$
1 :  if $u \in \mathcal{G}$
2 :    $\mathcal{G} \leftarrow \mathcal{G} \backslash u$
3 :    return
4 :      $\mathsf{RevokeUser}(u, K_O, PP)$
5 :  else
6 :    if $u \notin \mathcal{G}$
7 :    return
8 :      $\perp$
```

Game 6.2: Adaptive revocation game for MLSSE

We denote the adversary $\mathcal{A}$ as having access to oracles AddUser$(\cdot, \lambda(\cdot, K_O, PP)$, RevokeUser$(\cdot, K_O, PP)$, Search$(\cdot, \mathcal{I}_{\mathcal{D}}, K_S)$ and BuildIndex$(\cdot, K_O, PP)$ as $\mathcal{A}^{\mathcal{O}}$. After a polynomial number of queries, the challenger revokes all users that were queried to the AddUser$(\cdot, \lambda(\cdot), K_O, PP)$ oracle but were not subsequently queried to the $\mathsf{RevokeUser}(\cdot, K_O, PP)$ oracle (i.e. all users for which the adversary holds a valid user key). The adversary must then produce a search token T_ω which, when used as input to the Search algorithm, does not produce $\perp$, i.e. the adversary must produce a valid search query even though it does not hold a non-revoked key.

Definition 6.4.6. *(Revocation) Let* MLSSE *be a multi-level searchable symmetric encryption scheme where $k \in \mathbb{N}$ is the security parameter, for an information flow policy $\mathcal{P}$, and $\mathcal{A}$ a PPT adversary. We define the advantage of $\mathcal{A}$ in Game 6.2 as:*

$$Adv^{\mathbf{Revoke}}_{\mathcal{A}}(\mathsf{MLSSE}, X) = \left|\mathbb{P}[\mathbf{Exp}^{\mathbf{Revoke}}_{\mathcal{A}}[\mathsf{MLSSE}, X] = 1]\right|.$$

We say that MLSSE *achieves revocation if for all $\mathcal{A}$, all $k \in \mathbb{N}$ and all $\mathcal{P}$:*

$$Adv^{Revoke}_{\mathcal{A}}(X) \leq \text{negl}(k).$$

6.5 Construction

Our construction MLSSE is an adaptation of the scheme of Kamara et al. [86], which is based on the construction of the influential inverted index scheme $\mathsf{SSE\text{-}1}$ by Curtmola et al. [55].

Informally, MLSSE uses an array $\mathbb{A}$ of linked lists, along with a look-up table $\mathbb{T}$ to index the encrypted data. This produces a sequential search that lends itself well to the hierarchical access rights on the data items that we require. For each keyword $\omega_i \in \Delta$, we define a list L_{ω_i} which stores the identifiers for all data items containing that keyword and is ordered according to the access level of the data items — data items with the highest classification are placed at the beginning of the list, and those with the lowest classification at the end. Each list L_{ω_i} is encrypted and stored in $\mathbb{A}$ as a linked list. During the search phase the look-up table $\mathbb{T}$ is used to point the server to the correct node in the array depending on the information in the search query i.e. which keyword was searched for and what access rights the user that submitted the search query has. This node is decrypted using information in the search query and the node itself, revealing the address of the next node in the linked list. The server may continue to decrypt all other relevant nodes in the linked list, obtaining the set of search results relevant to the user's searched keyword and access level.

The key difference between our scheme and that of [86] is that, rather than pointing to the *beginning* of each linked list, the entry in $\mathbb{T}$ will point to the appropriate position within the linked list according to the access rights of the querier (recall that the list is ordered by access levels). Since it is not possible to move backwards through the encrypted lists, as only a pointer and decryption key are provided for the next node in the current node, the only search results available are those contained beyond this point in this list — that is, identifiers for those documents containing the keyword and whose classification is at most that of the querier, as required by the information flow policy.

Let BE be an IND-CPA secure broadcast encryption scheme. We define the following pseudorandom functions (PRFs), where the keys are represented by the first entry:

$$F : \{0,1\}^{\kappa} \times \{0,1\}^{*} \to \{0,1\}^{k},$$

$$G : \{0,1\}^{\kappa} \times \{0,1\}^{*} \to \{0,1\}^{*},$$

$$P : \{0,1\}^{\kappa} \times \{0,1\}^{*} \to \{0,1\}^{k},$$

$$H : \{0,1\}^{\kappa} \times \{0,1\}^{*} \to \{0,1\}^{*},$$

and a strong pseudorandom permutation (PRP), where the key is the first entry:

$$\phi : \{0,1\}^\kappa \times \{0,1\}^\kappa \times \{0,1\}^* \times \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa \times \{0,1\}^* \times \{0,1\}^\kappa,$$

$\mathbb{A}$ is a $|\Delta| \times |\mathbb{L}|$ array and $\mathbb{T}$ is a dictionary of size $|\Delta| \cdot |\mathbb{L}|$. We denote the address of a node N in $\mathbb{A}$ as $addr_{\mathbb{A}}(N)$.

Let $\lambda : \{\mathcal{U} \cup \mathcal{D}\} \to \mathbb{L}$ map users and data items to their relevant access levels as described in Section 6.4.1. We define a function γ which outputs three ordered lists $\mathsf{L}_{\omega_i}, \mathsf{X}_{\omega_i}$ and N_{ω_i} given the set of identifiers $\mathcal{D}^{aug}$ and the array $\mathbb{A}$. We refer to the n^{th} item in a list L_{ω_i} as $\mathsf{L}_{\omega_i}[n]$. The list L_{ω_i} contains identifiers of data items in $\mathcal{D}_{\omega_i}$ ordered from the identifiers with the highest to the lowest access levels, the list N_{ω_i} contains the addresses of $|\mathsf{L}_{\omega_i}|$ empty nodes chosen randomly from $\mathbb{A}$ and the list X_{ω_i} contains the indices of the identifiers in L_{ω_i} where each access level starts. For example, suppose we have an ordered list of identifiers $\mathsf{L}_{\omega_i} = (id_1, id_2, id_3, id_4, id_5)$ where:

$$a_1 = \lambda(id_1) = \lambda(id_2) = \lambda(id_3) > \lambda(id_4) = \lambda(id_5) = a_3.$$

Then we have that $\mathsf{X}_{\omega_i}[3] = 4$, which says that the list of nodes with access level at most a_3 starts at the fourth entry in L_{ω_i}. There is an entry per each access level in X_{ω_i}, even if two access levels have the same starting point in L_{ω_i}; from the example above we can see that $\mathsf{X}_{\omega_i}[2] = \mathsf{X}_{\omega_i}[3] = 4$. If an access level is not authorised to view any data items in $\mathcal{D}_{\omega_i}$ then the entry corresponding to that access level (as well as the entries corresponding to all access levels below it) in X_{ω_i} is set to $\perp$. An identifier of a data item $d_i \in \mathcal{D}_{\omega_i}$ will inherit the access level label of the respective data item, i.e. $\lambda(id_{d_i}) = \lambda(d_i)$.

The KeyGen algorithm of MLSSE initialises the system and generates the keys K_O, K_S, along with the public parameters, PP. The key K_O includes the secret key for the BE scheme and the sets of $|\mathbb{L}|$ keys for each pseudo-random function: F, G and P and the key for the pseudo-random permutation ϕ (referred to as the data owner's state, st_O) as well as the authorized user group $\mathcal{G}$. The server is enrolled and its secret key is also generated. PP includes the information flow policy $\mathcal{P}$and the public parameters for BE, PP_{BE}.

The $\mathsf{BuildIndex}$ algorithm initializes a set $free$ which consists of all nodes in the array $\mathbb{A}$. $\mathsf{BuildIndex}$ considers each keyword contained in the dataset in turn. For each keyword ω_i, the function γ generates $\mathsf{L}_{\omega_i}, \mathsf{X}_{\omega_i}$ and N_{ω_i}. The free list is then updated according to which nodes have been chosen by γ. The nodes in the array that form the linked lists consist of the identifier from L_{ω_i} of a data item containing ω_i, the address

```
$(K_O, K_S, PP) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^k, S, \mathcal{P}, \mathcal{U}, \mathbb{L}, \mathcal{G})$

**for** $i \in |\mathbb{L}|$
    $k_{a_i,1}, k_{a_i,2}, k_{a_i,3} \leftarrow\!\!\$\ \{0,1\}^k$
$(\mathrm{PP}_{\mathsf{BE}}, k_{\mathsf{BE}}) \leftarrow\!\!\$\ \mathsf{BE.KeyGen}(1^{\mathsf{k}}, |\mathcal{U}|)$
$st_O \leftarrow\!\!\$\ \{0,1\}^k$
$k_S \leftarrow\!\!\$\ \mathsf{BE.Add}(k_{\mathsf{BE}}, S)$
$\mathcal{G} \leftarrow \{S\}$
$st_S \leftarrow\!\!\$\ \mathsf{BE.Enc}(st_O, \mathcal{G}, k_{\mathsf{BE}})$
**return**
    $K_S \leftarrow (k_s, st_S)$
    $K_O \leftarrow (\{k_{a_i,1}\}_{i \in [|\mathbb{L}|]}, \{k_{a_i,2}\}_{i \in [|\mathbb{L}|]}, \{k_{a_i,3}\}_{i \in [|\mathbb{L}|]}, k_{\mathsf{BE}}, st_O, \mathcal{G})$
    $PP \leftarrow (\mathcal{P}, PP_{\mathsf{BE}})$
```

Algorithm 6.1: KeyGen algorithm for MLSSE

```
$\mathcal{I}_{\mathcal{D}} \leftarrow\!\!\$\ \mathsf{BuildIndex}(\mathcal{D}_{aug}, K_O, PP)$

$free \leftarrow \{addr(N_i)\}_{[i \in |\mathbb{A}|]}$
**for** $1 \leq i \leq |\mathcal{W}|$
    $(\mathsf{L}_{\omega_i}, \mathsf{X}_{\omega_i}, \mathsf{N}_{\omega_i}) \leftarrow \gamma(\mathcal{D}_{\omega_i})$
    $free \leftarrow free \setminus \mathsf{N}_{\omega_i}$
    **for** $1 \leq j \leq |\mathsf{N}_{\omega_i}| - 1$
        $r_j \leftarrow\!\!\$\ \{0,1\}^k$
        $\mathbb{A}[\mathsf{N}_{\omega_i}[j]] \leftarrow \Big(\big(\mathsf{L}_{\omega_i}[j], \mathsf{N}_{\omega_i}[j+1], P_{k_{\lambda(\mathsf{L}_{\omega_i}[j+1]),3}}(\omega_i)\big) \oplus H\big(P_{k_{\lambda(\mathsf{L}_{\omega_i}[j]),3}}(\omega_i), r_j\big), r_j\Big)$
    $r_{|\mathsf{N}_{\omega_i}|} \leftarrow\!\!\$\ \{0,1\}^k$
    $\mathbb{A}[\mathsf{N}_{\omega_i}[|\mathsf{N}_{\omega_i}|]] \leftarrow \Big(\big(\mathsf{L}_{\omega_i}[|\mathsf{N}_{\omega_i}|], 0, 0\big) \oplus H\big(P_{k_{\lambda(\mathsf{L}_{\omega_i}[|\mathsf{N}_{\omega_i}|]),3}}(\omega_i), r_|\mathsf{N}_{\omega_i}|\big), r_{|\mathsf{N}_{\omega_i}|}\Big)$
     **for** $1 \leqslant \ell \leqslant |\mathbb{L}|$
        **if** $\mathsf{X}_{\omega_i}[a_\ell] \neq \perp$
            $\mathbb{T}[F_{k_{a_\ell,1}}(\omega_i)] \leftarrow \big(\mathsf{N}_{\omega_i}\left[\mathsf{X}_{\omega_i}[a_\ell]\right] \oplus G_{k_{a_\ell,2}}(\omega_i)\big)$
        **else**
            $\mathbb{T}[F_{k_{a_\ell,1}}(\omega_i)] \leftarrow \perp$
 **return**
    $\mathcal{I}_{\mathcal{D}} \leftarrow (\mathbb{A}, \mathbb{T})$
```

Algorithm 6.2: BuildIndex algorithm for MLSSE

in the array of the next node in the linked list, the key used to decrypt the following node in the linked list and a random bit string $r_i \in \{0,1\}^{\kappa}$. The identifier, address of the next node and the key used to decrypt the following node in the linked list are XORed with the output of a PRF H in order to encrypt this information. For the first node in the linked list the input of H is the decryption key for the current node (which corresponds to an access level and keyword and forms part of the search query) along with r_i), hence the information stored in the node can only be decrypted by the server if the server has a search query generated by a user who is authorized to view the data item whose identifier is stored at that node. The decryption key for all subsequent nodes is contained in the previous node of the linked list. BuildIndex then proceeds to create the look-up table $\mathbb{T}$. Unlike prior schemes [55], each user may have a different access level and thus the starting points for search results within the linked lists may vary; a search query made by a user with a higher access level should traverse more of the list than that of a user with lower access rights (the user is authorised to search more data items). Table $\mathbb{T}$ has an entry for each access level/keyword pair containing the address of a node in $\mathbb{A}$, which is the node in the linked list L_{ω_i} from which the user with a specified access level is authorised to decrypt. If an access level is not authorised to view any part of the linked list then the value in $\mathbb{T}$ is set to $\perp$. Finally the index $\mathcal{I}_{\mathcal{D}} = (\mathbb{A}, \mathbb{T})$ is returned.

```
$T_{\omega,\lambda(u)} \leftarrow \mathsf{Query}(\omega, K_u)$

1 : Retrieve $st_S$ from server
2 : $st'_O \leftarrow \mathsf{BE.Dec}(k_u, st_S)$
3 : if $st'_O = \perp$
4 :     return $\perp$
5 : $t_{\omega,\lambda(u)} \leftarrow (F_{k_{\lambda(u),1}}(\omega), G_{k_{\lambda(u),2}}(\omega), P_{k_{\lambda(u),3}}(\omega), 0^{32})$
6 : return
7 :     $T_{\omega,\lambda(u)} \leftarrow \phi_{st'_O}(t_{\omega,\lambda(u)})$
```

Algorithm 6.3: Query generation algorithm for MLSSE

The Query algorithm for MLSSE generates a search query for a user u to search for a keyword w. The user first retrieves the current server state, st_S, from the server and attempts to decrypt the current server state st_S using their secret key k_u; we denote the output of the decryption by st'_O. Note that if u is not authorised then decryption will return $\perp$, if this is the case Query outputs $\perp$. The query itself comprises four parts. The first is the output of the PRF F applied to the keyword ω, keyed with the secret key for F associated with the user's access level $k_{\lambda(u),1}$. This part of the query is used to locate the relevant entry in $\mathbb{T}$. The second part is the output of the PRF G applied to the keyword ω and is used to mask the entry in $\mathbb{T}$ in order to locate the user's relevant starting position in the linked list corresponding to ω in $\mathbb{A}$. The third part is the output of the PRF P applied to the keyword ω, which is used to decrypt the first relevant node in $\mathbb{A}$ according to the user's access level. The fourth part of the query is composed of κ 0-bits, this is to ensure that under valid inputs the set of possible outputs of the PRP is always sparse. The PRP ϕ is applied to the search query, using st'_O as the key.

```
$\mathcal{R}_{\omega,\lambda(u)} \leftarrow \mathsf{Search}(T_{\omega,\lambda(u)}, I_{\mathcal{D}}, K_S)$

1 :  $st'_O \leftarrow \mathsf{BE.Dec}(K_S, st_S)$
2 :  Parse $\phi^{-1}_{st'_O}(T_{\omega,\lambda(u)})$ as $(\tau_1, \tau_2, \tau_3, \tau_4)$
3 :  **if** $\tau_4 \neq 0$
4 :    **return** $\perp$
5 :  $\mathcal{R}_{\omega,\lambda(u)} \leftarrow \emptyset$
6 :  **if** $\mathbb{T}[\tau_1] = \perp$
7 :    **return** $\perp$
8 :  $v_2 \leftarrow 1$
9 :  **while** $v_2 \neq 0$
10 :     Parse $\mathbb{T}[\tau_1] \oplus \tau_2$ as $y$
11 :     Parse $\mathbb{A}[y]$ as $(z_1, z_2)$
12 :     Parse $z_1 \oplus H(\tau_3, z_2)$ as $(v_1, v_2)$
13 :     $\mathcal{R}_{\omega,\lambda(u)} \leftarrow \mathcal{R}_{\omega,\lambda(u)} \cup \{v_1\}$
14 :  **return** $\mathcal{R}_{\omega,\lambda(u)}$
```

Algorithm 6.4: Search algorithm for MLSSE

The Search algorithm finds data item identifiers associated with the searched keyword from the subset of data item identifiers the user is authorized to search. The server decrypts st_S and applies the inverse of the PRP ϕ to the query it received; it parses the result as $(\tau_1, \tau_2, \tau_3, \tau_4)$. The server then looks up entry $\mathbb{T}[\pi_1]$ and if that entry is not equal to $\perp$, the server XORs the value with π_2 and parses the resulting value as y. The server looks up the node at $\mathbb{A}[y]$ and decrypts it using the output of H (which takes as input π_3 along with z_2).

The AddUser algorithm grants a user u the ability to search the index at a specific access level. The user is added to the set $\mathcal{G}$ of authorized users and a BE key, k_u, is derived for the new user. A new value for st_O is created and encrypted using the BE scheme to create a new server state, st_S. The server key is updated to include the current value of st_S. The new user is given their user secret key K_u which consists of their BE user key, k_u, and the secret keys associated with their access level $k_{\lambda(u),1}, k_{\lambda(u),2}$ and $k_{\lambda(u),3}$. The data owner's key, K_O is updated to include the current version of st_O.

The RevokeUser algorithm revokes a user's search privileges. The user is removed from $\mathcal{G}$ and a new value for st_O is selected and encrypted using the current version of $\mathcal{G}$ to form the new server state st_S. The server key is updated to include the current version of st_S and the updated version of K_O is output.

```
$(K_u) \leftarrow\!\$\ \mathsf{AddUser}(u, \lambda(u), K_O, PP)$
1 :  $\mathcal{G} \leftarrow \mathcal{G} \cup \{u\}$
2 :  $k_u \leftarrow\!\$\ \mathsf{BE.Add}(k_{\mathsf{BE}}, u)$
3 :  $st'_O \xleftarrow{\$} \{0,1\}^{\kappa}$
4 :  $st_O \leftarrow st'_O$
5 :  $st'_S \leftarrow\!\$\ \mathsf{BE.Enc}(st_O, \mathcal{G}, k_{BE})$
6 :  $st_S \leftarrow st'_S$
7 :  Update server key with current value of $st_S$
8 :  return
9 :    $K_u \leftarrow (k_u, k_{\lambda(u),1}, k_{\lambda(u),2}, k_{\lambda(u),3})$
10 :    $K_O \leftarrow (\{k_{a_i,1}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,2}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,3}\}_{i\in[|\mathbb{L}|]}, k_{\mathsf{BE}}, st_O, \mathcal{G})$
```

Algorithm 6.5: Add user algorithm for MLSSE

```
$(K_O) \leftarrow\!\$\ \mathsf{RevokeUser}(u, K_O)$
1 :  $\mathcal{G} \leftarrow \mathcal{G} \setminus \{u\}$
2 :  $st'_O \xleftarrow{\$} \{0,1\}^{\kappa}$
3 :  $st_O \leftarrow st'_O$
4 :  $st'_S \leftarrow\!\$\ \mathsf{BE.Enc}(st_O, \mathcal{G}, k_{BE})$
5 :  $st_S \leftarrow st'_S$
6 :  Update server key with current value of $st_S$
7 :  return
8 :    $K_O \leftarrow (\{k_{a_i,1}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,2}\}_{i\in[|\mathbb{L}|]}, \{k_{a_i,3}\}_{i\in[|\mathbb{L}|]}, k_{\mathsf{BE}}, st_O, \mathcal{G})$
```

Algorithm 6.6: Revoke user algorithm for MLSSE

Theorem 6.5.1. *Let* MLSSE *be the multi-level access SSE scheme defined in Algorithms 6.1-6.6. Given an IND-CPA secure broadcast encryption scheme* BE*, the pseudorandom functions F, G, P, H and a pseudorandom permutation ϕ, then* MLA *is secure in the sense of multi-level access (Definition 6.4.5) and revocation (Definition 6.4.6).*

6.6 Proofs of Security

In this section we give the proofs of security for our construction in Section 6.5.

Lemma 6.6.1. MLSSE *as defined in Algorithms 6.1-6.6 is secure against chosen keyword attacks (Game 6.1) under the same assumptions as in Theorem 6.5.1.*

Proof. Suppose $\mathcal{A}_{MLA}$ is an adversary with non-negligible advantage against Game 6.1 when instantiated with Algorithms 6.1-6.6. We shall reduce the MLA security to the

IND-CPA security of a symmetric encryption scheme Θ, which is defined as follows [87]:

Definition 6.6.2. *Let $F : \{0,1\}^\kappa \times \{0,1\}^* \rightarrow \{0,1\}^k$, be a PRF. We define a symmetric key encryption Θ scheme for messages of length k as follows:*

- $K \overset{\$}{\leftarrow} \Theta.\mathsf{KeyGen}(1^\kappa)$*: A probabilistic algorithm that takes as input a security parameter κ and generates a key $K \overset{\$}{\leftarrow} \{0,1\}^\kappa$.*
- $c \overset{\$}{\leftarrow} \Theta.\mathsf{Enc}(K, m)$*: A probabilistic algorithm that takes as input the key K and a message $m \in \{0,1\}^\kappa$. It chooses a value $r \overset{\$}{\leftarrow} \{0,1\}^\kappa$ and outputs a ciphertext $c =< r, F_K(r) \oplus m >$.*
- $m \leftarrow \Theta.\mathsf{Dec}(c, K)$*: A deterministic algorithm that takes as input a ciphertext $c =< r, s >$ and the key K, and outputs the plaintext message $m = F_K(r) \oplus s$.*

Theorem 6.6.3. *If F is a PRF, then Θ is a fixed-length symmetric key encryption scheme for messages of length k that has indistinguishable encryptions under a chosen plaintext attack.*

Proof. The proof for theorem 6.6.3 is in the textbook *Introduction to Modern Cryptography* by Katz and Lindell [87]. The ciphertext in this scheme is made up of two parts: the output of a PRF and a random k-bit value. Intuitively the security of the scheme holds, since the output of a PRF is indistinguishable from random to an adversary who observes a ciphertext, hence this scheme is similar to the one-time pad. The security relies on the value r not being used in a previous encryption, but since r is chosen at random every time a ciphertext is created, the probability of this happening is negligible in κ. □

Theorem 6.6.4. *If F is a PRF then Θ is a fixed-length symmetric key encryption scheme for messages of length k that has anonymous encryptions under a chosen plaintext attack, meaning given two ciphertexts one cannot ascertain whether they were encrypted under the same key or not.*

Proof. This result can be found in [63]. The anonymity of Θ follows directly from the pseudorandomness of F. □

In our construction, we use an instance of Θ to encrypt the nodes of the array. Since there is only a single encryption key K in the IND-CPA game associated with Θ, and because we use multiple keys for the instance of Θ in our construction, we must first define a sequence of games that interpolate from Game 6.1 to a modified game where all encryptions of index components that differ between the two challenge pre-indexes are performed under the *same* encryption key.

Let $\mathbf{G_0}$ be the real multi-level access game (Game 6.1). Now, observe that a rational adversary in Game 6.1 (that gains a non-negligible advantage) will certainly choose $\mathcal{D}_0^{aug}$ and $\mathcal{D}_1^{aug}$ such that there exists *at least* one element that is both different in each data set (to permit distinguishing) and which is labeled by $\lambda^\star > \lambda^{max}$ (since the maximal query leakage must be equal up to λ^{max}). Let us denote by $X = \{(x_{0,i}, x_{1,i})\}$ the set of pairs of data item identifiers $(x_{0,i}, x_{1,i})$, where $x_{j,i} \in \mathcal{D}_j^{aug}$, that are labelled by any label $\lambda^\star > \lambda^{max}$, and where $x_{j,i} \notin \mathcal{D}_{1-j}^{aug}$. Let us denote each pair in X by $X_i = (x_{0,i}, x_{1,i})$. Informally, X is the set of pairs of identifiers which differ in the challenge data sets. When $\mathsf{BuildIndex}$ is run, γ creates a list of identifiers of data items that contain a keyword ω. A separate list is created for every keyword. These lists are then ordered from the identifiers with the highest access levels to those with the lowest; these ordered lists are called $\mathsf{L}_{\omega_i}, i \in [|\Delta|]$. A set of nodes is then chosen at random from $\mathbb{A}$ to store each of these identifiers (along with other data). Let $y_{b,i}$ denote the set of nodes holding the identifiers in $x_{b,i}$. We have that $|y_{b,i}|$ is equal to the number of distinct keywords in the data item associated with the identifier in $x_{b,i}$. Let $Z = \{y_{b,1}, ..., y_{b,|X|}\}$, hence $|Z| = \sum_{i=1}^{|X|} |y_{b,i}|$. We can thus also represent Z as follows: $Z = \{z_{b,1}, ..., z_{b,|Z|}\}$.

We define a sequence of $|Z|$ games, denoted $\mathbf{G_i}$ for $i = 1, ..., |Z|$. Game $\mathbf{G_i}$ is defined in exactly the same way as Game $\mathbf{G_{i-1}}$, with the exception that, for a random bit b, the node $z_{b,i} \in Z$ is encrypted under a fixed key $k^\star$ rather than the key defined by the output of the PRF P during $\mathsf{BuildIndex}$. For $k = 1$ to $|Z|$, we show that no efficient distinguisher exists that can distinguish $\mathbf{G_k}$ from $\mathbf{G_{k-1}}$.

Suppose, in search for a contradiction, that such a distinguisher D_k does exist with non-negligible advantage δ. We show that, using D_k as a subroutine, one can construct an efficient adversary $\mathcal{A}_{anon}$ that can break the anonymity of Θ.

For the anonymity game, the challenger $\mathcal{C}_{anon}$ generates two keys, k and k', where $k \overset{\$}{\leftarrow} \{0,1\}^\kappa$ and k' is the output generated by the PRF P in $\mathsf{BuildIndex}$ that is used as the key for the PRF H associated with node $z_{b,i}$. $\mathcal{C}_{anon}$ also chooses a random bit $b^\star$. It then offers $\mathcal{A}_{anon}$ access to two encryption oracles, $\mathsf{Enc_1}$ and $\mathsf{Enc_2}$: if $b^\star = 0$, then each oracle uses a different encryption key (k or k'); if $b^\star = 1$, both oracles use the same key k.

$\mathcal{A}_{anon}$ simulates either Game $\mathbf{G_{k-1}}$ or Game $\mathbf{G_k}$ for D_k. To do so, $\mathcal{A}_{anon}$ creates the Game $\mathbf{G_{k-1}}$, with the following modification. Let $Y = \{z_{b,i}$ for $i < k\}$. When running $\mathsf{BuildIndex}$ and encrypting the node in Y, $\mathcal{A}_{anon}$ does not derive the key for the PRF H using the PRF P or perform the symmetric encryption. Instead, the node to be encrypted is sent to the oracle $\mathsf{Enc_1}$. The challenger returns an encryption under a fixed key k, and this ciphertext is used in the index.

When running $\mathsf{BuildIndex}$ and encrypting the identifier relating to item $z_{b,k}$, $\mathcal{A}_{anon}$ does not derive the key for the PRF H using the PRF P or perform the symmetric encryption. In this case, the plaintext is sent to the oracle Enc_2. $\mathcal{A}_{anon}$ receives back either an encryption under key k (if $b^\star = 0$) or key k' (if $b^\star = 1$).

Note that, if $b^\star = 0$, then $\mathbf{G_k}$ is identical to $\mathbf{G_{k-1}}$. Otherwise, the difference between the games is precisely that the encryption of $z_{b,k}$ is performed under a different key k' to all items in Y which are encrypted under k. Thus, if D_k can distinguish $\mathbf{G_k}$ from $\mathbf{G_{k-1}}$, they are able to identify whether $z_{b,k}$ is encrypted under the same key as all items in Y, or not. Since all of these encryptions were performed using the oracles provided by the anonymity game, $\mathcal{A}_{anon}$ can forward D_k's output to break the anonymity of the symmetric encryption scheme with the same non-negligible advantage δ. Clearly, since we assume that the encryption scheme is anonymous, such a distinguisher cannot exist and all possible efficient distinguishers must have at most a negligible advantage Adv_{D_k}.

Now, given that each of the above game hops has a negligible distinguishing advantage, the total game hop from $\mathbf{G_0}$ to $\mathbf{G_k}$ has negligible distinguishing advantage ($\sum_{i=1}^{\kappa} Adv_{D_i}$). We can therefore run an adversary against Game 6.1 against Game $\mathbf{G_k}$ instead, and it will fail with at most a negligible probability. We hence reduce Game G_k to the IND-CPA security of the symmetric key encryption scheme Θ.

Let $\mathcal{C}$ be a challenger for an adversary $\mathcal{A}_{INDCPA}$ in the IND-CPA security game of the symmetric encryption scheme $\Theta = (\Theta.\mathsf{KeyGen}, \Theta.\mathsf{Dec}, \Theta.\mathsf{Dec})$. $\mathcal{A}_{INDCPA}$ will in turn act as the challenger in the multi-level access game against an adversary $\mathcal{A}_{MLA}$. Suppose $\mathcal{A}_{MLA}$ is an adversary that can win Game $\mathbf{G_k}$ for a policy $\mathcal{P}$ with non-negligible probability δ. We construct an adversary $\mathcal{A}_{INDCPA}$ that uses $\mathcal{A}_{MLA}$ as a subroutine that can break the IND-CPA security of Θ as follows:

- $\mathcal{C}$ runs $\Theta.\mathsf{KeyGen}$ to generate a symmetric key $k^\star$, chooses a bit b uniformly at random, and gives the security parameter 1^κ to $\mathcal{A}_{INDCPA}$.

- $\mathcal{A}_{INDCPA}$ sets up the multi-level access game by initialising $L = \emptyset$, $\lambda^{max} = \perp$ and $\mathsf{chall} = \mathsf{false}$. It runs $\mathsf{KeyGen}(1^\kappa, S, \mathcal{P})$ to generate (K_O, K_S, PP), and gives $\mathcal{A}_{MLA}$ the values $(1^\kappa, \mathcal{P}, K_S, \mathrm{PP})$.

- $\mathcal{A}_{MLA}$ may now make oracle queries to $\mathcal{A}_{INDCPA}$. Each oracle is run as specified in Game 6.1.

- Eventually, $\mathcal{A}_{MLA}$ will output its choice of two data sets such that the maximal query leakage for the highest security label, λ^{max}, queried to AddUser is equal for both data sets (note that this is necessarily true for a successful adversary). $\mathcal{A}_{INDCPA}$ sets chall to be true and chooses a random bit b.

Now, observe that a rational adversary (that gains a non-negligible advantage) will certainly choose $\mathcal{D}_0^{aug}$ and $\mathcal{D}_1^{aug}$ such that there exists at least one element that is both different in each data set (to permit distinguishing) and which is labeled by $\lambda^\star > \lambda^{max}$ (since the maximal traces must be equal up to λ^{max}). Note that, by definition of λ^{max}, $\lambda^\star$ has not been queried to the AddUser oracle.

Let us denote by $X = \{(x_0, x_1)\}$ the set of pairs of elements (x_0, x_1), where $x_i \in \mathcal{D}_i^{aug}$, which are labelled by any $\lambda(x) \geqslant \lambda^\star$ and where $x_i \notin \mathcal{D}_{1-i}^{aug}$. Informally, X is the set of pairs of elements which differ in the challenge data sets.

- Set $\hat{b} = b$.
- $\mathcal{A}_{INDCPA}$ must now build a challenger index for $\mathcal{A}_{MLA}$, $\mathcal{D}_{\hat{b}}^{aug}$, which embeds a challenge for the IND-CPA game. To build the challenge index, $\mathcal{A}_{INDCPA}$ follows the BuildIndex algorithm as written, with the exception that, whenever an item $x_b \in X$ is to be encrypted using the key $K_{i,j}$ to form the entry in $\mathbb{A}$, the plaintexts associated with x_0 and x_1 are instead sent to the LoR oracle of $\mathcal{C}$. It receives back an encryption of x_b under key $k^\star$.
- The index is given to $\mathcal{A}_{MLA}$, who can make oracle queries as follows:
  - AddUser is run as written in Game 6.1. If the maximal trace of both challenge data sets are equal given the new user's security label, then the elements of the data sets are not in X and can be treated normally.
  - RevokeUser is run as written in Game 6.1.
  - BuildIndex: each run of BuildIndex should generate and use new keys, therefore it can be run as written.
- Eventually, $\mathcal{A}_{MLA}$ outputs a guess $\hat{b}'$ for which data set was encoded in the index.
- $\mathcal{A}_{INDCPA}$ outputs their guess $b' = \hat{b}'$ to $\mathcal{C}$ as their guess for b.

Observe that both data sets are identical, except for those elements contained in X. Each pair (x_0, x_1) of differing elements in X are sent to the LoR oracle of $\mathcal{C}$, who always encrypts x_b. Thus, if $b = 0$, then data set $\mathcal{D}_0^{aug}$ is encoded; otherwise data set $\mathcal{D}_1^{aug}$ is encoded.

If $\mathcal{A}_{MLA}$ can distinguish the datatset that was chosen with non-negligible advantage δ in the multi-level access game, we have that:

$$Adv_{\mathcal{A}_{MLA}} = \delta,$$

hence:

$$\mathbb{P}[\hat{b}' = \hat{b}] = \delta + \frac{1}{2}.$$

Using these assumptions we can calculate the advantage of $\mathcal{A}_{INDCPA}$ against the challenger, $\mathcal{C}$, in their game. We start by calculating the probability of $\mathcal{A}_{INDCPA}$ winning their game:

$$\begin{aligned}\mathbb{P}[b' = b] &= \mathbb{P}[b' = b|\hat{b}' = \hat{b}]\mathbb{P}[\hat{b}' = \hat{b}] + \mathbb{P}[b' = b|\hat{b}' \neq \hat{b}]\mathbb{P}[\hat{b}' \neq \hat{b}]\\ &= 1 \cdot \mathbb{P}[\hat{b}' = \hat{b}] + 0 \cdot \mathbb{P}[\hat{b}' \neq \hat{b}]\\ &= \delta + \frac{1}{2}.\end{aligned}$$

Hence,

$$\begin{aligned}Adv_{\mathcal{A}_{IND\text{-}CPA}} &= \left|\mathbb{P}[b' = b] - \frac{1}{2}\right|\\ &= \left|(\delta + \frac{1}{2}) - \frac{1}{2}\right|\\ &= \delta\end{aligned}$$

As we fixed δ to be non-negligible we have that the advantage of $\mathcal{A}_{IND\text{-}CPA}$ is also non-negligible which contradicts the security of the INDCPA-secure symmetric encryption scheme. We conclude that if the symmetric encryption scheme $\mathcal{SE}$ is IND-CPA secure then MLSSE as defined in Algorithms 6.1-6.6 is secure in the sense of multi-level access.

□

Lemma 6.6.5. MLSSE*, as defined in Algorithms 6.1 - 6.6, is secure against revocation (Game 6.2) under the same assumptions as for Theorem 6.5.1.*

Proof. Assume $\mathcal{A}_{MLA}$ is an adversary with non-negligible advantage δ in Game 6.2 when instantiated with the Algorithms 6.1 − 6.6. We will show that if $\mathcal{A}_{MLA}$ has non-negligible advantage in Game 6.2, then we can construct an adversary $\mathcal{A}_{\mathsf{BE}}$ that uses $\mathcal{A}_{MLA}$ as a subroutine to break the security of a broadcast encryption scheme $\mathsf{BE} = (\mathsf{BE.KeyGen}, \mathsf{BE.Encrypt}, \mathsf{BE.Decrypt})$, as defined in Game 2.7.

In order for Game 6.2 to output 1, it is required that the adversary produce a valid search query. To produce a valid search query, the user is required to know st_O, which is the value of the key for the PRP used to produce the search query. As A new value of st_O is randomly selected and encrypted each time a user is revoked from the system using $\mathsf{BE.Enc}(K_{\mathsf{BE}}, \mathcal{G} \setminus u_i, st_O)$ (Revoke), where $\mathcal{G} \setminus u_i$ is the new group of authorised users. This encrypted value is then broadcast to all users. The security of the BE scheme ensures that only authorised users can decrypt this ciphertext to obtain st_O with overwhelming probability. Hence the adversary is only able to create a valid search query if they are an authorized user, or if they are able to break the security of the BE scheme. We note that an adversary could potentially win the game if they output a bit string for T_ω that by chance leads to some output other than the failure symbol. This is prevented by including the fourth input to the search query which is a κ-bit zero string to encode the search query. We use this method of adding redundancy to the string, described by Bellare and Rogaway [21], to turn our application of the PRP to the search query into an authenticated encryption scheme. Encoding the search query by adding κ-bit zero string ensures the encodings are sparse. That is, the probability that a random bit string is valid is $2^{-\kappa}$ if the bit string is at least κ bits long. This means that the adversary will not be able to produce a valid PRP ciphertext will all but negligible probability. Let $\mathcal{C}$ be the challenger for the adversary $\mathcal{A}_{\mathsf{BE}}$ against the broadcast encryption scheme, $\mathcal{A}_{\mathsf{BE}}$ will act as the challenger for $\mathcal{A}_{MLA}$.

1. The KeyGen algorithm is run by $\mathcal{A}_{\mathsf{BE}}$ (with the exception of the generation of $\mathsf{BE.KeyGen}$). The secret key for the broadcast encryption scheme (k_{BE}) is generated and held by $\mathcal{C}$: $k_{\mathsf{BE}} \leftarrow \mathsf{BE.KeyGen}(1^\kappa)$ (this is equivalent to running line 3 of KeyGen). $\mathcal{A}_{\mathsf{BE}}$ then initiates $\emptyset \leftarrow \mathcal{G}$ and chooses $st_O \xleftarrow{\$} \{0,1\}$ and submits these values to $\mathcal{C}$, who runs $\mathsf{BE.Enc}(st_O, \mathcal{G}, k_{BE})$ to produce st_S, which is sent to $\mathcal{A}_{\mathsf{BE}}$. $\mathcal{A}_{\mathsf{BE}}$ then proceeds to run the rest of KeyGen. This generates the keys K_O, K_S which are held by $\mathcal{A}_{\mathsf{BE}}$, and the public parameters PP (with the exception that K_O does not contain k_{BE}). $\mathcal{A}_{MLA}$ is given inputs X

2. $\mathcal{A}_{\mathsf{BE}}$ issues a query to $\mathcal{C}$ for the secret key of $u_{\mathcal{A}_{MLA}}$. In response, $\mathcal{C}$ runs: $k_{u_{\mathcal{A}_{MLA}}} \leftarrow \mathsf{BE.AddUser}(k_{\mathsf{BE}})$ and sends $k_{u_{\mathcal{A}_{MLA}}}$ to $\mathcal{A}_{\mathsf{BE}}$. To fully enrol $u_{\mathcal{A}_{MLA}}$ as an authorized user the server state also needs to be updated. This is done by submitting $\mathcal{G}$ and a newly generated st_O to $\mathcal{C}$ who runs $\mathsf{BE.Enc}(st_O, \mathcal{G}, k_{BE})$ to generate st_S. $\mathcal{A}_{\mathsf{BE}}$ is now able to form $\mathcal{A}_{MLA}$'s secret key $K_{u_{\mathcal{A}_{MLA}}}$. This step is equivalent to running $\mathsf{AddUser}$.

3. $\mathcal{A}_{\mathsf{BE}}$ runs $\mathsf{BuildIndex}$ to produce the index and the ciphertexts $(\mathcal{I}_{\mathcal{D}}, \mathcal{C})$, and these values are sent to $\mathcal{A}_{MLA}$ along with $K_{u_{\mathcal{A}_{MLA}}}$.

4. $\mathcal{A}_{MLA}$ is given access to oracles $\textsc{AddUser}(\cdot, \lambda(\cdot), K_O, PP)$ and $\textsc{RevokeUser}(\cdot, \mathrm{K_O}, \mathrm{PP})$ as described in Game 6.2.

5. $\mathcal{A}_{\mathsf{BE}}$ revokes $\mathcal{A}_{MLA}$ from the system by running $\mathsf{RevokeUser}$. $\mathcal{A}_{\mathsf{BE}}$ runs line 1 of Revoke a second time in order to produce two values for $st_O : \theta_0, \theta_1$. These values act as the challenge plaintexts for $\mathcal{A}_{\mathsf{BE}}$, which are submitted to $\mathcal{C}$ along with a set of users $\mathcal{G} \subset \mathcal{U}$ (such that no revoked user is in $\mathcal{G}$).

6. $\mathcal{C}$ selects a bit $b \overset{\$}{\leftarrow} \{0, 1\}$.

7. $\mathcal{C}$ encrypts θ_b using the encryption algorithm of the BE scheme: $(st_S)_b \leftarrow \mathsf{BE.Enc}(\theta_b, \mathcal{G}, k_{\mathsf{BE}})$. $\mathcal{C}$ sends $(st_S)_b$ to $\mathcal{A}_{\mathsf{BE}}$ as their challenge ciphertext. $\mathcal{A}_{\mathsf{BE}}$ then transfers it to $\mathcal{A}_{MLA}$.

8. $\mathcal{A}_{MLA}$ then outputs a query that they believe to be valid (in other words, will produce a search result not equal to $\perp$): T_ω. This happens with non-negligible probability δ, due to our assumptions that $\mathcal{A}$ has non-negligible advantage δ in Game 6.2. $\mathcal{A}_{MLA}$ sends this to $\mathcal{A}_{\mathsf{BE}}$.

9. $\mathcal{A}_{\mathsf{BE}}$ runs line 2 of Search twice; once using st_{O0} as the key for the inverted PRP, and once using st_{O1} as the key for the inverted PRP. The possible outputs are:

$$\phi^{-1}_{st_{O0}}(T_\omega) = ({t_\omega}^0 \vee \perp); \text{ and}$$

$$\phi^{-1}_{st_{O1}}(T_\omega) = (t^1_{\omega,} \vee \perp).$$

10. If $t^0_\omega = t^1_\omega = \ \perp$, then we have that $\mathsf{Search}(st_S, \mathcal{I_D}, T_{\omega,\lambda(\mathcal{A})}) = \perp$, and $\mathcal{A}_{\mathsf{BE}}$ outputs their guess for b as $b' \overset{\$}{\leftarrow} \{0, 1\}$.

11. If $t^0_\omega \neq \perp$, this tells $\mathcal{A}_{BE}$ that $(st_S)_0$ was used to generate the query, and $\mathcal{A}_{BE}$ outputs their guess for b as $b' = 0$. If $t^1_\omega \neq \perp$, this tells $\mathcal{A}_{\mathsf{BE}}$ that $(st_S)_1$ was used to generate the query, and outputs their guess for b as $b' = 1$. We note here that Search can also output $\perp$ on line 5. However, the adversary has all the information to form this part of the query correctly hence, for a rational adversary, this line should never output $\perp$. Thus we assume that this line will not output $\perp$ in our game. As we have assumed that $\mathcal{A}_{MLA}$ has non-negligible advantage δ in their game, we calculate the advantage of $\mathcal{A}_{\mathsf{BE}}$ to be:

$$|\Bigg[\Bigg(\mathbb{P}\Big[((t^0_\omega \ \vee \ t^1_\omega) \neq \perp)\Big] \cdot 1 - \frac{1}{2}\Bigg) +$$

$$\Bigg(\mathbb{P}\Big[((t^0_\omega \ \wedge \ t^1_\omega) = \perp)\Big] \cdot \frac{1}{2} - \frac{1}{2}\Bigg)\Bigg]|$$

$$= |\Big[\delta \cdot 1 + (1 - \delta) \cdot \frac{1}{2}\Big] - \frac{1}{2}| = |\Big[\frac{(\delta + 1)}{2}\Big] - \frac{1}{2}| = \frac{\delta}{2}.$$

As we have assumed that δ is non-negligible, we have that $\frac{\delta}{2}$ is also non-neglible.

From this we conclude that there cannot exist an adversary $\mathcal{A}$ with non-negligible probability against Game 6.2.

□

6.6.1 Achieving dynamicity

We can extend MLSSE to support multi-level access on a dynamic data set by adding two new data structures to the index: a deletion table ($\mathbb{T}_d$) and a deletion array ($\mathbb{A}_d$). There are also four additional algorithms: AddToken, Add, DeleteToken, Delete. Array $\mathbb{A}_d$ stores a list of nodes for each data item which point to nodes in $\mathbb{A}$ that would need to be removed if the corresponding data item was deleted. This means that every node in $\mathbb{A}$ will have a corresponding node in $\mathbb{A}_d$, which is called its *dual* node. $\mathbb{T}_d$ is a table with an entry for each data item which points to the start of the corresponding linked list in $\mathbb{A}_d$, given a valid *delete token* for that data item. In addition to these two new structures the index consists of a search array $\mathbb{A}_s$ and a search table $\mathbb{T}_s$ (as in the original construction) and a *free list* that keeps track of all the unused space in $\mathbb{A}_s$.

In the dynamic scheme searching for a keyword is done similarly to the static construction in Section 6.5 and follows the concept of linked lists presented by [55].

To add a data item to the index, changes need to be made to $\mathbb{T}_d, \mathbb{A}_s$ and $\mathbb{A}_d$. The data owner creates an *add token* using AddToken and sends this to the server. The server then determines the free space available in $\mathbb{A}_s$ using the free list and adds the relevant information to the free nodes and updates the free list. When adding a new data item the relevant nodes cannot be added to the end of each linked list; instead we have to insert in the appropriate place in the linked list according to the access level of the new data item. Information in the add token will allow the server to locate the correct point at which to insert the nodes in each linked list, so instead of the entry in $\mathbb{T}_s$ just pointing to the end node of each linked list this is altered so that it points to the correct node in the linked list according to the access level of the new data item. The respective predecessor of each new node is modified to point to the new node instead of its previous ancestor.

In order to remove a data item, a deletion token is created which allows the server to locate and delete the correct entries in $\mathbb{T}_d$. This, in turn, allows the server to locate and delete the correct entries in $\mathbb{A}_s$. Some nodes will need to be updated in $\mathbb{A}_s$ (as some of the linked lists will have nodes which point to nodes that have been deleted) and this is done using homomorphic encryption.

Scheme	Index size	Search time	Query comp.	Search comp.
Single user [55]	$\mathcal{O}(\lvert\Delta\rvert) + \mathcal{O}(\lvert\Delta\rvert)$	$\mathcal{O}(1) + \mathcal{O}(\lvert\delta_\omega\rvert)$	1PRP +1PRF	$1XOR + \lvert\delta_\omega\rvert \cdot (\text{Dec}_{SKE})$
MLSSE	$\mathcal{O}(\lvert\Delta\rvert \cdot \lvert\lvert\mathbb{L}\rvert) + \mathcal{O}(\lvert\Delta\rvert)$	$\mathcal{O}(1) + \mathcal{O}(\lvert\delta_{\omega,\lambda(u)}\rvert)$	$1PRP + 3PRF$	$1PRP + \lvert\delta_{\omega,\lambda(u)}\rvert \cdot (1PRF + 2XOR)$
[83]	$\mathcal{O}(\lvert\Delta\rvert) + \mathcal{O}(\lvert\Delta\rvert)$	$\mathcal{O}(1) + \mathcal{O}(\lvert\delta_\omega\rvert) + \mathcal{O}(\lvert\delta_\omega\rvert)$	$1PRF + 1PRP$	$\lvert\delta_\omega\rvert \cdot (\text{Dec}_{SKE} + \text{Dec}_{CPABE})$
[98]	$\mathcal{O}\lvert\Delta\rvert$	$\mathcal{O}(\lvert\Delta\rvert)$	$3E$	$\lvert\Delta\rvert \cdot (9P)$

Table 6.1: Performance of MLSSE

6.7 Performance

In this section we discuss the performance of our multi-level SSE scheme and make some comparisons to schemes in the literature that are most similar to ours.

The features we will look at to evaluate performance are the search time and search computation, query computation time and index size. We feel these features contribute greatly to providing a useable scheme in practice. A compact index means less server storage needs to be used and fast query and search times make querying efficient for the user.

Our scheme is built using symmetric key primitives, which are widely known to be more efficient than their public key counterparts. Table 6.1 summarises the operations required for the specified algorithms in our scheme, where E denotes group exponentiation, P denotes a group pairing, XOR denotes an XOR operation, PRP denotes evaluation of a PRP and PRF denotes evaluation of a PRF.

Comparing our multi-level user scheme to a single user scheme (without multi-level access) we can see that adding the multi-level user functionality does not hamper the performance of the scheme to a great degree. The lookup table $\mathbb{T}$ is larger by a factor of $|\mathbb{L}|$, where $|\mathbb{L}|$ is the number of access levels, and 2 extra PRPs are required when generating a query. The search time of our multi-level scheme is less than that of a single user scheme as it linear in the size of the search results. Using a single user scheme without multiple access levels results in the user receiving all search results matching a keyword, whereas in our multi-level access scheme the user only receives a portion of these search results, namely, the ones that are relevant to their access level. The only situation where the search times will be the same for these two schemes is when the user is at the highest access level and is authorised to receive all search results containing a specified keyword. The same applies to the search computation for these two schemes.

The other two schemes in our comparison that support multi-level access both require more computationally intensive operations in their search algorithms, such as CP-ABE decryptions or pairings of group elements.

6.8 Summary

In this chapter, we present a new notion of searchable symmetric encryption with multi-level access. The notion of multi-level access to encrypted data accurately reflects many real-world scenarios where multiple users have varying levels of access to a central outsourced data set; a few examples being: classified government documents, subscription services and corporate data.

We present a new system model, along with a new security model that extends the previous notions of security in SSE to fulfil our new definitional framework.

This work is a useful first step in making SSE more suitable for deployment, since the multi-level user notion more accurately reflects a group of users in a real world data-sharing scenario than that of the standard SSE scheme notion, which only supports a homogeneous user group.

Chapter 7

Conclusion and Future Work

In this thesis we have considered the role of searchable encryption in the real world as opposed to only being an interesting theoretical concept. This thesis is partially about the need for more functional schemes in some areas of searchable encryption in order to match the expectations of potential users.

In Chapter 4 we present searchable encryption within the framework of four general scenarios, which reflect settings where searchable encryption could be applied in the real world. We provide a general method for performing encrypted search in each scenario, as well as mapping specific searchable encryption schemes into each scenario, and evaluating their applicability using several variable features. We further identified several factors that we believe are preventing the widespread adoption of searchable encryption, one of which is 'user education'; that is, potential users of searchable encryption, or developers of products, might not understand how the technology can be used to solve a problem. The work in this chapter is intended to address this issue, by providing informative guidance for potential adopters of searchable encryption technology.

In Chapter 5 we focus on extending the functionality in verifiable searchable encryption. One of the factors we identified as affecting the adoption of searchable encryption is that of 'usability', meaning that in order for searchable encryption to become a technology that people wish to use, its functionality needs to be similar to that of protocols they are more familiar with. The expressiveness of queries in verifiable searchable encryption was limited, compared with that of non-verifiable SE schemes, but verifiable computation schemes can perform a wide variety of computations. We also noticed that the system models for verifiable computation and searchable encryption are similar. Furthermore, the attributes computed on in verifiable computation can be made to correspond to the keywords used to describe data items in searchable encryption.

An extension which could be implemented for our construction in Section 5.3 is to allow multiple users to store data on a shared server without having to initialise their own scheme. In practice, this could result in a key distribution centre, which is used in verifiable delegable computation, setting up the system and publishing public parameters that any data owner can use, but enabling each data owner to generate their own CP-ABE decryption keys for the data they hold. We would also like to investigate ways of reducing the search query size in our construction, whilst maintaining their expressiveness.

Through research undertaken at Thales Research and Technology, we learnt that many potential application scenarios involve many users requiring access to a dataset, where the users possess differing access rights to the data. There are few existing solutions within the realm of searchable encryption that support this system model. Existing solutions were primarily based on public-key cryptographic primitives, which can be computationally intensive. This inspired our work in Chapter 6, which presents a searchable encryption scheme with *multi-level* access, constructed using simple and efficient symmetric-key primitives, such as pseudorandom functions and pseudorandom permutations.

We identified several areas for future work during the development of our multi-level SSE scheme. In Section 6.6.1, we describe an extension to our construction in Section 6.5 that allows it to support a dynamic dataset. We aim to formally define this dynamic extension with a full construction, along with a formal security proof. Furthermore, our MLSSE construction is secure against colluding users that are authorized to search over the encrypted data. It would be interesting to extend this notion of security to provide security against the collusion of revoked users with the server. Another area of interest is that of user addition and revocation. In our construction we have used a broadcast encryption scheme as a black box to add and revoke users in the system. In order to quantify the efficiency of our scheme, we would like to investigate this method of revocation, and determine which broadcast encryption would produce the most practical construction.

The addition and revocation of users in a multi-user SSE scheme is also of independent interest. It would be interesting to investigate generic methods of achieving user addition and revocation for multi-user SSE schemes that can be applied to any single-user SSE scheme. In some practical scenarios it may be beneficial for the data owner to know which users have issued which search queries; this is called *user accountability*. If the data owner wanted to audit the scheme in our construction, they would be able to tell the access level of the user submitting each search query, but not the individual identities. This could be desirable in some cases, but we could also investigate extending our construction to support user accountability.

As is common in the SSE literature, our construction leaks the access pattern and search pattern (with respect to access levels). The standard system model is to use one server to perform the search over the encrypted data. We hope to investigate the security improvements we can achieve when distributing the index over multiple servers. We hypothesise that this would also distribute the leakage, so one server only learns a fraction of the leakage associated with the whole search.

Bibliography

[1] Cipherlocker^{TM}. https://cipherlocker.com/, 2017. 76

[2] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005. 55, 56

[3] M. Abdalla, M. Bellare, and G. Neven. Robust encryption. In *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010*, volume 5978 of *Lecture Notes in Computer Science*, pages 480–497. Springer, 2010. 55

[4] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 563–574. ACM, 2004. 64

[5] J. Alderman, C. Janson, C. Cid, and J. Crampton. Hybrid publicly verifiable computation. IACR Cryptology ePrint Archive, Report 2015/320, 2015. 86, 110

[6] J. Alderman, C. Janson, K. M. Martin, and S. L. Renwick. Extended functionality in verifiable searchable encryption. In *Cryptography and Information Security in the Balkans - Second International Conference, BalkanCryptSec 2015*, Lecture Notes in Computer Science, pages 187–205. Springer, 2015. 29, 83, 96

[7] J. Alderman, K. M. Martin, and S. L. Renwick. Multi-level access in searchable symmetric encryption. IACR Cryptology ePrint Archive, Report 2017/211, 2017. 15

[8] D. Apon, J. Katz, E. Shi, and A. Thiruvengadam. Verifiable oblivious storage. In *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 131–148. Springer, 2014. 28, 86

[9] M. Azraoui, Elkhiyaoui, M. Önen, and R. Molva. Publicly verifiable conjunctive keyword search in outsourced databases. In *2015 IEEE Conference on Communications and Network Security, CNS 2015*, pages 619–627. IEEE, 2015. 85

[10] M. Backes, M. Barbosa, D. Fiore, and R. M. Reischuk. ADSNARK: nearly practical and privacy-preserving proofs on authenticated data. In *2015 IEEE Symposium on Security and Privacy, SP 2015*, pages 271–286. IEEE Computer Society, 2015. 28, 86

[11] M. Backes, D. Fiore, and R. M. Reischuk. Verifiable delegation of computation on outsourced data. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 863–874. ACM, 2013. 28, 86

[12] J. Baek, R. Safavi-Naini, and W. Susilo. On the integration of public key data encryption and public key encryption with keyword search. In *Information Security, 9th International Conference, ISC 2006*, volume 4176 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2006. 59

[13] J. Baek, R. Safavi-Naini, and W. Susilo. Public key encryption with keyword search revisited. In *Computational Science and Its Applications - ICCSA 2008*, volume 5072 of *Lecture Notes in Computer Science*, pages 1249–1259. Springer, 2008. 56, 57, 72, 76

[14] L. Ballard, S. Kamara, and F. Monrose. Achieving efficient conjunctive keyword searches over encrypted data. In *Information and Communications Security, 7th International Conference, ICICS 2005*, volume 3783 of *Lecture Notes in Computer Science*, pages 414–426. Springer, 2005. 63

[15] F. Bao, R. H. Deng, X. Ding, and Y. Yang. Private query on encrypted data in multi-user settings. In *Information Security Practice and Experience, 4th International Conference*, volume 4991 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2008. 75, 130, 132

[16] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001. 85

[17] O. Baudron, D. Pointcheval, and J. Stern. Extended notions of security for multicast public key cryptosystems. In *Automata, Languages and Programming,*

27th International Colloquium, ICALP 2000, volume 1853 of *Lecture Notes in Computer Science*, pages 499–511. Springer, 2000. 60

[18] E. Bell and L. La Padula. Secure computer system: Unified exposition and multics interpretation. Technical report, Mitre Corporation, 1976. 128

[19] M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 2000. 60

[20] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 535–552. Springer, 2007. 59, 69, 75, 77

[21] M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000. 163

[22] S. U. Ben Lynn. https://crypto.stanford.edu/pbc/notes/crypto/prp.html. 116

[23] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from rams to delegatable succinct constraint satisfaction problems: extended abstract. In *Innovations in Theoretical Computer Science, ITCS '13*, pages 401–414. ACM, 2013. 28, 86

[24] S. Benabbas, R. Gennaro, and Y. Vahlis. Verifiable delegation of computation over large datasets. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011. 28, 86

[25] J. Benaloh, M. Chase, E. Horvitz, and K. E. Lauter. Patient controlled encryption: ensuring privacy of electronic medical records. In *Proceedings of the first ACM Cloud Computing Security Workshop, CCSW 2009*, pages 103–114. ACM, 2009. 129

[26] J. Bethencourt, H. T. Chan, A. Perrig, E. Shi, and D. X. Song. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (S&P 2007)*, pages 350–364. IEEE Computer Society, 2007. 58

[27] S. Bhattacherjee and P. Sarkar. Reducing communication overhead of the subset difference scheme. *IEEE Trans. Computers*, 65:2575–2587, 2016. 99, 100

[28] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012*, pages 326–349. ACM, 2012. 28, 86

[29] E. Blass, T. Mayberry, and G. Noubir. Multi-user oblivious ram secure against malicious servers. IACR Cryptology ePrint Archive, Report 2015/121, 2015. 54, 70, 76

[30] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970. 61

[31] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 5479 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2009. 64, 69

[32] A. Boldyreva, N. Chenette, and A. O'Neill. Order-preserving encryption revisited: Improved security analysis and alternative solutions. IACR Cryptology ePrint Archive, Report 2012/625, 2012. 64, 77

[33] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004. 55, 56, 85, 128

[34] D. Boneh and B. Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007. 57, 72

[35] C. Bösch, P. H. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Computing Surveys*, 47:18:1–1851, 2014. 67

[36] R. Bost, P. Fouque, and D. Pointcheval. Verifiable dynamic symmetric searchable encryption optimality and forward security. IACR Cryptology ePrint Archive, Report 2016/062, 2016. 85

[37] J. W. Byun and D. H. Lee. On a security model of conjunctive keyword search over encrypted relational database. *Journal of Systems and Software*, 84:1364–1372, 2011. 63

[38] J. W. Byun, D. H. Lee, and J. Lim. Efficient conjunctive keyword search on encrypted data storage system. In *Public Key Infrastructure, Third European PKI Workshop: Theory and Practice, EuroPKI 2006*, volume 4043 of *Lecture Notes in Computer Science*, pages 184–196. Springer, 2006. 63

[39] J. W. Byun, H. S. Rhee, H. Park, and D. H. Lee. Off-line keyword guessing attacks on recent keyword search schemes over encrypted data. In *Secure Data Management, Third VLDB Workshop, SDM 2006*, volume 4165 of *Lecture Notes in Computer Science*, pages 75–83. Springer, 2006. 58, 128

[40] N. Cao, C. Wang, M. Li, K. Ren, and W. Lou. Privacy-preserving multi-keyword ranked search over encrypted cloud data. In *INFOCOM 2011. 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 829–837. IEEE, 2011. 131

[41] D. Cash, P. Grubbs, J. Perry, and T. 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. 59, 134

[42] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society, 2014. 64, 69, 70

[43] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M. Rosu, and M. Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference*, volume 8042 of *Lecture Notes in Computer Science*, pages 353–373. Springer, 2013. 64, 69

[44] Q. Chai and G. Gong. Verifiable symmetric searchable encryption for semi-honest-but-curious cloud servers. In *Proceedings of IEEE International Conference on Communications, ICC 2012*, pages 917–922. IEEE, 2012. 84, 85, 124, 126

[45] H. T. Chan, E. Shi, and X. Wang. 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. 54, 70

[46] Y. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005*, volume 3531 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2005. 61, 69, 134, 145, 146

[47] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010. 64, 69, 145, 146

[48] L. Chen and Q. Tang. Public-key encryption with registered keyword search. In *Public Key Infrastructures, Services and Applications - 6th European Workshop, EuroPKI 2009*, volume 6391 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2009. 58, 72

[49] R. Chen, F. Guo, X. Wang, and G. Yang. A new general framework for secure public key encryption with keyword search. In *Information Security and Privacy - 20th Australasian Conference, ACISP 2015*, volume 9144 of *Lecture Notes in Computer Science*, pages 59–76. Springer, 2015. 59, 72

[50] Y. Chen, D. Lin, J. Zhang, and Z. Zhang. Generic constructions of integrated pke and peks. *Designs, Codes and Cryptography*, 78:493–526, 2016. 59

[51] R. Cheng, J. Yan, C. Guan, F. Zhang, and K. Ren. Verifiable searchable symmetric encryption from indistinguishability obfuscation. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15*, pages 621–626. ACM, 2015. 85, 125, 126

[52] J. Chi, D. Feng, Z. Lv, and M. Zhang. Efficiently attribute-based access control for mobile cloud storage system. In *13th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom 2014*, pages 292–299. IEEE Computer Society, 2014. 132

[53] K. Chung, K. Yael Tauman, F. Liu, and R. Raz. Memory delegation. In *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference*, volume 6841 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 2011. 28, 86

[54] C. Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding, 8th IMA International Conference*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001. 56

[55] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006*, pages 79–88. ACM, 2006. 50, 61, 62, 64, 69, 73, 85, 128, 129, 131, 132, 134, 142, 143, 145, 146, 148, 149, 152, 155, 165, 166

[56] S. Devadas, C. W. Fletcher, L. Ren, E. Shi, E. Stefanov, M. van Dijk, and X. Yu. Path ORAM: an extremely simple oblivious RAM protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 299–310. ACM, 2013. 54, 70

[57] S. Devadas, M. van Dijk, C. W. Fletcher, L. Ren, E. Shi, and D. Wichs. Onion oram: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography - 13th International Conference, TCC 2016-A*, volume 9563 of *Lecture Notes in Computer Science*, pages 145–174. Springer, 2016. 54, 70

[58] G. Di Crescenzo and V. Saraswat. Public key encryption with searchable keywords based on jacobi symbols. In *Progress in Cryptology - INDOCRYPT 2007, 8th International Conference on Cryptology*, volume 4859 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2007. 56, 72

[59] F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1155–1166. ACM, 2016. 77, 134

[60] K. Emura, A. Miyaji, and M. S. Rahman. IACR Cryptology ePrint Archive, Report 2017/321, 2017. 57

[61] L. Fang, W. Susilo, C. Ge, and J. Wang. A secure channel free public key encryption with keyword search scheme without random oracle. In *Cryptology and Network Security, 8th International Conference*, volume 5888 of *Lecture Notes in Computer Science*, pages 248–258. Springer, 2009. 57

[62] D. Feng, Z. Lv, and M. Zhang. Multi-user searchable encryption with efficient access control for cloud storage. In *IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom 2014*, pages 366–373. IEEE Computer Society, 2014. 132

[63] M. Fischlin. Pseudorandom function tribe ensembles based on one-way permutations: Improvements and applications. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryp-*

tographic Techniques, volume 1592 of *Lecture Notes in Computer Science*, pages 432–445. Springer, 1999. 158

[64] K. Fu, S. Kamara, and Y. Kohno. Key regression: Enabling efficient key distribution for secure distributed storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2006*. The Internet Society, 2006. 131

[65] Z. Fu, J. Shu, X. Sun, and N. Linge. Smart cloud search services: verifiable keyword-based semantic search over encrypted cloud data. *Consumer Electronics, IEEE Transactions on*, 60(4):762–770, 2014. 126

[66] S. Garg, P. Mohassel, and C. Papamonthou. TWORAM: efficient oblivious RAM in two rounds with applications to searchable encryption. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference*, volume 9816 of *Lecture Notes in Computer Science*, pages 563–592. Springer, 2016. 54, 70

[67] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference*, volume 6223 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2010. 29, 86

[68] E.-J. Goh. Secure indexes. IACR Cryptology ePrint Archive, Report 2003/216, 2003. 61, 63, 69, 70, 134, 145, 146

[69] B. Goi, S. Heng, and W. Yau. Off-line keyword guessing attacks on recent public key encryption with keyword search schemes. In *Autonomic and Trusted Computing, 5th International Conference, ATC 2008*, volume 5060 of *Lecture Notes in Computer Science*, pages 100–105. Springer, 2008. 58

[70] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the Association for Computing Machinery*, 43:431–473, 1996. 52, 54

[71] P. Golle, J. Staddon, and B. R. Waters. Secure conjunctive keyword search over encrypted data. In *Applied Cryptography and Network Security, Second International Conference, ACNS 2004*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004. 62, 63, 134

[72] M. T. Goodrich, J. Z. Sun, and R. Tamassia. Efficient tree-based revocation in groups of low-state devices. In *Advances in Cryptology - CRYPTO 2004, 24th*

Annual International Cryptology Conference, volume 3152 of *Lecture Notes in Computer Science*, pages 511–527. Springer, 2004. 99, 100

[73] Guillevic. Comparing the pairing efficiency over composite-order and prime-order elliptic curves. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013*, volume 7954 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2013. 100

[74] D. Halevy and A. Shamir. The LSD broadcast encryption schem. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2002. 99, 100

[75] M. Hattori, T. Hirano, T. Ito, N. Matsuda, T. Mori, Y. Sakai, and K. Ohta. Ciphertext policy delegatable hidden vector encryption and its application to searchable encryption in the multi user setting. In *Cryptography and Coding - 13th IMA International Conference, IMACC 2011*, volume 7089 of *Lecture Notes in Computer Science*, pages 190–209. Springer, 2011. 131

[76] Y. H. Hwang and P. J. Lee. Public key encryption with conjunctive keyword search and its extension to a multi-user system. In *Pairing-Based Cryptography - Pairing 2007, First International Conference*, volume 4575 of *Lecture Notes in Computer Science*, pages 2–22. Springer, 2007. 57, 60, 131

[77] H. Imai and R. Zhang. Generic combination of public key encryption with keyword search and public key encryption. In *Cryptology and Network Security, 6th International Conference, CANS 2007*, volume 4856 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2007. 59

[78] Y. Ishai, E. Kushilevitz, S. Lu, and R. Ostrovsky. Private large-scale databases with distributed searchable symmetric encryption. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2016. 64

[79] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2012. 134

[80] N. Jho, J. Y. Hwang, J. H. Cheon, M. Kim, D. H. Lee, and E. S. Yoo. One-way chain based broadcast encryption schemes. In *Advances in Cryptology - EURO-*

CRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, volume 3494 of *Lecture Notes in Computer Science*, pages 559–574. Springer, 22005. 100

[81] Z. Jinsheng, Z. Wensheng, and D. Qiao. A multi-user oblivious ram for outsourced data. Iowa State University Digital Repository, Computer Science Technical Reports, 2014. 76

[82] P. Joong, K. Kim, and D. J. Park. Public key encryption with conjunctive field keyword search. In *Information Security Applications, 5th International Workshop, WISA 2004*, volume 3325 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2004. 57

[83] A. Kaci, T. Bouabana-Tebibel, and Z. Challal. Access control aware search on the cloud computing. In *2014 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2014*, pages 1258–1264. IEEE, 2014. 129, 132, 166

[84] S. Kamara and T. Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. IACR Cryptology ePrint Archive, Report 2017/126, 2017. 70

[85] S. Kamara and C. Papamonthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013*, volume 7859 of *Lecture Notes in Computer Science*, pages 258–274. Springer, 2013. 62, 69, 70, 134, 145, 146

[86] S. Kamara, C. Papamonthou, and T. Roeder. Dynamic searchable symmetric encryption. In *The ACM Conference on Computer and Communications Security, CCS'12*, pages 965–976. ACM, 2012. 62, 69, 134, 135, 145, 146, 152

[87] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* Chapman and Hall/CRC Press, 2007. 19, 20, 158

[88] J. Katz, C. Papamonthou, and Y. Zhang. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *25th USENIX Security Symposium, USENIX Security 16*, pages 707–720. USENIX Association, 2016. 59

[89] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology - EURO-*

CRYPT 2008, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008. 58, 63, 72, 131

[90] F. Kerschbaum. Frequency-hiding order-preserving encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 656–667. ACM, 2015. 64

[91] D. Khader. Public key encryption with keyword search based on k-resilient IBE. IACR Cryptology ePrint Archive, Report 2006/358, 2006. 57, 72

[92] Z. A. Kissel and J. Wang. Verifiable symmetric searchable encryption for multiple groups of users. In *Proceedings of the 2013 International Conference on Security and Management*, pages 179–185. CSREA Press, 2013. 85, 129, 131, 143

[93] K. Kurosawa and Y. Ohtaki. Uc-secure searchable symmetric encryption. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012*, volume 7397 of *Lecture Notes in Computer Science*, pages 285–298. Springer, 2012. 62, 69, 84

[94] K. Kurosawa and Y. Ohtaki. How to update documents verifiably in searchable symmetric encryption. In *Cryptology and Network Security - 12th International Conference, CANS 2013*, volume 8257 of *Lecture Notes in Computer Science*, pages 309–328. Springer, 2013. 125, 126

[95] D. H. Lee, J. H. Park, H. S. Rhee, and W. Susilo. Improved searchable public key encryption with designated tester. In *Proceedings of the 2009 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2009*, pages 376–379. ACM, 2009. 57

[96] J. Li, K. Ren, Z. Wan, and B. Zhu. Privacy-aware attribute-based encryption with user accountability. In *Information Security, 12th International Conference, ISC 2009*, volume 5735 of *Lecture Notes in Computer Science*, pages 347–362. Springer, 2009. 25

[97] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou. Fuzzy keyword search over encrypted data in cloud computing. In *INFOCOM 2010, 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 441–445. IEEE, 2010. 125

[98] K.-C. Li, J. Shen, J. Wang, and J. Zhao. Fine-grained searchable encryption in multi-user setting. *Soft Computing*, pages 1–12, 2016. 133, 166

[99] M. Li, S. Yu, N. Cao, and W. Lou. Authorized private keyword search over encrypted data in cloud computing. In *2011 International Conference on Distributed Computing Systems, ICDCS*, pages 383–392. IEEE Computer Society, 2011. 129

[100] P. V. Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker. Computationally efficient searchable symmetric encryptio. In *Secure Data Management, 7th VLDB Workshop, SDM 2010*, volume 6358 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2010. 61, 69, 70

[101] Y. Lindell. How to simulate it – a tutorial on the simulation proof technique. IACR Cryptology ePrint Archive, Report 2016/046, 2016. 33

[102] C. Liu, L. Zhu, M. Wang, and Y. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Information Sciences*, 265:176–188, 2014. 85, 134

[103] I. Mori. Cyber security breaches survey 2017. Technical report, Ipsos Mori and the University of Portsmouth, 2017. 13

[104] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001. 98, 99, 100

[105] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015. 64, 77

[106] M. Naveed, M. Prabhakaran, and C. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy, SP 2014*, pages 639–654. IEEE Computer Society, 2014. 69

[107] J. Ning, Z. Cao, X. Dong, L. Wei, and X. Lin. Large universe ciphertext-policy attribute-based encryption with white-box traceability. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security*, volume 8713 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2014. 100, 101

[108] T. Nishide, K. Ohta, and K. Yoneyama. Attribute-based encryption with partially hidden encryptor-specified access structures. In *Applied Cryptography and*

Network Security, 6th International Conference, ACNS 2008, volume 5037 of *Lecture Notes in Computer Science*, pages 111–129. Springer, 2008. 25

[109] H. of Commons. Investigatory powers bill. http://www.publications.parliament.uk/pa/bills/cbill/2015-2016/0172/160172.pdf, 2016 (accessed 06.05.16). 80

[110] C. Office. Goverment security classifications. Technical report, 2013. 128

[111] T. Okamoto and K. Takashima. Fully secure unbounded inner-product and attribute-based encryption. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security*, volume 7658 of *Lecture Notes in Computer Science*, pages 349–366. Springer, 2012. 100, 101

[112] M. Onen, R. Molva, and C. Van Rompay. Multi-user searchable encryption in the cloud. In *Information Security - 18th International Conference, ISC 2015*, volume 9290 of *Lecture Notes in Computer Science*, pages 299–316. Springer, 2015. 149

[113] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 1999. 78

[114] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. G. Choi, W. George, A. Keromytis, and S. Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy, SP 2014*, pages 359–374. IEEE Computer Society, 2014. 69

[115] B. Parno, M. Raykova, and V. Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439. Springer, 2012. 29, 86

[116] J. Pieprzyk, H. Wang, and P. Wang. Keyword field-free conjunctive keyword searches on encrypted data and extension for dynamic groups. In *Cryptology and Network Security, 7th International Conference, CANS 2008*, volume 5339 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2008. 63, 76

[117] R. A. Popa, C. Redfield, S. Tu, H. Balakrishman, F. Kaashoek, S. Madden, N. Zeldovich, and A. Burrows. CryptDB. https://css.csail.mit.edu/cryptdb, 2011 (accessed 06.06.16). 78

[118] R. A. Popa, C. Redfield, and N. Zeldovich. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011*, volume 85–100. ACM, 2011. 77

[119] PwC. 2015 information security breaches survey. Technical report, 2015. 81

[120] S. L. Renwick. Predicate encryption scenarios. Technical report, Thales Research and Technology (UK), 2014. 15, 66

[121] S. L. Renwick and K. M. Martin. Practical architectures for deployment of searchable encryption in a cloud environment. 2015. 15

[122] Y. Rouselakis and B. Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, pages 463–474. ACM, 2013. 100, 101

[123] Y. Rouselakis and B. Waters. Efficient statically-secure large-universe multi-authority attribute-based encryption. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015*, volume 8975 of *Lecture Notes in Computer Science*, pages 315–332. Springer, 2015. 100

[124] E. Ryu and T. Takagi. Efficient conjunctive keyword-searchable encryption. In *21st International Conference on Advanced Information Networking and Applications (AINA 2007)*, pages 409–414. IEEE Computer Society, 2007. 63

[125] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *Theory of Cryptography, 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2009. 63, 70

[126] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE, 2000. 60, 69, 78, 128

[127] E. Stefanov, C. Papamonthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014*. The Internet Society, 2014. 69, 85, 125, 126

[128] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou, and H. Li. Verifiable privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking. *IEEE Transactions on Parallel Distributed Systems*, 25(11):3025–3035, 2014. 125, 126

[129] W. Sun, S. Yu, W. Lou, T. Hou, and H. Li. Protecting your right: Verifiable attribute-based keyword search with fine-grainedowner-enforced search authorization in the cloud. *IEEE Transactions on Parallel Distributed Systems*, 27(4):1187–1198, 2016. 85, 126

[130] C. Wang, N. Cao, K. Ren, and W. Lou. Enabling secure and efficient ranked keyword search over outsourced cloud data. *IEEE Transactions Parallel Distributed Systems*, 23(8):1467–1479, 2012. 126

[131] J. Wang, H. Ma, J. Li, H. Zhu, S. Ma, and X. Chen. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer Science Information Systems*, 10(2):667–684, 2013. 126

[132] J. Wang, H. Ma, Q. Tang, J. Li, H. Zhu, S. Ma, and X. Chen. Efficient verifiable fuzzy keyword search over encrypted data in cloud computing. *Computer Science Information Systems*, 10(2):667–684, 2013. 124

[133] S. Wang, W. Yang, and Y. Lin. Balanced double subset difference broadcast encryption scheme. *Security and Communication Networks*, 8:1447–1460, 2015. 99, 100

[134] B. Waters. Ciphertext-policy attribute-based encryption: an expressive, efficient, and provably secure realization. In *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011. 103

[135] Y. Yang. Towards multi-user private keyword search for cloud computing. In *IEEE International Conference on Cloud Computing, CLOUD 2011*, pages 758–759. IEEE, 2011. 74, 130, 131, 132

[136] Y. Yang. Attribute-based data retrieval with semantic keyword search for e-health cloud. *Journal of Cloud Computing: Advances, Systems and Applications*, 4, 2015. 129

[137] Q. Zheng, S. Xu, and G. Ateniese. VABKS: verifiable attribute-based keyword search over outsourced encrypted data. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014*, pages 522–530. IEEE, 2014. 85, 124, 126