Natural Language Processing on Encrypted Patient Data

by

Alex J. Grinman

B.S., Mathematics, Computer Science, M.I.T., 2015

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

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Author_____
Department of Electrical Engineering and Computer Science
September 7, 2016

Certified by_____
Shafi Goldwasser
Professor
Thesis Supervisor

Accepted by_____
Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Natural Language Processing on Encrypted Patient Data

by

Alex J. Grinman

Submitted to the Department of Electrical Engineering and Computer Science
on September 7, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

While many industries can benefit from machine learning techniques for data analysis, they often do not have the technical expertise nor computational power to do so. Therefore, many organizations would benefit from outsourcing their data analysis. Yet, stringent data privacy policies prevent outsourcing sensitive data and may stop the delegation of data analysis in its tracks. In this thesis, we put forth a two-party system where one party capable of powerful computation can run certain machine learning algorithms from the natural language processing domain on the second party's data, where the first party is limited to learning only specific functions of the second party's data and nothing else. Our system provides simple cryptographic schemes for locating keywords, matching approximate regular expressions, and computing frequency analysis on encrypted data. We present a full implementation of this system in the form of a extendible software library and a command line interface. Finally, we discuss a medical case study where we used our system to run a suite of unmodified machine learning algorithms on encrypted free text patient notes.

Thesis Supervisor: Shafi Goldwasser
Title: Professor

Acknowledgments

I am extremely grateful to my advisor, Shafi Goldwasser, for helping me find a thesis project where I can apply my interests in theory to real-world problems. I am also very thankful for her advice on all aspects of our project.

I would also like to thank Dr. Charlotta Lindvall for helping me understand the importance of computational studies on patient data and the associated information privacy problems preventing widespread access to medical data. I am also grateful to Josh Haimson for his support in helping me understand and use their machine learning algorithm to search on patient data.

I would also like to thank the Akamai Theory Fund, award number 1443501, and the Electrical Engineering and Computer Science Research Assistantship Fund, award number 1704400, for providing financial support as I worked on this thesis.

I would also like to thank my close friend Kevin King for his constant helpfulness and willingness to bounce ideas around.

Finally, I would like to thank my family, especially my father Vlad, for their support and eagerness to help me with any personal or even technical problem that came up.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As machine learning algorithms make their way into more fields and industries, the problem of data privacy becomes increasingly more important. In practice, in order for machine learning algorithms to produce effective results, an extremely large dataset is needed. Therefore, a powerful computational cluster is required to handle the immense amount of data computation. To bypass this problem, organizations often need to delegate their machine learning computations to another, possibly untrusted, party. Computational power is only one reason to delegate computation on your data. In some cases, organizations do not even know which machine learning algorithms to run on their data and want third parties to not only run machine learning algorithms on their data but also determine which algorithms to run. This makes the data privacy problem even more difficult. Organizations want to delegate the ability to run arbitrary machine learning algorithms but want to keep any sensitive, personally identifiable data private.

To address the data privacy problem, current industry best practices suggest methods for "anonymizing" data, or blacklisting certain types of data points. For example, a financial institution might remove all sequences of numbers that look like Social Security Numbers. A medical institution might remove all names, identification numbers, and birthdays. While this method might be simple to employ in structured data where identifiable information is easily removable, many machine learning algorithms are most effective on unstructured or "free text" data. Anonymizing these types of documents proves to be very difficult as private information can be hidden in any sentence throughout a large body of human-written text. For example, financial advisers might manually write account notes about a customer in a bank record. A doctor often manually writes notes describing patient symptoms or feelings. Some best practices for anonymizing data therefore suggest to manually read free text data to find and remove personally identifiable information. Since these datasets are very large, it isn't always feasible for organizations to manually remove these types of sensitive data points.

1.1 Overview of Natural Language Processing (NLP) Primitives

In this work, we consider certain machine learning algorithms from the Natural Language Processing (NLP) domain. Informally, NLP is a set techniques for analyzing and deriving meaning from human language. Specifically, NLP algorithms take free text as input and output one or more classifications about the free text. NLP algorithms are usually specialized to specific types of input data and output classifications. For example, given a paragraph describing a person's review of a movie, an NLP algorithm might determine if the review is positive or negative. NLP algorithms use a variety of methods to process free text. Specifically, our work considers the following NLP methods: keyword search, regular expression matching, and bag-of-words.

Keyword Search. One of the simplest methods to classify free text is to check if the free text contains a keyword. The presence of specific keywords might indicate the meaning of the text. For instance, a positive a movie review might contain keywords like "great", "best", or "interesting", while a negative review might contain keywords like "bad", "worst", or "boring". Therefore many NLP algorithms scan free text for well known keywords, and use this information as one of the factors in classifying the free text.

Regular Expression Matching. As with keywords, the presence of specific phrases or expressions in free text may reveal the meaning of the free text. A regular expression is a string of characters that encodes a specific search pattern. Regular expressions are particularly useful for identifying a known phrase or a sequence of characters that can be written many ways. For example, the regular expression `((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}` can be used to match all US phone numbers. Another regular expression might be used to match the many ways of writing numerical ratings in movie reviews like "5/10", "5 out of 10", or "five out of ten". Therefore, many NLP algorithms scan free text to check for matches against well known regular expressions and use this information as another factor in classifying the free text.

Bag-of-Words. The bag-of-words algorithm, also known as frequency analysis, takes free text as input and maps each distinct word to the number of times it repeats in the free text. For example, the free text "Alice likes fast cars and Bob likes red cars" would produce the following bag-of-words:

```
"likes": 2
"cars" : 2
"Alice": 1
"Bob"  : 1
```

```
"fast" : 1
"red"  : 1
```

In addition to distinct words, bag-of-words can also be applied to "n-grams" where an n-gram is an ordered list of n words. That is, an n-gram bag-of-words algorithm takes free text as input and maps each distinct ordered tuple of n words to the number of times this ordered tuple of n ordered words appear in the free text. For example, the free text "Alice likes driving fast but Bob hates driving fast" would produce of the following bag-of-words bi-gram:

```
"Alice likes"  : 1
"likes driving": 1
"driving fast" : 2
"fast but"     : 1
"but Bob"      : 1
"Bob hates"    : 1
```

Bag-of-words are useful for determining which words or n-grams (phrases) appear most commonly throughout multiple free text documents. Many NLP algorithms uses this information as a factor in deciding that two free text documents are similar and therefore should have the same classification.

1.2 Our Contributions

In this work we design and implement a cryptographic two-party system where one party can outsource common natural language processing computations to a computationally powerful, partially untrusted second party. Specifically, our system enables the computationally powerful party to perform keyword search, approximate regular expression matching, and bag-of-words computations on a second party's encrypted free text data. Our contributions are as follows.

- **Two-party Model.** We define a two-party computation model for outsourcing keyword search and bag-of-words natural language processing computations on encrypted data and motivate it with practical examples. We outline the privacy goals that this system aims to achieve. We also define the three main operational phases of our system, the free text encryption phase, the auxiliary key request phase, and the compute phase to show how the system would work in a practical setting.

- **Definitions of Privacy.** We provide formal privacy definitions for cryptographic schemes that perform keyword search and bag-of-words computations on encrypted free text. Essentially, we require that for any encrypted free text the leaked information constitutes only the locations of allowed keywords in keyword search and the

frequency counts of underlying plaintexts (where underlying plaintexts themselves are not leaked) and nothing else.

- **Cryptographic Constructions.** We provide cryptographic constructions for performing keyword search and bag-of-words computations on encrypted free text.
- **Proofs of Privacy.** We prove that our cryptographic construction achieve our definitions of privacy under the Random Oracle Model and the existance of a Pseudo-random Permutation. Looking forward, this will be implemented using standard hash functions such as SHA-256 and block ciphers such as AES.
- **Implementation.** We provide a complete software implementation our cryptographic constructions as well as a command line interface to make our software usable in practice.
- **Medical Case Study.** Finally, we show how we used our software to run a suite of unmodified natural language processing algorithms on encrypted patient data and how it achieved the same results as running the natural language processing algorithms on plaintext patient data.

In the remainder of this chapter, we will define the basic two party model as stated above, explain our results, and provide a roadmap for the rest of the thesis, and discuss related work.

1.3 The Two-Party System

In our system there are two parties, (1) the data **owner** and (2) the data **learner**. The owner has control over some set of free text documents and the learner is tasked with performing keyword search and bag-of-words computations on the documents. However, the owner's data contains personally identifiable information, denoted PII, that only the owner is privileged to know. Therefore, the owner must have the learner perform these computations without giving the learner the ability to discover any PII. The learner is assumed to operate in the *honest but curious* paradigm. The learner is curious and therefore wants to learn as much as possible, but the learner is honest and will not share the results of their computations with unauthorized parties, including other leaners.

In an ideal privacy definition, we would like to state that the owner hides all PII from the learner while giving the learner the ability to perform keyword searches and bag-of-words computations. However, this definition would be seemingly unachievable because it seems hard to formally define what personally identifiable information is. It might be that a keyword looks harmless but it actually identifies a person. Therefore, we must be more specific with our definition of privacy. We say that a keyword search scheme is private if

the learner can detect the locations of keywords in an encrypted free text document for keywords that the owner allows and nothing else. We say that a bag-of-words computation scheme is private if the learner can detect the locations of repeating plaintexts and nothing else. This definition also requires that the plaintext itself is not revealed and instead a unique placeholder for every plaintext is used in place.

One motivating case for our model is a hospital (data owner) and external medical research scientists (data learners). The hospital owns many patient records, each of which contains structured and unstructured data. The researchers wish to examine patient records and learn about the possible links between certain diseases and symptoms. The researchers have computational resources to perform execute such algorithms, and the hospital wants to aid the researchers to benefit from their results.For example, to show the effectiveness of computational studies on electronic health records, a team of medical research scientists used natural language processing techniques to diagnose bipolar patients [19]. They ran these natural language process algorithms over a dataset with about 5 million patients, and achieved a high success rate for correctly classifying patients with bipolar disease.However, this study is only the very beginning. By making valuable data sets like these available, data owners and learners can collaborate to make big discoveries that otherwise could have gone unnoticed.

1.3.1 The Basic model

Our system has three main operational phases. In the encrypt phase, the data owner encrypts a free text document and publishes the resulting ciphertexts. Next in the request phase, a data learners can then request auxiliary keys for specific keywords or one auxiliary key for bag-of-words. Finally in the compute phase, a learner uses these auxiliary keys to find the locations of keywords and count repetitions of underlying plaintext in the encrypted free text document.

Phase 1 - Encryption

The data owner encrypts its free text documents and posts the resulting encrypted free text to a public location where data learners can read it. Once a free text document is encrypted and published, it never needs to be re-encrypted. This is important because the data owner is not computationally powerful and therefore should not be iterating over free text data often. Encryption is especially efficient in the case of multiple data learners since the owner does not need to encrypt free text individually for each learner.

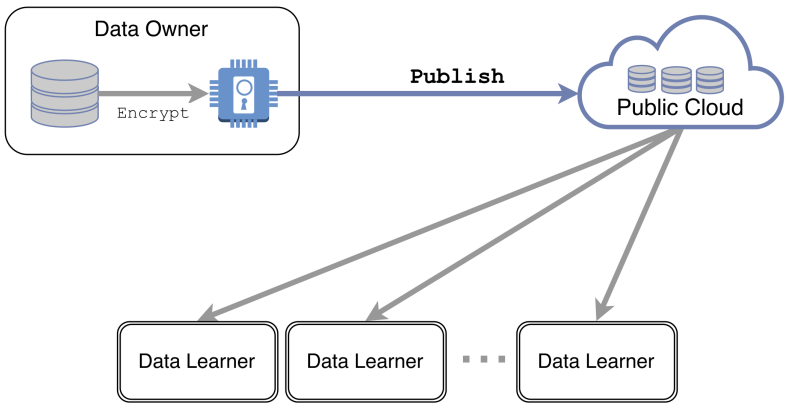

Figure 1-1: The data owner encrypts then publishes. Multiple data leaners pull encrypted free text records.

Phase 2 - Auxiliary Key Request

As shown in figure 1-2, learners request auxiliary keys for specific keywords and for bag-of-words (frequency count) from the owner. The owner has the option to approve or reject these auxiliary key requests. For example, the owner might reject a request if the keyword looks like a person's name or something clearly identifiable. The owner might also reject because the learner is not authorized to search for the supplied keyword or the learner is not authorized to compute a frequency count of underlying plaintext words. If the owner approves the request, the corresponding auxiliary keys are sent to the sender.

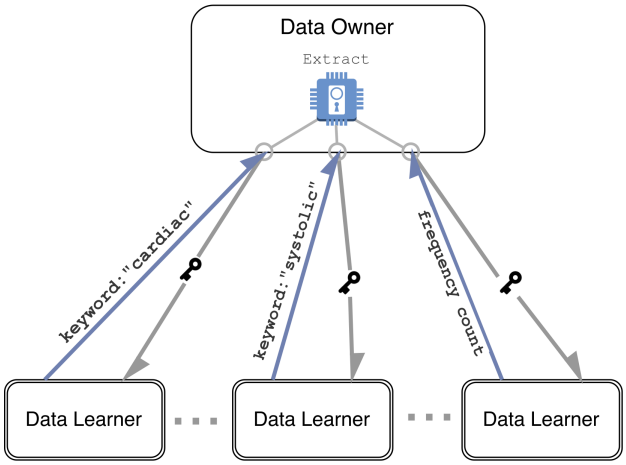

Figure 1-2: Data learners request search keys. For example, the first data learner asks for a search key to detect all ciphertexts that are encryptions of "cardiac". The data owner extracts the search key (to approve the request) and returns it to the learner.

Phase 3 - Compute

Using the encrypted free text from phase 1 and the auxiliary keys from phase 2, the learner will perform the keyword search and bag-of-words computations on the encrypted free text. As shown in figure 1-3, the learner will discover the locations of of the requested keywords in the encrypted free text. The learner will also discover the locations of repeating plaintexts in the encrypted free text, where each distinct plaintext is marked by a unique placeholder.

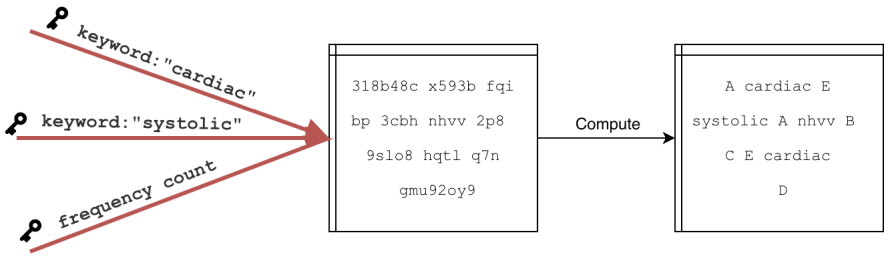

Figure 1-3: Applying the auxiliary keys enables the learner to transform encrypted free text data to reveal locations of keywords for corresponding keys ("cardiac" and "systolic" in this example) and compute bag-of-words by revealing locations of repeating plaintexts. A,B,C,D,E are placeholders to represent unique plaintext words.

Remark 1.3.1. We do not require a fixed order execution of these phases and each phase can be performed an arbitrary number of times as long as the parties are still willing to participate. Therefore, data owners can encrypt new data as it is created, and learners can compute over this data using their existing auxiliary keys. Learners can also request new auxiliary keys as their learning criterion changes based on possible intermediate findings.

Remark 1.3.2. As an extension of the basic model, the system could support further interaction between the owner and the learner, where the owner sends the learner some bounded auxiliary information about the ciphertext to the learner. This can also be formally incorporated into our security definitions. Looking forward, this is something that our implementation takes advantage of. Specifically, our implementation of bag-of-words computations supports a protocol where the learner can ask the owner to reveal the plaintext corresponding to a specific placeholder.

1.3.2 Public vs Private Encryption

It is important to note that our system is designed to operate in the private-key encryption model. This means that only the data owner can encrypt free text. However, there exist cryptographic systems that operate in the public-key encryption model, where the data

owner holds onto a private-key but any other party like the data learner would be given the public-key which they can use to encrypt free text words. Such a scheme could be useful if there were multiple data owners. However, in this work we are primarily concerned with models where only one party controls all data but several computationally more powerful parties may want to search on the data. One reason for this decision is that we discovered that there exist many applications that do not need the public-key encryption setting. As a motivating example, we can consider hospital data systems. While each hospital collects its own records, the hospital data systems are usually powered by a single large provider. One such example is the Partners Healthcare Organization.

This relaxation allows us to design a simpler and more efficient system based only on private-key (symmetric) cryptographic primitives. In fact, as we will later show, our schemes rely on practical, widely-used symmetric encryption primitives that enables us to achieve a high level of performance. This efficiency is key, since our primary goal is to enable outsourcing machine learning computations over vast datasets.

1.4 Implementation of our System

We present an implementation of our system in the form a command line interface, named Alvis, and a software library that can both be interfaced to work with arbitrary suites of NLP algorithms. Our implementation is in the `Golang` programming language and works on most operating systems. The source code can be found on `https://github.mit.edu/agrinman/alvis`.

The command line interface is used as a standalone, deployable application to generates keys, encrypts free text, requests auxiliary keyword and bag-of-words keys, and performs keyword search and bag-of-word computations on encrypted free text. The software library enables developers to programmatically integrate and use auxiliary keys to perform data analysis on encrypted data in their custom NLP algorithms. Together, the command line interface and the software library allow for both direct and programmatic use of our system.

Finally, we show that our implementation is efficient. In section 4.4.1 we give detailed performance benchmarks for each of our functions. Most importantly, we show that the bottle neck of our system is performing AES (block cipher) operations. That is, each invocation of our function performs about as fast as an AES operation. In today's computing environments, AES is often implemented as a hardware instruction which allows our system to perform significantly better on modern hardware.

1.5 Computing on Encrypted Patient Data

We used our implementation to perform Natural Language Processing (NLP) on encrypted free text patient notes. We were inspired by a partnership between MIT and MGH that sought out to learn about patients with heart failure problems that received Cardiac Resynchronization Therapy (CRT). While this a successful therapy for a majority of patients, about one third of CRT patients do not experience positive results [14]. More interestingly, the causes of failure are not well understood [14].

One reason for the difficulty in understanding CRT failure conditions is the way that the clinical results are recorded in patient records. Record keeping, while electronic, leaves a lot to be desired for recording specific patient results, especially during CRT treatment. Data is stored in many formats, and structured data often only contains a limited number of important metrics. This means that the bulk of information, which could potentially reveal CRT failure reasons, is hidden in free text doctor notes. Thus, clinical researchers would need to manually read this data to determine causes, a task that is infeasible for a large number of patient records [16].

Freel, Haimson, and Traub from the Massachusetts Institute of Technology (MIT), and a clinical research doctor, Lindvall from the Massachusetts General Hospital (MGH) [16] developed a suite of NLP algorithms, denoted FHTL, that improved the prediction accuracy for the success of Cardiac Resynchronization Therapy, a treatment that fails for one third of patients, by 9% [16].

We framed this medical case study in our two party model. Naturally, the hospital is the data owner, maintaining a set of patient records. The MIT NLP researchers, collaborated with a medical institution, like MGH, to learn about why Cardiac Resynchronization Therapy treatments fail and how to predict their success.

We use our system implementation to simulate the FHTL program on encrypted free text patient notes. Our main results is that we can use to run the unmodified suite of NLP algorithms on a encrypted patient data while achieving the same results as if the algorithms were run on unencrypted data. These results are based on running trials of our software and the FHTL algorithm on only a small subset of the original patient data files. The original work by Freel et al [16] used about 900 patient data files, while we were limited to 10 (deceased) patient files due to access restrictions and privacy regulations. We note that FHTL was designed to run on small datasets [16] and that our trials indicate that the our software will work on larger data sets, however more auxiliary keyword keys may be needed to show that the execution of FHTL on encrypted data classifies as well as it would on plaintext data.

One our motivations for developing this system is to give outside researchers access to large patient data sets without leaking the personally identifiable information of patients. Coincidentally, our limited access to the patient data set is a perfect example of why need methods for preforming data analysis on encrypted patient data.

1.6 Thesis Roadmap

In the following chapter, we define the cryptographic primitives used by our work, including the security assumptions that we will make in order to prove the privacy of our schemes. Next, in chapter 3, we provide the formal definitions for our schemes to perform keyword search and bag-of-words computations on encrypted data and we give formal definitions of privacy for both of these schemes. We then provide the constructions for these schemes based on the cryptographic primitives from chapter 2, and we prove that these constructions meet the privacy definitions. In chapter 4, we describe the practical implementation our constructions and demonstrate both a command line interface that is easy to use and software library that is extendible and simple to integrate into existing programs. We provide a performance analysis of our construction to show that it is highly efficient. In chapter 5, we present a detailed medical case study where we use our software to run a suite of unmodified natural language process algorithms on encrypted patient data, achieving the same results as if the algorithms were run on plaintext patient data. Finally, we conclude the thesis by describing next steps and future work goals.

1.7 Related Work

There has been substantial work in topics related to searching on encrypted data. In this section we describe several types of searchable encryption models and compare these existing solutions to our work. Overall, most of the related work is based on advanced cryptographic primitives such as Bilinear Maps and Learning With Errors, while our system is based on simple primitives like hash functions and pseudorandom permutations that are efficient and heavily used, which enables us to fulfill the end goal of this work, to build this system and use it in practice.

1.7.1 Client-Server Model: Symmetric Searchable Encryption

Much of the work on searchable encryption focuses on client-server model where the goal is for the client to encrypt a database while the server maintains it and can perform delegated searches for the client. Most constructions enable keyword search on the encrypted data.

This model is similar to our system in that the server is an adversary and it is more computationally powerful. The server is therefore tasked with searching records. Client's

can create a type of decryption key that allows the server to find matching database records. The objective is for the server to learn that the a record matches and returns it to the client who can then decrypt. Thus, the server effectively learns only that some records matched an unknown query. In our system, the goal is for the computationally powerful party to learn significantly more information, like locations of keywords and detections of repeating underlying plaintexts, which would enable this party to compute more complicated natural language processing algorithms over encrypted data.

Curtmola, Garay, Kamara, and Ostrovosky [15] provide a detailed discussion of efficient searchable encryption constructions based on symmetric encryption that work best in the client-server model. Much of their work establishes security definitions of various constructions for symmetric searchable encryption. Our work differs in that we present a collaborative, interactive protocol that is designed to leak more information like a frequency search.

Pandey and Rouselakis [20] introduced the new concept of Property Preserving Encryption (PPE) where a "Test" procedure can be executed on ciphertexts to determine if the underlying plain-text has some property. They present a symmetric construction for preserving the orthogonality property for vectors. This framework fits our model but currently their constructions are based on bilinear maps which prove to be to inefficient for the large data volume that our work considers. Additionally, we focus on simpler properties that we have observed are the main search primitives for effective machine learning algorithms.

1.7.2 Public-key Encrypted Keyword Search

In the Public-key Encrypted Keyword Search (PEKS) model by Boneh et al [11], multiple clients communicate through one or more servers. Therefore multiple parties must be able to encrypt. One common use case for this model is a mail server used by multiple parties. Emails must be confidential between parties, but each party should be able to delegate searching an email for keywords to the mail server. One way to construct PEKS is to use Identity Based Encryption (IBE). IBE was first introduced by Shamir [22] and the first construction was produced by Boneh and Franklin [12] using Pairing Based Cryptography. More so, the Boneh-Franklin IBE construction is anonymous which ensures that ciphertexts do not reveal any information about the underlying identity (keyword). Unfortunately, as we mentioned above, pairings are slow operations and limit usability when applied many times over large data sets. Additionally, we emphasize again that our model does not require the public encryption aspect for which PEKS is designed. Removing the public aspect allows us to design a scheme based on simpler primitives and gain far better performance.

1.7.3 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE), by Gentry [17], is a general solution for computing on encrypted data. This scheme is the king of all solutions as it would allow the evaluation of arbitrary machine learning algorithms over encrypted data. However, current constructions are currently extremely inefficient and would not feasible for the large data volume our work considers.

1.7.4 Functional Encryption

Function Encryption (FE), first formalized by Boneh, Sahai, and Waters [13], uses generalizations of IBE for placing arbitrary functions in the place of identity evaluation or attribute matching. Hence, FE is a framework that enables an identity to decrypt a function of the plain-text, where the function is taken over the ciphertext and the identity key. There are several definitions of security and proposals for FE with specific functions and general functions, where functions are expressed as circuit components with n-bit inputs. In the literature, there are constructions of FE schemes for functions such as inner product predicate [13] and Goldwasser et al [18] put forth the first FE scheme for general functions. However, all existing proposals either do not support multiple functions or are very inefficient as they use math that goes beyond simple computations such as hash functions. An interesting case is FE for Regular Languages.

FE for Regular Languages. Most of our preliminary work focused on creating a FE scheme to accomplish regular expression matching for general languages, one of our major unsolved search primitives.

We noticed that in our two party model, the data owner is the only party that ever needs to encrypt. This is fundamentally simpler than FE schemes in the public-key model. The public-key model seems to be at least as hard as the private-key model, because a construction for the public-key system could keep the public-key secret. However, most of the FE schemes we encountered were in the public-key model because they are based on generalizations of IBE and ABE schemes that are intentionally designed to support public encryption. For example, one interesting FE construction we came across is Functional Encryption for Regular Languages scheme by Waters [23]. This construction almost fits the requirements for regular expression matching, except, the scheme falls short because it only works in the public-index model where the underlying attributes are not kept private. Unfortunately, the "attributes", the words that regular expressions are evaluated on, are made public in Water's scheme. Therefore, Water's scheme doesn't protect the actual free text words which we seek to keep private.

The scarcity of private-key FE schemes led us to believe that it could be easier to construct

a private-key FE scheme for matching regular expressions. Specifically, we attempted to create a private-key FE scheme $\mathcal{FRE} = (Setup, Extract, Encrypt, CheckMatch)$, a tuple of four algorithms that work as follows.

- *Setup* creates the master secret key to be kept private throughout.
- *Extract* takes a Deterministic Finite Automaton (DFA) D that accepts a regular language $\mathcal{L}(D)$, and the master secret key, to derive a functional secret key sk_{f_D} fixed on D, where
$$f_D(x) = \begin{cases} 1 & \text{if } x \in \mathcal{L}(D) \\ 0, & \text{otherwise} \end{cases}$$
- *Encrypt* takes a string x and the master secret key to create a ciphertext c.
- $CheckMatch$ takes a ciphertext c corresponding to an encryption of some string x, and a functional secret key sk_{f_D} to compute f_D over the plaintext. That is, $CheckMatch(c, sk_{f_D}) = f_D(x)$.

We tried to adapt Water's public-index/public-key scheme to protect the underlying plaintext words. However, our main difficulty in constructing such a scheme is related to the sequential, character by character approach that a DFA uses to eventually either accept or reject an input string. That is, to functionally evaluate a DFA on a ciphertext it seems that encryption must individually, sequentially protect each character of the plaintext. However, if the DFA representing the regular language is also revealed by the functional secret key, then an adversary might be able to learn about the plaintext based on where the functional DFA evaluation fails on some ciphertext.

While we were not able to create a functional encryption scheme for evaluating general regular expressions, we were still able to solve the problem in a practical way. Since our keyword search is fast, our system can support searching with many keyword search keys. Therefore, for a simple enough, finite, regular language we can simply generate the most popular strings in the language and split them into keywords. Thus the matching of a subset of keywords represents matching an approximated regular expression.

Chapter 2

Notation and Cryptographic Primitives

In this chapter we describe the cryptographic primitives and assumptions used by our work. We start by defining some helpful notations.

2.1 Notation

Free text. An ordered list of words $W = [w_0, w_1, \ldots, w_n]$, where each word $w_i \in \{0,1\}^*$, is denoted as free text. Encryption of free text denotes an order list containing encryptions of each w_i. Often we denote a set of free text, which is a set of ordered lists, as $\mathcal{WL} = \{W_i\}$ and denote the set of all possible free texts as $\mathcal{WL}^*$.

Concatenation. Let $x, y \in \{0,1\}^*$. Then $x||y$ denotes the bit string concatenation of x and y.

Random Sampling. Let $r \xleftarrow{\$} \mathcal{S}$ denote selecting $r \in \mathcal{S}$ uniformly at random. For example, if $\mathcal{S} = \{x | x \in \{0,1\}^n\}$, then $\Pr[r | r \xleftarrow{\$} \mathcal{S}] = \frac{1}{2^n}$.

Oracle. An oracle machine, often denoted by $\mathcal{O}$, is an abstract Turing machine that can only be accessed using a black-box interface. Often, other turing machines can be given access to an oracle, to which they can only send inputs and receive outputs without seeing or modifying the underlying construction of the oracle machine.

Negligible Functions. A function $\mu(\cdot)$ is negligible function if and only if for all constants $c \in \mathbb{N}$ there exists an $x_0 \in \mathbb{N}$ such that:

$$\forall x > x_0 : |\mu(x)| < \frac{1}{x^c}$$

Let $\mathsf{negl}(x)$ denote a negligible function in x for some $c, x_0 \in N$.

Negligible Advantage. For some negligible function $\mathsf{negl}(n)$, an random variable $e \in \{0,1\}$ has negligible advantage if

$$\Pr[e = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

Polynomial Functions. $\mathsf{poly}(x)$ denotes an unspecified polynomial function in x where there exists a constant c such that $|\mathsf{poly}(x)| < x^c$.

$\mathcal{RF}_n$ **(Random Function Family).** Denote the uniform distribution over the set of all functions from domain $\{0,1\}^n$ to range $\{0,1\}^n$ as $\mathcal{RF}_n$.

$\mathcal{RP}_n$ **(Random Permutation Family).** Denote the uniform distribution over all permutations from domain $\{0,1\}^n$ to range $\{0,1\}^n$ as $\mathcal{RP}_n$.

F_k **(Keyed Function.)** Given a function $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$, the function $F_k : \mathcal{D} \to \mathcal{R}$ is derived by fixing F with some "key" $k \in \mathcal{K}$, such that $F_k(\cdot) = F(k, \cdot)$.

2.2 Cryptographic Hash Functions

Cryptographic hash functions map arbitrary length bit string inputs to fix sized outputs. Cryptographic hash functions are used to transform long messages into short digests that can be used to provide integrity for the full message. Digests are often included alongside encrypted messages to detect any modifications to the encrypted message.

Definition 2.2.1. A hash function $h : \{0,1\}^* \to \{0,1\}^m$, for $m > 0$, is cryptographic if it satisfies the following properties:

- Pre-Image Resistance. Given any $y \in \{0,1\}^m$ it is computationally infeasible to find any pre-image x such that $h(x) = y$.
- Second Pre-Image Resistance. Given $x \in \{0,1\}^*$, it is computationally infeasible to find $x' \neq x$ such that $h(x') = h(x)$.
- Collision Resistance. It is computationally infeasible to find any x, x' such that $x \neq x'$ and $h(x) = h(x')$.

Pre-image resistance means that given the output, it is computationally difficult to construct the input. Hence, given the hash of some data, an adversary will not be able to reconstruct the original data. This property is often known as "one-way"ness. Second pre-image resistance ensures that it is difficult to find a different message that hashes to a target message. Finally, collision resistance is a stronger notion of second pre-image resistance that ensures it is difficult to find *any* two messages that hash to the same output.

2.2.1 Secure Hashing Algorithm 2 (SHA-256)

SHA256 is a hash function, designed National Security Agency, and standardized by the National Institute for Standards and Technology [21].

Definition 2.2.2. SHA256 is a hash function,

$$\mathsf{SHA256} : \{0,1\}^* \rightarrow \{0,1\}^{256}$$

SHA256 is widely used today in many practical systems [6]. For the security of our work, we assume that this hash function meets the definition of 2.2.1.

Assumption 2.2.1. SHA256 is a cryptographic hash function.

In our security proofs, we will go one step further and replace SHA256 with a random oracle and prove that our system is secure in random oracle model.

2.3 Random Oracles

To prove the security of our cryptosystems that rely on cryptographic hash functions, we need the concept of random oracle.

Definition 2.3.1. A *Random Oracle* is an oracle maps inputs $x \in \{0,1\}^*$ to outputs in $y \in \{0,1\}^l$, where each bit of y is chosen uniformly and independently at random and l is sufficiently long [10].

In other words, a random oracle is an oracle that on every unique input responds with a truly random, unique output. On repeated inputs, the oracle responds with the same, previously generated, outputs. In the work Mihir and Rogaway [8], the theoretical random oracle has infinite length outputs, where the length can be truncated to a desired length l. Typically, a random oracle is made available to all parties in cryptosystem and is not kept private.

2.3.1 Random Oracle Model

Cryptosystems that rely on cryptographic hash functions are often proven in the *Random Oracle Model* to capture the requirements of an ideal cryptographic hash function. Therefore, random oracles are used as the ideal replacement to cryptographic hash functions in

security proofs. Hence, a system that is provably secure when cryptographic hash functions are replaced with random oracles is known as a system that is secure under the Random Oracle Model.

2.4 Symmetric-Key Encryption

The symmetric-key encryption primitive is fundamental for establishing private communication channels between two or more parties. An encryption scheme is symmetric when the same cryptographic key is used for both encryption and decryption.

Definition 2.4.1. A *symmetric-key encryption scheme* [9, Chapter 4] is a tuple of three PPT cryptographic algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ where

- Gen is a randomized key generation algorithm that takes a security parameter 1^k as input and returns $sk \xleftarrow{\$} \{0,1\}^k$, denoted as $sk \leftarrow Gen(1^k)$.
- Enc is a possibly randomized or deterministic encryption algorithm that takes as input a key $sk \in \{0,1\}^k$ and $m \in \{0,1\}^*$, and returns a ciphertext $c \in \{0,1\}^*$, denoted as $c \leftarrow \mathsf{Enc}_{sk}(m)$.
- Dec is deterministic encryption algorithm that takes as input a key $sk \in \{0,1\}^k$ and $c \in \{0,1\}^*$ and returns a message $m \in \{0,1\}^* \cup \{\perp\}$, denoted as $m \leftarrow Dec_{sk}(c)$.

and $\forall sk \leftarrow \mathsf{Gen}(1^k), m \in \{0,1\}^*, \mathsf{Dec}_{sk}(\mathsf{Enc}_{sk}(m)) = m$.

2.4.1 Definitions of Privacy

In order to define the security of a symmetric-key encryption scheme, we must first describe the adversarial model. Namely, what abilities does the adversary possess? The security definitions below will establish ciphertext indistinguishability under varying adversarial abilities. Ciphertext indistinguishability says that an adversary will not be able to distinguish ciphertexts based on the messages they decrypt to.

First, we can consider other types of security definitions. For example, a requirement that the encryption key is never revealed or that no ciphertexts can be decrypted without knowledge of the key. However, these definitions do not address the fundamental issue at hand, the adversary should not be able to learn anything about the underlying message by only seeing ciphertexts. The indistinguishability of ciphertexts provides a stronger notion of privacy; an adversary that cannot distinguish between ciphertexts of any message will be not be able to determine the decryption of a ciphertext nor the secret key.

Next, we will formally define different models of security for the indistinguishability of ciphertexts and the adversarial powers associated with them.

Indistinguishability under Chosen-Plaintext Attack ($\mathsf{IND}-\mathsf{CPA}$)

Under chosen-plaintext attack the adversary has access to an encryption oracle and then is given the ability to chose a pair of previously un-queried messages where the challenger randomly encrypts one these messages. To win the game, he adversary must correctly guess which message the ciphertext is an encryption of. The adversary is limited to a probabilistic polynomial-time Turing machine. Now we formally present the initialization, querying, and challenge phases of the security game.

1. INITIALIZATION. The challenger generates $sk \leftarrow \mathsf{Gen}(1^k)$. The secret key sk remains secret to the challenger.

2. QUERYING. The adversary chooses polynomially many (in k) messages

$$M = \{m_0, \ldots, m_{\mathsf{poly}(k)}\}$$

   The adversary receives encryptions for each message,

$$C = \{\mathsf{Enc}_{sk}(m_0), \ldots, \mathsf{Enc}_{sk}(m_{\mathsf{poly}(k)}\}$$

   Note that the adversary can receive encryptions one at a time, and dependently choose the next message to query.

3. CHALLENGE. The adversary chooses two messages (m_0, m_1). The challenger privately selects $b \xleftarrow{\$} \{0,1\}$, and returns $c_b \leftarrow \mathsf{Enc}_{sk}(m_b)$ to the adversary. The adversary can continue querying by choosing polynomially many (in k) more messages and receive encryptions for each message. The adversary responds to the challenge outputting $b' \in \{0,1\}$ to guess the value of b. If $b' = b$ the adversary wins the game, otherwise the adversary loses.

Definition 2.4.2. A symmetric-key encryption scheme is said to be secure under *Indistinguishability under Chosen-Plaintext Attack* if for all PPT adversaries $\mathcal{A}$,

$$\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(k)$$

where the probability is taken over the random coins of the challenger to select sk and the random coins of the adversary to choose messages M and m_0, m_1. Thus, the adversary has a negligible advantage of guessing which message corresponds to the challenge encryption.

Indistinguishability under Adaptive Chosen-Ciphertext Attack (IND-CCA(2))

Next, we define a similar security definition, where the main difference is that the adversary is given polynomially many queries to both encryption and decryption oracles. The adversary then similarly chooses a pair of challenge messages. The challenger randomly encrypts

one these messages, after which the adversary must guess which message the ciphertext is an encryption of. The adversary is once again limited to a probabilistic polynomial-time Turing machine. Now we formally present the initialization, querying, and challenge phases of the security game.

1. INITIALIZATION. The challenger generates $sk \leftarrow \mathsf{Gen}(1^k)$. The secret key sk remains secret to the challenger.

2. QUERYING. The adversary chooses polynomially many (in k) messages

$$M = \{m_0, \ldots, m_{\mathsf{poly}(k)}\}$$

and ciphertexts

$$C = \{c_0, \ldots, c_{\mathsf{poly}(k)}\}$$

The adversary then receives encryptions each of message:

$$E = \{\mathsf{Enc}_{sk}(m_0), \ldots, \mathsf{Enc}_{sk}(m_{\mathsf{poly}(k)}\}$$

and decryptions of each ciphertext:

$$D = \{\mathsf{Dec}_{sk}(c_0), \ldots, \mathsf{Dec}_{sk}(c_{\mathsf{poly}(k)})\}$$

Note that the adversary can receive encryptions and decryptions one at a time, and dependently choose the next query.

3. CHALLENGE. The adversary chooses two messages (m_0, m_1). The challenger privately selects $b \xleftarrow{\$} \{0,1\}$, and returns $c_b \leftarrow \mathsf{Enc}_{sk}(m_b)$ to the adversary. The adversary can continue querying by choosing polynomially many (in k) more messages or ciphertexts and receive encryptions for each message and decryptions for each ciphertexts *except* the challenge ciphertext c_b. The adversary responds to the challenge outputting $b' \in \{0,1\}$ to guess the value of b. If $b' = b$ the adversary wins the game, otherwise the adversary loses.

Definition 2.4.3. A symmetric-key encryption scheme is said to be secure under *Indistinguishability under Chosesn-Plaintext Attack* if for all PPT adversaries $\mathcal{A}$,

$$\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(k)$$

where the probability is taken over the random coins of the challenger to select sk and the random coins of the adversary to choose ciphertexts C, messages M and m_0, m_1. Thus, the adversary has a negligible advantage of guessing which message corresponds to the challenge encryption.

$\mathsf{IND}-\mathsf{CPA} \implies$ **Randomized Encryption**

Proposition 2.4.1. A symmetric encryption scheme secure under $\mathsf{IND}-\mathsf{CPA}$ must use a randomized encryption algorithm.

Proof. Suppose for purposes of contradiction, that a symmetric encryption scheme $\mathcal{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is secure under $\mathsf{IND}-\mathsf{CPA}$ but Enc is deterministic. We can construct an adversary $\mathcal{A}$ to do the following steps:

1. Generate a random message $m \in \{0,1\}^*$, query the encryption oracle for $c \leftarrow \mathsf{Enc}(m)$.
2. Generate $m' \neq m$ and set the challenge message pair to be (m, m'). The challenger returns c_b.
3. if $c_b = c$ then return 0, otherwise return 1

Since Enc is deterministic, then by definition, for any $sk \leftarrow \mathsf{Gen}(1^k)$, for any message $m \in \{0,1\}^*$, for any pair of encryptions $c_0 \leftarrow \mathsf{Enc}_{sk}(m), c_1 \leftarrow \mathsf{Enc}_{sk}(m)$ it will be that $c_0 = c_1$. Therefore, $\mathcal{A}$ wins the game with probability 1.

This contradicts the definition of security for $\mathcal{SE}$ under $\mathsf{IND}-\mathsf{CPA}$. Thus it must be that $\mathcal{SE}$ is either not $\mathsf{IND}-\mathsf{CPA}$ or Enc is randomized. □

This proposition is important as it shows that deterministic encryption cannot satisfy even the weaker definition of security, $\mathsf{IND}-\mathsf{CPA}$. Since our work aims to purposefully leak repeated encryptions in some conditions, it is important to understand that it will not provide $\mathsf{IND}-\mathsf{CPA}$ security.

2.5 Pseudorandom Functions and Permutations

We use Block Ciphers to construct both randomized and deterministic symmetric encryption schemes by executing a block cipher under different modes of operation. In order to analyze the security of block ciphers, we must first introduce pseudorandom functions and permutations which we can be used to model an ideal block cipher.

2.5.1 Pseudorandom Functions

A Pseudorandom function family (PRF) is a family of probabilistic polynomial-time (PPT) computable functions $F : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$ whose outputs are computationally indistinguishable from random [9, Chapter 3].

Definition 2.5.1. *Pseudorandom Function Family.* Let l, m be some polynomial functions of the security parameter 1^k. A family of functions $F_s : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^m$ is pseudorandom if and only if

- Easy-to-compute. $\forall s \in \{0,1\}^k, \forall x \in \{0,1\}^l$ $F_s(x)$ can be computed in polynomial-time.
- Computational Indistinguishability. For all PPT algorithms $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(k)$ such that

$$|\Pr[\mathcal{A}(1^k, F_s) = 1 \mid s \xleftarrow{\$} \{0,1\}^k] - \Pr[\mathcal{A}(1^k, F_\$) = 1 \mid F_\$ \xleftarrow{\$} \mathcal{RF}_m]| \leq \mathsf{negl}(k)$$

  where the probability is taken over choice of s and $F_\$$. In other words, $\mathcal{A}$ can distinguish F_s from a random function $F_\$$ with no more than negligible probability.

2.5.2 Pseudorandom Permutations

A Pseudorandom permutation family (PRP) is a family of PPT computable permutations $F : \mathcal{K} \times \mathcal{D} \to \mathcal{D}$ where a randomly selected permutation from this family maps $\mathcal{D}$ onto $\mathcal{D}$ in a manner that is computationally indistinguishable from a random permutation on $\mathcal{D}$ [9, Chapter 3].

Definition 2.5.2. *Pseudorandom Permutation Family.* Let l be some polynomial function of the security parameter 1^k. A family of permutations $F : \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ is pseudorandom if and only if

- Easy-to-compute. $\forall s \in \{0,1\}^k, \forall x \in \{0,1\}^l$ $F_s(x)$ can be computed in polynomial-time.
- Computational Indistinguishability. For all PPT algorithms $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(k)$ such that

$$|\Pr[\mathcal{A}(1^k, F_s) = 1 \mid s \xleftarrow{\$} \{0,1\}^k] - \Pr[\mathcal{A}(1^k, F_\$) = 1 \mid F_\$ \xleftarrow{\$} \mathcal{RP}_l]| \leq \mathsf{negl}(k)$$

  where the probability is taken over choice of s and $F_\$$. In other words, $\mathcal{A}$ can distinguish F_s from a random function $F_\$$ with no more than negligible probability.

2.5.3 Security Definitions for Indistinguishability

The computational indistinguishability requirement for pseudo-randomness ensures that no probabilistic polynomial-time algorithm $\mathcal{A}$, sometimes referred to as a computational distinguisher, can distinguish a pseudorandom function or permutation from that of random function or permutation. Next, we will formally define several adversarial models of security for the indistinguishability of PRFs and PRPs.

Indistinguishability Under Chosen-Plaintext Attack (IND-CPA)

In this security model, the computational distinguisher, denoted as the PPT adversary $\mathcal{A}$, has oracle access to a pseudorandom function in question F_s and a random function $F_\$$ with the same domain and range. Let $\mathcal{D}, \mathcal{R}$ be the domain and range of F_s, respectively. The model has two main phases.

Query Phase.

1. $\mathcal{A}$ is given access to two oracles $\mathcal{O}_0, \mathcal{O}_1$. One of these oracles corresponds to F_s and other $F_\$$. $\mathcal{A}$ does not know oracle maps to which function.
2. $\mathcal{A}$ can submit polynomially (in 1^k) many queries to $\mathcal{O}_b$, where $b \in \{0, 1\}$, where for each input query $x \in \mathcal{D}$, the adversary receives the response $\mathcal{O}_b(x)$.

Guess Phase.

1. $\mathcal{A}$ is then asked to decide which $\mathcal{O}_0, \mathcal{O}_1$ is the pseudorandom function F_s.
2. $\mathcal{A}$ outputs $g \in \{0, 1\}$ to respond that $\mathcal{O}_g$ is the pseudorandom function.
3. $\mathcal{A}$ wins if $\mathcal{O}_g = F_s$. Otherwise, $\mathcal{A}$ loses.

If $\mathcal{A}$ can win with a non-negligible probability, then $\mathcal{A}$ distinguishes F_s from a random function. If no such PPT algorithm $\mathcal{A}$ can be constructed such that $\mathcal{A}$ wins with non-negligible advantage for randomly sampled F_s from F, PRF family, then F is a pseudorandom function family secure for indistinguishability under chosen-plaintext attack.

Note that this adversarial model applies analogously when F_s is pseudorandom permutation and $F_\$$ is a random permutation over the domain of F_s. Hence, PRPs can similarly be provably secure in indistinguishability under chosen-plaintext attack model.

Indistinguishability Under Non-Adaptive/Adaptive Chosen-Ciphertext Attack (IND-CCA)

In the case of pseudorandom permutations, we can consider the case where an adversary is allowed to query an oracle that inverts the permutation. More formally, a PPT adversary $\mathcal{A}$ has oracle access to pseudorandom permutation in question F_s and its inverse F_s^{-1} as well as oracle access to $F_\$$, a random permutation on the domain of F_s, and the inverse of this random permutation $F_\$^{-1}$. Let $\mathcal{D}, \mathcal{R}$ be the domain and range of F_s, respectively. Note that $\mathcal{D} = \mathcal{R}$ for permutations. This model similarly has two main phases.

Query Phase.

1. $\mathcal{A}$ is given access to four oracles $\mathcal{O}_0, \mathcal{O}_0^{-1}, \mathcal{O}_1, \mathcal{O}_1^{-1}$. $\mathcal{A}$ does not know which pair of oracles, $\mathcal{O}_0, \mathcal{O}_0^{-1}$ or $\mathcal{O}_1, \mathcal{O}_1^{-1}$ maps to the pair of permutations F_s, F_s^{-1}.
2. $\mathcal{A}$ can submit polynomially (in 1^k) many queries to $\mathcal{O}_b$, where $b \in \{0,1\}$, where for each input query $x \in \mathcal{D}$, the adversary receives the response $\mathcal{O}_b(x)$.
3. $\mathcal{A}$ can submit polynomially (in 1^k) many queries to $\mathcal{O}_b^{-1}$, where $b \in \{0,1\}$, where for each input query $y \in \mathcal{R}$, the adversary receives the response $\mathcal{O}_b^{-1}(x)$.

Guess Phase.

1. $\mathcal{A}$ is then asked to decide which pair of oracles $\mathcal{O}_0, \mathcal{O}_0^{-1}$ or $\mathcal{O}_1, \mathcal{O}_1^{-1}$ corresponds to the pseudorandom function and its inverse F_s, F_s^{-1}.
2. Repeat querying:
   - **Non-Adaptive**: $\mathcal{A}$ is allowed polynomially more queries to the $\mathcal{O}_0, \mathcal{O}_1$
   - **Adaptive**: $\mathcal{A}$ is allowed polynomially more queries to all oracles $\mathcal{O}_0, \mathcal{O}_0^{-1}, \mathcal{O}_1, \mathcal{O}_1^{-1}$
3. $\mathcal{A}$ outputs $g \in \{0,1\}$ to respond that $\mathcal{O}_g, \mathcal{O}_g^{-1}$ is the pseudorandom function and its inverse.
4. $\mathcal{A}$ wins if $\mathcal{O}_g, \mathcal{O}_g^{-1} = F_s, F_s^{-1}$. Otherwise, $\mathcal{A}$ loses.

If $\mathcal{A}$ can win with a non-negligible probability, then $\mathcal{A}$ distinguishes F_s, F_s^{-1} from a random permutation and its inverse. If no such PPT algorithm $\mathcal{A}$ can be constructed such that $\mathcal{A}$ wins with non-negligible advantage for randomly sampled F_s and a randomly sampled permutation on $\mathcal{D}$, then F_s is secure in indistinguishability under chosen-ciphertext attack.

2.6 Block Ciphers

Definition 2.6.1. A *block cipher* is a function $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. k is the *key length* and n is the *block length*. A block cipher is a permutation. Therefore, there exits an inverse function $E^{-1} : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ such that

$$\forall s \in \{0,1\}^k, x \in \{0,1\}^n : E_s^{-1}(E_s(x)) = x$$

Block ciphers can only operate on inputs of a fixed size block length. In order to use block ciphers over arbitrarily long messages, we must introduce block cipher operating modes.

2.6.1 Block Cipher Modes

Block cipher operating modes enable the use of block ciphers to implement primitives like symmetric-key encryption schemes for arbitrarily sized inputs in order to provide confidentiality and/or integrity. Given a block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$, a block cipher mode is a set a procedures that take arbitrarily sized inputs and apply functions in n-bit blocks of the input using the block cipher and its inverse. The forward application of the block cipher is called encryption while the inverse application is called decryption [9, Chapter 4].

In this section we will describe a widely used cipher block mode and state its security based on the assumption that the block cipher is a pseudorandom permutation.

Cipher Block Chaining - CBC

Let $E_k : \{0,1\}^n \to \{0,1\}^n$ be a block cipher with a fixed key k and inverse E_k^{-1}. Below we define the encryption and decryption procedures for operating a block cipher in *Cipher Block Chaining* mode.

ENCRYPTION. The encryption function, denoted $\mathsf{CBC-ENCRYPT}$, takes the following input:

- $E_k : \{0,1\}^n \to \{0,1\}^n$, a block cipher with a fixed key k.
- $\mathsf{IV} \in \{0,1\}^n$, the initialization vector (IV).
- $M = \{M_1, \ldots, M_c\}$, the message blocks where each $M_i \in \{0,1\}^n$, and c is positive integer greater than 0.

The algorithm then produces ciphertext blocks $C = \{C_1, \ldots, C_c\}$ as follows:

$$C_i = \begin{cases} \mathsf{IV}, & \text{if } i = 0 \\ E_k(M_i \oplus C_{i-1}), & \text{otherwise} \end{cases}$$

DECRYPTION. The decryption function, denoted $\mathsf{CBC-DECRYPT}$, takes the following input:

- $E_k^{-1} : \{0,1\}^n \to \{0,1\}^n$, the inverse of a block cipher with a fixed key k.
- $\mathsf{IV} \in \{0,1\}^n$, the initialization vector (IV).
- $C = \{C_1, \ldots, C_c\}$, the ciphertext blocks where each $C_i \in \{0,1\}^n$, and c is positive integer greater than 0.

Let $C_0 = \mathsf{IV}$. The algorithm then produces plaintext message blocks $M = \{M_1, \ldots, M_c\}$ as follows:

$$M_i = E_k^{-1}(C_i) \oplus C_{i-1}$$

Noticeably, if one bit in a ciphertext block C_i is flipped, the decrypted block M_i is completely corrupted, M_{i+1} has the same bit flipped, but blocks M_{i+2} and on are uncorrupted. Hence, it is trivial to see that one could construct $\mathcal{A}$ to exploit this property to always win in the $\mathsf{IND-CCA2}$ security model. Thus, a pseuodrandom permutation block cipher operating in CBC mode is not $\mathsf{IND-CCA2}$ secure.

However, given a pseudorandom permutation block cipher, operated in CBC mode, we have the following theorem.

Theorem 2.6.1. Let $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be pseudorandom permutation block cipher. The symmetric-key encryption scheme $\mathcal{SE} - \mathsf{CBC} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined as

- $\mathsf{Gen}(1^k) : sk \xleftarrow{\$} \{0,1\}^k$
- $Enc_{sk}(m)$
  1. Generate $\mathsf{IV} \xleftarrow{\$} \{0,1\}^n$.
  2. Compute $C \leftarrow \mathsf{CBC-ENCRYPT}(E_{sk}, IV, m)$
  3. Output (IV, C)
- $Dec_{sk}(c)$
  1. Let $(\mathsf{IV}, C) \leftarrow c$
  2. Compute $M \leftarrow \mathsf{CBC-DECRYPT}(E_{sk}^{-1}, IV, C)$
  3. Output M

is secure under $\mathsf{IND-CPA}$.

The proof of 2.6.1 from [9, Chapter 4] shows that if there is an adversary $\mathcal{A}$ that wins in the $\mathsf{IND-CPA}$ security game for $\mathcal{SE} - \mathsf{CBC}$, then this adversary can be used to construct an adversary $\mathcal{A}'$ to win in the $\mathsf{IND-CPA}$ security game for the pseudorandom permutation E.

2.6.2 Padding

Typically, block cipher operating modes, like the one introduced in the previous section, require that the input length is a multiple of the block length. To support arbitrary sized inputs, we must *pad* the input.

Definition 2.6.2. A PKCS#7 *padding* of an input string $x \in \{0,1\}^*$, for block-length n, appends a 1 followed by 0^r to x, where

$$r = \begin{cases} n, & \text{if } \exists c \in \mathbb{N} \text{ s.t. } |x| = c \cdot n \\ n - 1 - (|x| \mod n), & \text{otherwise} \end{cases}$$

is returned.

A PKCS#7 *un-padding* of $y \in \{0,1\}^{q \cdot n}$, where $q \in \mathbb{N}, q > 0$, starts from least significant side (right) and removes "0"s until the first "1" appears, after which the "1" is removed and the remainder of the string is return.

From here on, when we refer to padding, the PKCS#7 scheme introduced above is implied.

2.7 Advanced Encryption standard (AES)

The Advanced Encryption Standard (AES), originally created by Daemen and Rijmen, is a block cipher that is widely used in practice today and standardized the National Institutes of Standards and Technology [7]. Specifically, based on today's best practices [4], we will use the 256 bit key variant of AES, denoted $\mathsf{AES}-256$.

Definition 2.7.1. AES is a block cipher with key length 256 and block length 128.

$$\mathsf{AES}-256 : \{0,1\}^{256} \times \{0,1\}^{128} \to \{0,1\}^{128}$$

2.7.1 Assumptions about AES in our Work

In this work, we use AES as our block cipher and CBC as our block mode. Thus, to make provable security claims about our cryptographic schemes, we must first assume that our block cipher is modeled after a pseudorandom permutation.

Assumption 2.7.1. AES is pseudorandom permutation family.

Therefore, based on our block cipher assumption, we can make a security claim about the symmetric-key encryption algorithm, denoted as $\mathsf{AES}-\mathsf{CBC}$.

Claim 2.7.1. By 2.7.1 and 2.6.1, $\mathsf{AES}-\mathsf{CBC}$ is secure under $\mathsf{IND}-\mathsf{CPA}$.

Using the AES block cipher and the CBC block operating mode as described in 2.6.1, it follows hat $\mathsf{AES}-\mathsf{CBC}$ is secure under chosen-plaintext attack.

Chapter 3

Privacy Definitions and Constructions for Computing Keyword Search and Bag-of-Words on Encrypted Free Text

In this chapter we formally define two schemes, one for computing keyword search and another for computing bag-of-words, both on encrypted free text. We then provide formal privacy definitions for each scheme. Next, we describe constructions of each scheme based on the cryptographic primitives defined in the previous chapter. Finally, we prove that these schemes achieve our security definitions.

3.1 Computing Functions on Encrypted Data

When we discuss computing functions on encrypted free text, we mean that we would like to compute a function of the underlying free text data without decrypting the text. Informally, for a ciphertext c that is the encryption of w, we output $f(w)$ for a pre-specified function f. Our constructions use hash functions and pseuodrandom permutations for computing keyword search and bag-of-words on encrypted free text.

Keyword Search. Given some encrypted free text and a keyword w, the keyword search function can be used finds all encrypted free text words whose underlying plaintext is equal to w. Specifically, given an auxiliary key sk_w for keyword w and an encrypted free text document $C = \{c_1, \ldots, c_n\}$, there exists a function D, such that for any $i \in \{1, \ldots, n\}, D(sk_w, c_i) = 1$ if c_i is an encryption of w and 0 otherwise. In the keyword setting, auxiliary keys sk_w can be derived for each keyword w. Even though other words in the encrypted free text document remain hidden, the presence of the keyword in the free

text document is revealed just as it would if the data was unencrypted free text.

Bag-of-Words. Given some encrypted free text, the bag-of-words computation can be used to find the locations and count the frequency of n-grams of the underlying free text data. Specifically, given an auxiliary key sk_f and an encrypted free text document $C = \{c_1, \ldots, c_n\}$, there exists a function D, such that for any $i, j \in \{1, \ldots, n\}, D(sk_f, c_i, c_j) = 1$ if c_i and c_j are encryptions of the same underlying plaintext and 0 otherwise. D can be used on every pair of ciphertexts in C to count the number of distinct repetitions of underlying plaintexts and locate their index in the encrypted free text. This information can be used to determine if two free text documents are similar based on which underlying plaintext words repeat across the free text documents. Therefore, the main function of a bag-of-words computation, as described in 1.1, still applies just as it would on unencrypted free text data.

3.2 Privacy Definition Overview

Our goal for privacy is to limit the information leaked by our schemes to precisely the results of computing keyword search and bag-of-words computations on the encrypted free text data and nothing else. That is, the auxiliary keys, sent during the auxiliary key request phase, should only leak the functional decryption of the encrypted free text as specified above and nothing else. For example, an auxiliary key sk_w for a keyword w should not reveal any information about encryptions of a word $w' \neq w$. Similarly, a bag-of-words auxiliary key should only reveal the locations of repeated underlying plaintexts but should not reveal any information about what the underlying plaintext is. The fundamental data privacy requirement in our algorithms is to ensure that knowledge of an auxiliary key precisely limits the leaked information to that of the desired output of the function.

3.3 Keyword Search

The keyword search scheme, denoted PKS for "Private-key Keyword Search", enables a learner to check if a ciphertext is an encryption of some keyword. The leaner sends a request to the owner with a specific keyword, and the owner decides to either approve the request by deriving an auxiliary key for the specified keyword or reject the request and do nothing. The learner can ask the owner for polynomially many keyword keys during any time in the life cycle of the interaction between parties.

3.3.1 Cryptographic Scheme

Definition 3.3.1. The *Private-key Keyword Search* (PKS) cryptosystem is a tuple of four algorithms $(Setup, Extract, Hide, Check)$, where

- $\mathsf{PKS.Setup}(1^k)$ uses the security parameter 1^k to randomly generate a master secret key msk.
- $\mathsf{PKS.Extract}_{msk}(w)$ uses the master secret key, msk, and outputs a derived secret key sk_w for keyword w.
- $\mathsf{PKS.Hide}_{msk}(w)$ uses the master secret key, msk, to output a ciphertext c_w for input word w.
- $\mathsf{PKS.Check}_{sk_w}(c_{w'})$ uses the secret key, sk_w for keyword w, on ciphertext $c_{w'}$ and outputs 1 if $w = w'$, otherwise outputs 0.

and the scheme is *correct* if $\forall(k > 0, msk \leftarrow \mathsf{Setup}(1^k), w \in \{0,1\}^*, sk_w \leftarrow \mathsf{Extract}_{msk}(w))$:

$$\mathsf{Check}_{sk_w}((\mathsf{Hide}_{msk}(w)) = 1$$

3.3.2 Indistinguishability under Adaptive Chosen-Keyword Attack (IND-CKA)

To capture the definition of privacy for this scheme, we present an adversarial model which we denote Indistinguishability under Adaptive Chosen-Keyword Attack ($\mathsf{IND-CKA}$). Like the other indistinguishability definitions in chapter 2, our model has two parties, the *challenger* and a PPT *adversary* $\mathcal{A}$ which is given oracle access to $\mathsf{PKS.Hide}$ and $\mathsf{PKS.Extract}$.

Remark 3.3.1. We emphasize that in this privacy definition we give the adversary the ability to get encryptions (oracle access to $\mathsf{PKS.Hide}_{msk}$). An alternative definition might have the challenger instead decide on free text to encrypt and give to the adversary before the query phase. We chose to give the adversary the ability to ask for encryptions because we did not want to limit our definitions of privacy to a system where data is not dynamic.

Definition 3.3.2. The Private-key Keyword Search security $\mathsf{Game}_{\mathsf{PKS}}$ is defined as follows.

Setup Phase.

1. Challenger generates $msk \leftarrow \mathsf{PKS.Setup}(1^k)$.
2. Challenger chooses a set $\mathcal{KW} \subset \{0,1\}^*$ of allowed keywords, where $|\mathcal{KW}|$ is $\mathsf{poly}(k)$.

Query Phase.

1. $\mathcal{A}$ can submit $\mathsf{poly}(k)$ queries: $c \leftarrow \mathcal{O}_H(w)$ where $w \in \{0,1\}^*$ and $\mathcal{O}_H$ is an oracle for the function $\mathsf{PKS.Hide}_{msk}$

2. $\mathcal{A}$ can submit $\mathsf{poly}(k)$ queries: $sk_w \leftarrow \mathcal{O}_E(w)$ where $w \in \mathcal{KW}$ and $\mathcal{O}_E$ is an oracle for the function $\mathsf{PKS.Extract}_{msk}$. Let $\mathcal{KW}_{sk}$ Denote the set of keywords queried in this phase.

Challenge Phase.

1. $\mathcal{A}$ selects two words $w_0, w_1 \in \{0,1\}^*$ where $w_0, w_1 \notin \mathcal{KW}_{sk}$.
2. Challenger secretly computes $b \xleftarrow{\$} \{0,1\}$, $c_b \leftarrow \mathsf{PKS.Hide}_{msk}(w_b)$ and outputs c_b to $\mathcal{A}$.
3. $\mathcal{A}$ can submit $\mathsf{poly}(k)$ queries: $c \leftarrow \mathcal{O}_H(w)$ where $w \in \{0,1\}^*$.
4. $\mathcal{A}$ can submit $\mathsf{poly}(k)$ queries: $sk_w \leftarrow \mathcal{O}_E(w)$ where $w \in \mathcal{KW}, w \notin \{w_0, w_1\}$.
5. $\mathcal{A}$ outputs $b' \in \{0,1\}$.

where the adversary wins $\mathsf{Game}_{\mathsf{PKS}}$ if $b' = b$.

a PKS construction is said to be *secure under Indistinguishability under Adaptive Chosen-Keyword Attack*, for any set of approved keywords $\mathcal{KW} \subset \{0,1\}^*$ where $|\mathcal{KW}| = \mathsf{poly}(k)$, and for all PPT adversaries $\mathcal{A}$ that play $\mathsf{Game}_{\mathsf{PKS}}$,

$$\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(k)$$

where the probability is taken over the random coins of challenger in selecting msk and the random coins of the adversary in selecting queries and the challenge words. That is, the adversary has a negligible advantage in winning $\mathsf{Game}_{\mathsf{PKS}}$.

Remark 3.3.2. We argue that the above security definition captures the privacy of our data owner, data learner model because it upholds the privacy requirement that a data learner (the adversary) should not be able to learn anything about a ciphertext that hides a keyword for which the learner does not possess a secret key. Note that this security definition is very similar to $\mathsf{IND-CPA}$ model defined in 2.4.1. In fact, if the challenger were to choose an empty allowed keyword set ($|\mathcal{KW}_{sk}| = 0$) then $\mathsf{IND-CKA}$ collapses directly into $\mathsf{IND-CPA}$. $\mathcal{KW}_{sk}$ is the crucial ingredient in the definition which restricts the adversary from obtaining a secret key that can check if the challenge ciphertext hides one the challenge words.

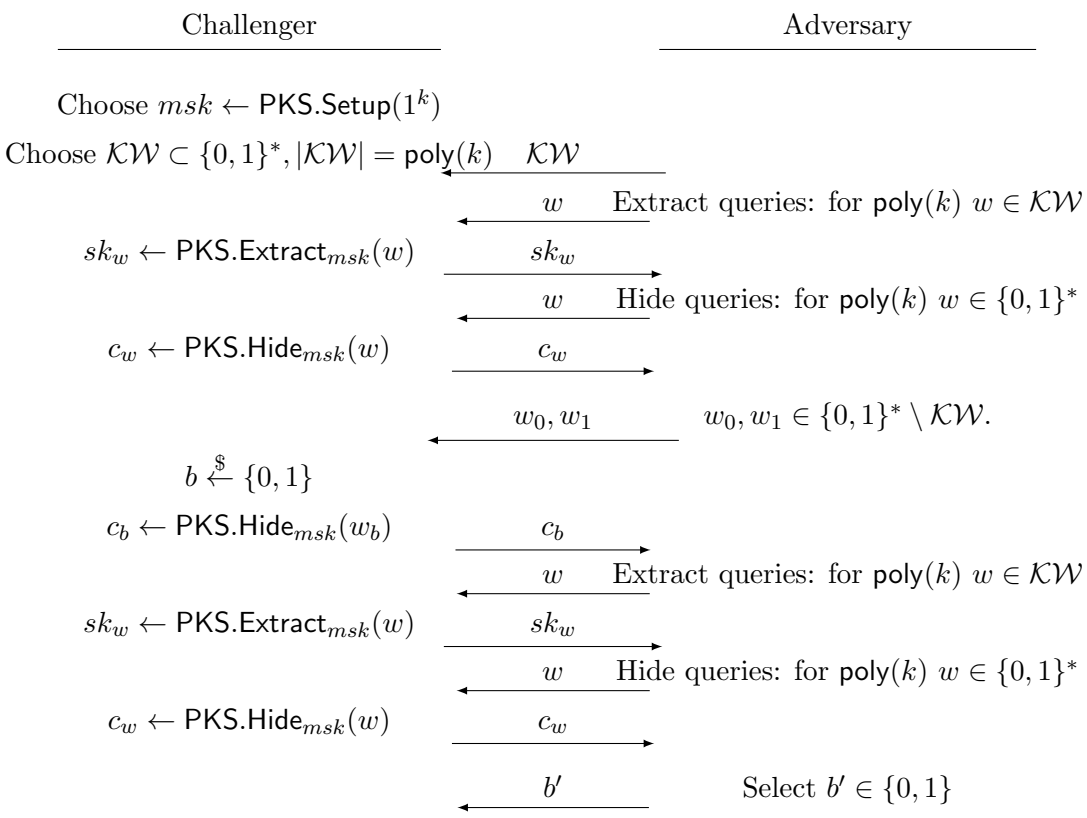

Figure 3-1: Diagram of $\mathsf{Game}_{\mathsf{PKS}}$

3.3.3 Construction of an IND – CKA PKS Scheme

Let $\mathcal{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Let $\mathcal{H}$ be a cryptographic hash function as defined in 2.2.1, with output length k. Below is a construction of a PKS scheme.

- $\mathsf{PKS.Setup}(1^k)$:
  1. $msk \xleftarrow{\$} \{0,1\}^k$
- $\mathsf{PKS.Extract}_{msk}(w)$:
  1. $sk_w \leftarrow \mathcal{H}(w||msk)$
  2. Output sk_w
- $\mathsf{PKS.Hide}_{msk}(w)$:
  1. $sk_w \leftarrow \mathsf{PKS.Extract}_{msk}(w)$
  2. $c \leftarrow \mathsf{Enc}_{sk_w}(1^n)$
  3. Output c
- $\mathsf{PKS.Check}_{sk_w}(c)$:

1. $m \leftarrow \mathsf{Dec}_{sk_w}(c)$
2. If $m = 1^n \rightarrow$ Output 1
3. Otherwise $\rightarrow$ Output 0

3.3.4 Correctness of PKS

It is easy to see that the construction above is correct. For any security parameter 1^k for $k > 0$, let $msk \in \{0,1\}^k$, let $w \in \{0,1\}^*$ be any word, and let $sk_w \in \{0,1\}^k$. By the correctness definition of the symmetric-key encryption scheme $\mathcal{SE}$,

$$\mathsf{Check}_{sk_w}((\mathsf{Hide}_{msk}(w)) = \mathsf{Dec}_{sk_w}(\mathsf{Enc}_{sk_w}(1^n)) = 1^n$$

Hence, the equality condition in step 2 of the PKS.Check function will hold, and the function will output 1 as desired.

3.3.5 Security Proof of PKS

Theorem 3.3.1. Let $\mathcal{PKS} = (\mathsf{PKS.Setup}, \mathsf{PKS.Extract}, \mathsf{PKS.Hide}, \mathsf{PKS.Check})$, where its hash function is replaced with a random oracle as defined in 2.3.1 and its symmetric encryption scheme uses a pseudorandom permutation family $E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ as defined in 2.6.1.

Then $\mathcal{PKS}$ is secure under Indistinguishability under Adaptive Chosen-Keyword Attack ($\mathsf{IND-CKA}$).

Proof. To show that $\mathcal{PKS}$ is secure under $\mathsf{IND-CKA}$, we must show that there does not exist a PPT algorithm $\mathcal{A}$ that wins $\mathsf{Game}_{\mathsf{PKS}}$ with more than negligible advantage. To prove this, we will use a hybrid argument to show that over the random coin tosses of $\mathcal{A}$ and the challenger, the outputs of oracle queries to PKS.Extract and PKS.Hide and the challenge ciphertext c_b are computationally indistinguishable from random.

Let $\mathsf{poly}_1(k), \mathsf{poly}_2(k)$ be some polynomials in k. We can represent the distribution of a $\mathsf{Game}_{\mathsf{PKS}}$ transcript between an adversary and a challenger with the following tuple:

$$\begin{aligned} D_0 = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &sk_{kw_0}, \ldots, sk_{kw_{\mathsf{poly}_1(k)}}, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &c_{w_0}, c_{w_1}, \ldots, c_{w_{\mathsf{poly}_2(k)}}, \\ &c_b, \\ &b') \end{aligned}$$

where each kw_i is a unique keyword generated by the challenger in $\mathcal{KW}$, each sk_{kw_i} is the output of invoking the $\mathsf{PKS.Extract}$ oracle on kw_i, each w_i is a unique plaintext word generated by $\mathcal{A}$, each c_{w_i} is the output of invoking the $\mathsf{PKS.Hide}$ oracle on w_i, the challenge ciphertext $c_b = \mathsf{PKS.Hide}(w_b)$, where $b \in \{0,1\}$ is chosen randomly by the challenger, and the adversary's output is $b' \in \{0,1\}$.

From the definitions of $\mathsf{PKS.Extract}$ and $\mathsf{PKS.Hide}$, where $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is assumed to be a pseudorandom permutation family, we can re-write D_0 as follows

$$
\begin{aligned}
D_0 = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\
&\mathcal{H}(kw_0||msk), \ldots, \mathcal{H}(kw_{\mathsf{poly}_1(k)}||msk), \\
&w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\
&\mathsf{Enc}_{\mathcal{H}(w_0||msk)}(1^n), \mathsf{Enc}_{\mathcal{H}(w_1||msk)}(1^n), \ldots, \mathsf{Enc}_{\mathcal{H}(c_{\mathsf{poly}_2(k)}||msk)}(1^n), \\
&\mathsf{Enc}_{\mathcal{H}(w_b||msk)}(1^n), \\
&b') \\
= (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\
&\mathcal{H}(kw_0||msk), \ldots, \mathcal{H}(kw_{\mathsf{poly}_1(k)}||msk), \\
&w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\
&(iv_0, E_{\mathcal{H}(w_0||msk)}(1^n \oplus iv_0)), (iv_1, E_{\mathcal{H}(w_1||msk)}(1^n \oplus iv_1)), \\
&\ldots, (iv_{\mathsf{poly}_2(k)}, E_{\mathcal{H}(w_{\mathsf{poly}_2(k)}||msk)}(1^n \oplus iv_{\mathsf{poly}_2(k)})) \\
&(iv_f, E_{\mathcal{H}(w_b||msk)}(1^n \oplus iv_f)), \\
&b')
\end{aligned}
$$

where iv_f and each iv_i is a randomly sampled initialization vector as defined in 2.6.1.

Step 1 - PKS.Extract Outputs are Pseudorandom

Next, let r_{kw_0} be uniformly chosen. Define the hybrid distribution[1]

$$\begin{aligned} D_1 = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &\mathbf{r_{kw_0}}, \ldots, \mathcal{H}(kw_{\mathsf{poly}_1(k)}||msk), \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, E_{\mathcal{H}(w_0||msk)}(1^n \oplus iv_0), (iv_1, E_{\mathcal{H}(w_1||msk)}(1^n \oplus iv_1)), \\ &\ldots, (iv_{\mathsf{poly}_2(k)}, E_{\mathcal{H}(w_{\mathsf{poly}_2(k)}||msk)}(1^n \oplus iv_{\mathsf{poly}_2(k)})), \\ &(iv_f, E_{\mathcal{H}(w_b||msk)}(1^n \oplus iv_0)), \\ &b') \end{aligned}$$

where D_1 is the same as D_0 except $\mathcal{H}(kw_0||msk)$ is replaced by r_{kw_0}.

By our assumption that $\mathcal{H}$ is a random oracle, $\mathcal{H}$ maps every unique input to a uniformly at random output of length k. Therefore, since msk is random, $\mathcal{H}(kw_0||msk)$ is computationally indistinguishable from r_{kw_0}. Thus, D_0 is computationally indistinguishable from D_1, denoted $D_0 \approx D_1$.

Next, we define distributions $D_2, \ldots, D_{\mathsf{poly}_1(k)}$, where each consecutive D_i replaces the next $\mathcal{H}(kw_i||msk)$ with random r_{kw_i}

$$\begin{aligned} D_i = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{i-1}}, \mathbf{r_{kw_i}}, \ldots, \mathcal{H}(kw_{\mathsf{poly}_1(k)}||msk), \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, E_{\mathcal{H}(w_0||msk)}(1^n \oplus iv_0), (iv_1, E_{\mathcal{H}(w_1||msk)}(1^n \oplus iv_1)), \\ &\ldots, (iv_{\mathsf{poly}_2(k)}, E_{\mathcal{H}(w_{\mathsf{poly}_2(k)}||msk)}(1^n \oplus iv_{\mathsf{poly}_2(k)})), \\ &(iv_f, E_{\mathcal{H}(w_b||msk)}(1^n \oplus iv_f)), \\ &b') \end{aligned}$$

By the same random oracle argument, for each consecutive pair of hybrid distributions, $D_{i-1} \approx D_i$. It follows that $D_0 \approx D_{\mathsf{poly}_1(k)}$.

[1]This represents a modified $\mathsf{Game}_{\mathsf{PKS}}$ between the challenger and the adversary, where the challenger now responds with uniform random values for the first Extract.

Step 2 - PKS.Hide Keys are Pseudorandom

Next, let r_{w_0} be uniformly chosen. Define the hybrid distribution[2]

$$\begin{aligned} D_{\mathsf{poly}_1(k)+1} = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{\mathsf{poly}_1(k)}} \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, E_{\mathbf{r_{w_0}}}(1^n \oplus iv_0), (iv_1, E_{\mathcal{H}(w_1||msk)}(1^n \oplus iv_1)), \\ &\ldots, (iv_{\mathsf{poly}_2(k)}, E_{\mathcal{H}(w_{\mathsf{poly}_2(k)}||msk)}(1^n \oplus iv_{\mathsf{poly}_2(k)})), \\ &(iv_f, E_{\mathcal{H}(w_b||msk)}(1^n \oplus iv_f)), \\ &b') \end{aligned}$$

Again, by our assumption that $\mathcal{H}$ is a random oracle and since msk is random, $\mathcal{H}(w_0||msk)$ is computationally indistinguishable from r_{w_0}. Therefore, $D_{\mathsf{poly}_1(k)} \approx D_{\mathsf{poly}_1(k)+1}$. Similarly, we define distributions $D_{\mathsf{poly}_1(k)+2}, \ldots, D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)}$, where each consecutive $D_{\mathsf{poly}(k)+i}$ replaces the next $\mathcal{H}(w_i||msk)$ with random r_{w_i}

$$\begin{aligned} D_{\mathsf{poly}_1(k)+i} = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{\mathsf{poly}_1(k)}} \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, E_{r_{w_0}}(1^n \oplus iv_0), (iv_1, E_{r_{w_1}}(1^n \oplus iv_1)), \\ &\ldots, (iv_i, E_{\mathbf{r_{w_i}}}(1^n \oplus iv_i)), \ldots \\ &(iv_{\mathsf{poly}_2(k)}, E_{\mathcal{H}(w_{\mathsf{poly}_2(k)}||msk)}(1^n \oplus iv_{\mathsf{poly}_2(k)})), \\ &(iv_f, E_{r_{w_b}}(1^n \oplus iv_f)), \\ &b') \end{aligned}$$

By the same random oracle argument as before, for each consecutive pair of hybrid distributions $D_{\mathsf{poly}_1(k)+i-1} \approx D_{\mathsf{poly}_1(k)+i}$. It follows that, $D_0 \approx D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)}$.

[2]This represents a modified $\mathsf{Game}_{\mathsf{PKS}}$ between the challenger and the adversary, where the challenger now uses a uniformly random key in the first response to a PKS.Hide queries.

Step 3 - PKS.Hide Query Ciphertexts are Pseudorandom

Next, let u_0 be uniformly chosen. Define the hybrid distribution[3]

$$\begin{aligned} D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1} = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{\mathsf{poly}_1(k)}} \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, \mathbf{u_0}), (iv_1, E_{r_{w_1}}(1^n \oplus iv_\cdot)), \\ &\ldots, (iv_{\mathsf{poly}_2(k)+1}, E_{r_{w_{\mathsf{poly}_2(k)}}}(1^n \oplus iv_{\mathsf{poly}_2(k)+1})) \\ &(iv_f, E_{r_{w_b}}(1^n \oplus iv_f)), \\ &b') \end{aligned}$$

Suppose there exists a PPT algorithm that computationally distinguishes between

$$D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)} \text{ and } D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1}$$

It follows, there exists a PPT algorithm **Dist** that takes a transcript from $\mathsf{Game}_{\mathsf{PKS}}$ as input and outputs 1 or 0. Without loss of generality, we interpret 1 as a guess that the transcript is from the distribution $D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)}$ and 0 as a guess that the transcript is from the distribution $D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1}$. It follows that **Dist** is correct with probability greater than $\frac{1}{2} + \mathsf{negl}(k)$ where the probability is taken over the coin tosses of $\mathcal{A}$ and the challenger.

Then, we can use **Dist** to construct a PPT algorithm PRPDistinguisher to computationally distinguish pseudorandom permutation families from random. Let $\mathsf{PRPDistinguisher}(\mathcal{O}^g)$ be a PPT algorithm that distinguishes a pseudorandom permutation g from a truly random permutation with oracle access to g using $\mathcal{O}^g$.

$\mathsf{PRPDistinguisher}(\mathcal{O}^g)$:

1. Generate the transcript by running $\mathcal{A}$ and the challenger, using the hybrid distribution $t \in D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1}$ to represent the challenger's modified responses, except now

[3]This represents a modified $\mathsf{Game}_{\mathsf{PKS}}$ between the challenger and the adversary, where the challenger now responds to the first PKS.Hide query with a uniformly random value.

replace the first invocation of the PRP E with a call to the oracle $\mathcal{O}^g(1^n \oplus iv_0)$:

$$\begin{aligned} t = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{\mathsf{poly}_1(k)}} \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, \mathcal{O}^{\mathbf{g}}(1^n \oplus iv_0)), (iv_1, E_{r_{w_1}}(1^n \oplus iv_1)), \\ &\ldots, (iv_{\mathsf{poly}_2(k)+1}, E_{r_{w_{\mathsf{poly}_2(k)}}}(1^n \oplus iv_{\mathsf{poly}_2(k)+1})) \\ &(iv_f, E_{r_{w_b}}(1^n \oplus iv_f)), \\ &b') \end{aligned}$$

where each $r_{kw_i}, r_{w_i}, iv_f, iv_i \xleftarrow{\$} \{0,1\}^k$

2. Output $\mathbf{Dist}(t)$

By replacing E with $\mathcal{O}^g$ in the definition of **Dist**, it follows that

$$\begin{aligned} \Pr[\mathsf{PRPDistinguisher}(\mathcal{O}^g) = 1 \mid g \text{ is pseudorandom }] = & \\ &= \Pr[\mathbf{Dist}(t) = 1 \mid t \leftarrow D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)}] \\ &> \frac{1}{2} + \mathsf{negl}(k) \end{aligned}$$

where the probability is taken over coin tosses of PRPDistinguisher and **Dist**. Therefore, PRPDistinguisher is a computational distinguisher for pseudorandom permutations. This contradicts our pseudorandomness assumption for E.

Therefore, no such PPT algorithm **Dist** exists. It follows that

$$D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)} \approx D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1}$$

from which it follows that $D_0 \approx D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1}$.

Next, let each hybrid distribution

$$D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+i} \in \{D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+1}, \ldots, D_{\mathsf{poly}_1(k)+2\cdot\mathsf{poly}_2(k)}\}$$

be defined as

$$\begin{aligned} D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+i} = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{\mathsf{poly}_1(k)}} \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, u_0), (iv_1, u_1), \ldots, (\mathbf{iv_i}, \mathbf{u_i}), \\ &\ldots, (iv_{\mathsf{poly}_2(k)}, E_{r_{w_{\mathsf{poly}_2(k)}}}(1^n \oplus iv_{\mathsf{poly}_2(k)})), \\ &(iv_f, E_{r_{w_b}}(1^n \oplus iv_f)), \\ &b') \end{aligned}$$

where by pseudorandomness of E, we use the same argument as above to show that $D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+i-1} \approx D_{\mathsf{poly}_1(k)+\mathsf{poly}_2(k)+i}$. It follows that $D_0 \approx D_{\mathsf{poly}_1(k)+2\cdot\mathsf{poly}_2(k)}$.

Step 4 - Challenge Ciphertext is Pseudorandom

Let u_f be uniformly chosen and define the final hybrid distribution[4]

$$\begin{aligned} D_F = (&kw_0, \ldots, kw_{\mathsf{poly}_1(k)}, \\ &r_{kw_0}, \ldots, r_{kw_{\mathsf{poly}_1(k)}} \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_0, u_0), (iv_1, u_1), \ldots, (iv_{\mathsf{poly}_2(k)}, u_{\mathsf{poly}_2(k)}), \\ &(iv_f, \mathbf{u_f}), \\ &b') \end{aligned}$$

Again, by the pseudorandomness of E, we use the same argument as above to show that $D_{\mathsf{poly}_1(k)+2\cdot\mathsf{poly}_2(k)} \approx D_F$. Then, it follows that $D_0 \approx D_F$. Clearly, D_F is a random distribution over the coin tosses of the challenger and the adversary. Therefore D_0 is computationally indistinguishable from random.

Hence, any transcript of $\mathsf{Game}_{\mathsf{PKS}}$ is computationally indistinguishable from random. Specifically, the challenge ciphertext c_b and the outputs of all oracle queries are computationally indistinguishable from random. Therefore, any PPT adversary wins $\mathsf{Game}_{\mathsf{PKS}}$ with negligible advantage. □

[4]This represents a modified $\mathsf{Game}_{\mathsf{PKS}}$ between the challenger and the adversary, where the challenger now sends a challenge ciphertext that is sampled uniformly at random.

3.4 Bag-of-Words (Frequency Count)

The bag-of-words or frequency count search scheme, denoted PFS for "Private-key Frequency Search", enables a learner to find all locations where a ciphertext is the encryption of repeated plaintext in a set of encrypted free text records. The learner asks the owner for a frequency count auxiliary key only once. After this key is released to the learner, the learner is able to compute frequency counts for all underlying plaintexts.

3.4.1 Cryptographic Scheme

Definition 3.4.1. The *Private-key Frequency Search* (PFS) cryptosystem is a tuple of three algorithms $(Setup, Disguise, Recognize)$, where

- $\mathsf{PFS.Setup}(1^k)$ uses the security parameter 1^k and randomly generates a master secret key msk, and a frequency secret key sk_f.
- $\mathsf{PFS.Disguise}_{msk,sk_f}(w)$ uses the master secret key, msk, and frequency key, sk_f, to output c_w, a ciphertext for a word w.
- $\mathsf{PFS.Recognize}_{sk_f}(c_w, c_{w'})$ uses the frequency secret key, sk_f, and outputs 1 if $w = w'$ and 0 otherwise.

and the scheme is *correct* if

$$\forall(k > 0, msk, sk_f \leftarrow \mathsf{Setup}(1^k), w, w' \in \{0,1\}^*, d_0 \leftarrow \mathsf{Disguise}_{msk}(w), d_1 \leftarrow \mathsf{Disguise}_{msk}(w'))$$

if $w = w'$ then:

$$\mathsf{Recognize}_{sk_f}(d_0, d_1) = 1$$

and if $w \neq w'$ then:

$$\mathsf{Recognize}_{sk_f}(d_0, d_1) = 0$$

3.4.2 Indistinguishability under Restricted-Plaintext Attack (IND-RPA)

To capture the definition of privacy for this scheme, we present an adversarial model which we denote Indistinguishability under Restricted-Plaintext Attack ($\mathsf{IND-RPA}$). Our model has two parties, the *challenger* and a PPT *adversary* $\mathcal{A}$ which is given the frequency key sk and oracle access to PFS.Disguise. Unlike other indistinguishability definitions, this model requires the adversary to select challenge plaintexts that have not been sent as queries to any oracles and after the challenge ciphertext is set, the adversary can continue querying the oracle except with the chosen challenge plaintexts. Therefore, in this model the adversary only has restricted access to the encryption oracle.

Remark 3.4.1. Once again, in this privacy definition we give the adversary the ability to get encryptions (oracle access to $\mathsf{PFS.Disguise}_{msk}$). Alternatively the challenger could have

instead generated some free text to encrypt and send to the adversary before the query phase. We chose to give the adversary the ability to ask for encryptions because we do not want to limit our definitions of privacy to a system where data is not dynamic.

Definition 3.4.2. In the Private-key Frequency Search security $\mathsf{Game}_{\mathsf{PFS}}$

Setup Phase.

1. Challenger generates $msk, sk_f \leftarrow \mathsf{PFS.Setup}(1^k)$.
2. Challenger sends sk_f to $\mathcal{A}$.

Query Phase.

1. $\mathcal{A}$ can submit $\mathsf{poly}(k)$ queries: $c \leftarrow \mathcal{O}_D(w)$ where $w \in \{0,1\}^*$ and $\mathcal{O}_D$ is an oracle for the function $\mathsf{PFS.Disguise}_{msk,sk_f}$. Let $\mathcal{W}$ be the set of words queried in this phase.

Challenge Phase.

1. $\mathcal{A}$ selects two words $w_0, w_1 \in \{0,1\}^*$ where $w_0, w_1 \notin \mathcal{W}$.
2. Challenger secretly computes $b \xleftarrow{\$} \{0,1\}, c_b \leftarrow \mathsf{PFS.Disguise}_{msk,sk_f}(w_b)$ and outputs c_b to $\mathcal{A}$.
3. $\mathcal{A}$ can submit $\mathsf{poly}(k)$ queries: $c \leftarrow \mathcal{O}_D(w)$ where $w \in \{0,1\}^* \setminus \{w_0, w_1\}$.
4. $\mathcal{A}$ outputs $b' \in \{0,1\}$.

where $\mathcal{A}$ wins $\mathsf{Game}_{\mathsf{PFS}}$ is $b' = b$.

A PFS construction is said to be *secure under Indistinguishability under Restricted-Plaintext Attack* if for all PPT adversaries $\mathcal{A}$,

$$\Pr[b' = b] \leq \frac{1}{2} + \mathsf{negl}(k)$$

where the probability is taken over the random of coins of the challenger for selecting msk and sk_f, and the random coins of the adversary for selecting queries and the challenge words. Thus, the adversary has a negligible advantage in guessing which word is disguised in the challenge ciphertext.

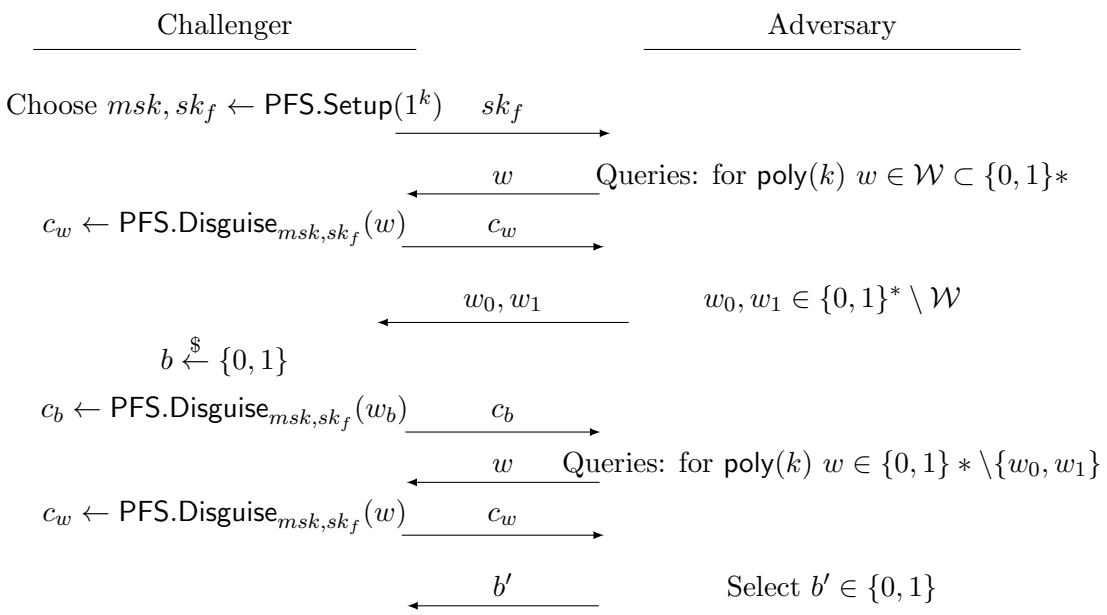

Figure 3-2: Diagram of $\mathsf{Game}_{\mathsf{PFS}}$

Remark 3.4.2. The restricted nature of the game captures the privacy requirements of our data owner, data learner model because it upholds that a data learner (the adversary) should not be able to learn anything more than if two ciphertexts are disguises of the same underlying plaintext. The restrictions ensure that the adversary cannot use previously known information about the ciphertext of the challenge plaintexts. Thus, if the adversary can distinguish between ciphertexts of previously un-queried plaintext, then the adversary wins.

3.4.3 Construction of an IND − RPA PFS Scheme

Let the block cipher $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a pseudorandom permutation used in a symmetric-key encryption scheme $\mathcal{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as defined in 2.6.1. Let $\mathcal{H}$ be a cryptographic hash function as defined in 2.2.1, with output length k. Below is a construction of a PFS scheme.

- $\mathsf{PFS.Setup}(1^k)$:
  1. $sk_f \xleftarrow{\$} \{0,1\}^k$
  2. $msk \xleftarrow{\$} \{0,1\}^k$
  3. Output msk, sk_f
- $\mathsf{PFS.Disguise}_{msk,sk_f}(w)$:
  1. $c \leftarrow \mathsf{Enc}_{sk_f}(\mathcal{H}(w||msk))$

2. Output c

- $\mathsf{PFS.Recognize}_{sk_f}(c, c')$:

  1. $y \leftarrow \mathsf{Dec}_{sk_f}(c)$
  2. $y' \leftarrow \mathsf{Dec}_{sk_f}(c')$
  3. Output 1 if $y = y'$, otherwise output 0

3.4.4 Correctness of PFS

It is easy to see that the construction above is correct. For any security parameter 1^k for $k > 0$, let $msk, sk_f \in \{0,1\}^k$. By the correctness definition of the symmetric-key encryption scheme $\mathcal{SE}$, for all $w, w' \in \{0,1\}^*$:

if $w = w'$ then:

$$\mathsf{Recognize}_{sk_f}(\mathsf{Disguise}_{msk,sk_f}(w), \mathsf{Disguise}_{msk,sk_f}(w')) = 1$$

since

$$\begin{aligned} Dec_{sk_f}(\mathsf{Enc}_{sk_f}(\mathcal{H}(w||msk)) &= \mathcal{H}(w||msk) \\ &= \mathcal{H}(w'||msk) \\ &= Dec_{sk_f}(\mathsf{Enc}_{sk_f}(\mathcal{H}(w'||msk)) \end{aligned} \tag{3.1}$$

and if $w \neq w'$ then:

$$\mathsf{Recognize}_{sk_f}(\mathsf{Disguise}_{msk,sk_f}(w), \mathsf{Disguise}_{msk,sk_f}(w')) = 0$$

since

$$\begin{aligned} Dec_{sk_f}(\mathsf{Enc}_{sk_f}(\mathcal{H}(w||msk)) &= \mathcal{H}(w||msk) \\ &\neq \mathcal{H}(w'||msk) \\ &= Dec_{sk_f}(\mathsf{Enc}_{sk_f}(\mathcal{H}(w'||msk))) \end{aligned} \tag{3.2}$$

3.4.5 Security Proof for PFS

Theorem 3.4.1. Let $\mathcal{PFS} = (\mathsf{PFS.Setup}, \mathsf{PFS.Disguise}, \mathsf{PFS.Recognize})$, where its hash function is replaced with a random oracle as defined in 2.3.1 and its symmetric encryption scheme uses a pseudorandom permutation family $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ as defined in 2.6.1.

Then $\mathcal{PFS}$ is secure under Indistinguishability under Restricted Plaintext Attack ($\mathsf{IND}-\mathsf{RPA}$).

Proof. To show that $\mathcal{PFS}$ is secure under $\mathsf{IND}-\mathsf{RPA}$, we must show that there does not exist a PPT algorithm $\mathcal{A}$ that wins $\mathsf{Game}_{\mathsf{PFS}}$ with more than negligible advantage. To prove this, we will use a hybrid argument to show that over the random coin tosses of $\mathcal{A}$ and the challenger, the outputs of oracle queries to $\mathsf{PFS.Disguise}$ and challenge ciphertext c_b are computationally indistinguishable from random.

Let $\mathsf{poly}(k)$ be some polynomial in k. We can represent the distribution of a $\mathsf{Game}_{\mathsf{PFS}}$ transcript between an adversary and a challenger with the following tuple:

$$\begin{aligned} D_0 = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &c_{w_2}, \ldots, c_{w_{\mathsf{poly}(k)}}, \\ &c_b, \\ &b') \end{aligned}$$

where sk_f is randomly generated by the challenger, each w_i is a unique plaintext word generated by $\mathcal{A}$, each c_{w_i} $(i \geq 2)$ is the output of invoking the $\mathsf{PFS.Disguise}$ oracle on w_i, the challenge ciphertext $c_b = \mathsf{PFS.Disguise}(w_b)$, where $b \in \{0,1\}$ is chosen randomly by the challenger, and the adversary's output $b' \in \{0,1\}$.

From the definition of $\mathsf{PFS.Disguise}$, where $E : \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ is assumed to be a pseudorandom permutation family, we can re-write D_0 as follows

$$\begin{aligned} D_0 &= (sk_f, \\ &\quad w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &\quad c_{w_2}, \ldots, c_{w_{\mathsf{poly}(k)}}, \\ &\quad c_b, \\ &\quad b') \\ &= (sk_f, \\ &\quad w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &\quad (iv_2, E_{sk_f}(\mathcal{H}(w_2||msk) \oplus iv_2)), \ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(\mathcal{H}(w_{\mathsf{poly}(k)}||msk) \oplus iv_{\mathsf{poly}(k)})), \\ &\quad (iv_f, E_{sk_f}(\mathcal{H}(w_b||msk) \oplus iv_f)), \\ &\quad b') \end{aligned}$$

where iv_f and each iv_i is a randomly sampled initialization vector as defined in 2.6.1.

Step 1 - Input to PFS.Disguise Blockcipher is Pseudorandom

Let r_{w_2} be uniformly chosen, and define the hybrid distribution[5]

$$\begin{aligned} D_1 = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &(iv_2, E_{sk_f}(\mathbf{r_{w_2}} \oplus iv_2)), \ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(\mathcal{H}(w_{\mathsf{poly}(k)}||msk) \oplus iv_{\mathsf{poly}(k)})), \\ &(iv_f, E_{sk_f}(\mathcal{H}(w_b||msk) \oplus iv_f)), \\ &b') \end{aligned}$$

where D_1 is the same as D_0 except $(iv_2, E_{sk_f}(\mathcal{H}(w_2||msk) \oplus iv_2))$ is replaced by $(iv_2, E_{sk_f}(\mathbf{r_{w_2}} \oplus iv_2))$.

By our assumption that $\mathcal{H}$ is a random oracle, $\mathcal{H}$ maps every unique input to a uniformly at random output of length k. Therefore, since msk is random, $\mathcal{H}(w_2||msk) \oplus iv_2$ is computationally indistinguishable from $r_{w_2} \oplus iv_2$. Thus, $D_0 \approx D_1$.

Next, we define distributions $D_2, \ldots, D_{\mathsf{poly}(k)}$. Each consecutive D_i replaces the next $(iv_i, E_{sk_f}(\mathcal{H}(w_i||msk) \oplus iv_i))$ with $(iv_i, E_{sk_f}(r_{w_i} \oplus iv_i))$, where r_{w_i} is random.

$$\begin{aligned} D_i = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &(iv_2, E_{sk_f}(r_{w_2} \oplus iv_2)), \ldots, (iv_i, E_{sk_f}(\mathbf{r_{w_i}} \oplus iv_i)), \ldots \\ &\ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(\mathcal{H}(w_{\mathsf{poly}(k)}||msk) \oplus iv_{\mathsf{poly}(k)})), \\ &(iv_f, E_{sk_f}(\mathcal{H}(w_b||msk) \oplus iv_f)), \\ &b') \end{aligned}$$

By the same argument as before, for each consecutive pair of hybrid distributions, $D_{i-1} \approx D_i$. It follows that $D_0 \approx D_{\mathsf{poly}(k)}$.

[5]This represents a modified $\mathsf{Game}_{\mathsf{PFS}}$ between the challenger and the adversary, where the challenger responds to the first PFS.Disguise query, by replacing the input to E with a uniformly chosen random value.

Let r_{w_b} be uniformly chosen and define the distribution[6]

$$\begin{aligned} D_{\mathsf{poly}(k)+1} = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &(iv_2, E_{sk_f}(r_{w_2} \oplus iv_2)), \ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(r_{w_{\mathsf{poly}(k)}} \oplus iv_{\mathsf{poly}(k)})), \\ &(iv_f, E_{sk_f}(\mathbf{r_{w_b}} \oplus iv_f)), \\ &b') \end{aligned}$$

By the same argument as before, $D_{\mathsf{poly}(k)} \approx D_{\mathsf{poly}(k)+1}$. It follows that $D_0 \approx D_{\mathsf{poly}(k)+1}$.

Step 2 - PFS.Disguise **Ciphertexts are Pseudorandom**

Let u_2 be uniformly chosen. Define the following hybrid distribution[7]

$$\begin{aligned} D_{\mathsf{poly}(k)+2} = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &(iv_2, \mathbf{u_2}), \ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(r_{w_{\mathsf{poly}(k)}} \oplus iv_{\mathsf{poly}(k)})), \\ &(iv_f, E_{sk_f}(r_{w_b} \oplus iv_f)), \\ &b') \end{aligned}$$

where this distribution is like the last, except $(iv_2, E_{sk_f}(r_{w_2} \oplus iv_2)$ is replaced by (iv_2, u_2).

Suppose there exists a PPT algorithm that computationally distinguishes between $D_{\mathsf{poly}(k)+1}$ and $D_{\mathsf{poly}(k)+2}$. It follows, there exists a PPT algorithm **Dist** that takes a transcript from $\mathsf{Game}_{\mathsf{PFS}}$ as input and outputs 1 or 0. Without loss of generality, **Dist** guesses 1 if the transcript is from the distribution $D_{\mathsf{poly}(k)+1}$ and 0 if the transcript is from the distribution $D_{\mathsf{poly}(k)+2}$. It follows that **Dist** is correct with probability greater than $\frac{1}{2} + \mathsf{negl}(k)$ where the probability is taken over the coin tosses of $\mathcal{A}$ and the challenger.

Then, we can use **Dist** to construct a PPT algorithm PRPDistinguisher to computationally distinguish pseudorandom permutation families from random. Let $\mathsf{PRPDistinguisher}(\mathcal{O}^g)$ be a PPT algorithm that distinguishes a pseudorandom permutation g from a truly random permutation with oracle access to g using $\mathcal{O}^g$.

$\mathsf{PRPDistinguisher}(\mathcal{O}^g)$:

[6]This is the same modified $\mathsf{Game}_{\mathsf{PFS}}$ between the challenger and the adversary as in $D_{\mathsf{poly}(k)}$, except now the challenger responds with challenge ciphertext, compued by replacing the input of E with a uniformly chosen random value.

[7]This represents a modified $\mathsf{Game}_{\mathsf{PFS}}$ between the challenger and the adversary, where the challenger replaces the first call to E with a uniformly chosen random value.

1. Generate the transcript by running $\mathcal{A}$ and the challenger, using the hybrid distribution $t \in D_{\mathsf{poly}(k)+2}$ to represent the challenger's modified responses, except now replace the first invocation of the PRP E with a call to the oracle $\mathcal{O}^g(r_{w_2})$:

$$\begin{aligned} t = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &(iv_2, \mathcal{O}^{\mathbf{g}}(r_{w_2}), \ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(r_{w_{\mathsf{poly}(k)}})), \\ &(iv_f, E_{sk_f}(r_{w_b} \oplus iv_f)), \\ &b') \end{aligned}$$

   where each $r_{w_i}, iv_f, iv_i \xleftarrow{\$} \{0,1\}^k$

2. Output $\mathbf{Dist}(t)$

By the definition of **Dist** and since E_{sk_f} is replaced with the $\mathcal{O}^g$,

$$\begin{aligned} \Pr[\mathsf{PRPDistinguisher}(\mathcal{O}^g) = 1 \mid g \text{ is pseudorandom }] = & \\ &= \Pr[\mathbf{Dist}(t) = 1 \mid t \leftarrow D_{\mathsf{poly}(k)+1}] \\ &> \frac{1}{2} + \mathsf{negl}(k) \end{aligned}$$

where the probability is taken over coin tosses of PRPDistinguisher and **Dist**. Therefore, PRPDistinguisher is a computational distinguisher for pseudorandom permutations. This contradicts our pseudorandomness assumption for E.

Therefore, no such PPT algorithm **Dist** exists. It follows that $D_{\mathsf{poly}(k)+1} \approx D_{\mathsf{poly}(k)+2}$, from which it follows that $D_0 \approx D_{\mathsf{poly}(k)+2}$.

Next, let each hybrid distribution

$$D_{\mathsf{poly}(k)+i} \in \{D_{\mathsf{poly}(k)+2}, \ldots, D_{2 \cdot \mathsf{poly}(k)}\}$$

be defined as

$$\begin{aligned} D_{\mathsf{poly}(k)+i} = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}(k)}, \\ &(iv_2, u_2), \ldots, (iv_i, \mathbf{u_i}), \ldots, \\ &\ldots, (iv_{\mathsf{poly}(k)}, E_{sk_f}(r_{w_{\mathsf{poly}(k)}} \oplus iv_{\mathsf{poly}(k)})), \\ &(iv_f, E_{sk_f}(r_{w_b} \oplus iv_f)), \\ &b') \end{aligned}$$

where by pseudorandomness of E, we use the same argument as above to show that $D_{\mathsf{poly}(k)+i-1} \approx D_{\mathsf{poly}(k)+i}$. It follows that $D_0 \approx D_{2\cdot\mathsf{poly}(k)}$.

Step 3 - Challenge Ciphertext is Pseudorandom

Finally, let u_f be uniformly chosen and define the final hybrid distribution[8]

$$\begin{aligned} D_F = (&sk_f, \\ &w_0, w_1, \ldots, w_{\mathsf{poly}_2(k)}, \\ &(iv_2, u_2), \ldots, (iv_{\mathsf{poly}(k)}, u_{\mathsf{poly}(k)}), \\ &(iv_f, \mathbf{u_f}), \\ &b') \end{aligned}$$

Again, by the pseudorandomness of E, we use the same argument as above to show that $D_{2\cdot\mathsf{poly}_2(k)} \approx D_F$. Then, it follows that $D_0 \approx D_F$. Clearly, D_F is a random distribution over the coin tosses of the challenger and the adversary. Therefore D_0 is computationally indistinguishable from random.

Hence, any transcript of $\mathsf{Game}_{\mathsf{PFS}}$ is computationally indistinguishable from random. Specifically, the challenge ciphertext c_b and the outputs of all oracle queries are computationally indistinguishable from random. Therefore, any PPT adversary wins $\mathsf{Game}_{\mathsf{PFS}}$ with negligible advantage. □

[8]This represents a modified $\mathsf{Game}_{\mathsf{PFS}}$ between the challenger and the adversary, where the challenger now sends a challenge ciphertext that is sampled uniformly at random.

Chapter 4

Implementation of a Software Library and Command Line Interface for PKS and PFS

In this chapter we describe our implementation of PKS and PFS in the form a command line interface and a software library. The command line interface, implemented using the software library, is intended to be used as a standalone, deployable application to generate master keys, encrypt free text data files, extract keyword and frequency auxiliary keys, and apply extracted keys to encrypted data to perform keyword search and bag-of-words computations. Additionally, the software library enables developers to programmatically integrate obtained auxiliary keys to perform data analysis on encrypted data in their custom NLP algorithms. Together, the command line interface and the software library allow for both direct and programmatic use of PKS and PFS.

4.1 Preliminaries

Our implementation is written in the Golang, an open source and cross-platform programming language developed by Google [2]. Our CLI, named Alvis, is available in binary form for *macOS*, *Linux*, and *Windows* in both `x86-32` and `x86-64` architectures.

4.1.1 Dependencies

The most important dependency in our implementation is on the Golang standard cryptography library `crypto` [1]. Below is a list of the cryptography modules used in our system.

- `crypto/aes` - implements AES (2.7).
- `crypto/cipher` - implements block cipher operating modes, namely CBC (2.6.1).
- `crypto/rand` - implements cryptographic pseudorandom number generator.

- `crypto/sha256` - implements the cryptographic hash function SHA256 (2.2.1).

A complete list of our implementation's dependent packages is provided in appendix A.1.

4.2 Helpful Functions

Before we dive into the details of the software library implementation, we will describe several helpful functions that are used throughout our implementation.

Hash Function

We use the `crypto/sha256` package's implementation of SHA-256, denoted SHA256, as our cryptographic hash function.

Random Number Generation: rand

The rand function uses the `crypto/rand` package to pseudo-randomly generate bit strings. More specifically, rand(n) returns a pseudo-random bit string $y \in \{0,1\}^n$.

Base36 Encoding: b36Encode, b36Decode

The b36Encode, b36Decode functions use the standard Golang big number package (`math/big`) to convert byte arrays to strings in Base 36 and vice versa. The Base 36 alphabet is shown in 4.1.

$$\alpha_{b36} = \{0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\} \tag{4.1}$$

We use Base 36 to encode ciphertexts to be compatible with NLP algorithms. As we will see, our compute algorithms collate ciphertexts and plaintexts into a "partially decrypted" free text format that unaltered NLP algorithms can search on. Since many NLP algorithms first manipulate free text, such as transforming all words to lower case or filtering out punctuation, we are restricted to the limited Base 36 character set as opposed to a more compact encoding scheme like Base 64.

We can compute the compactness of Base 36 as follows. A single character (8 bits) in Base 36 represents $\log_2(36) = 5.17$ bits. Thus, for every 5.17 bits, 8 bits are needed. Thus, Base 36 expands a bit string by ≈ 1.55 times.

PKCS#7 Padding: Pad, Unpad

The Pad, Unpad functions implement the standard PKCS#7 padding scheme as defined in 2.6.2.

AES-CBC: Encrypt, Decrypt

The Encrypt, Decrypt functions implement the encryption, decryption operations using the AES block cipher together with the CBC block mode as defined in section 2.6.1.

4.3 Library Modules

Our implementation is comprised of the following four modules.

- **PKS** implements the PKS construction as specified by 3.3.3.
- **PFS** implements the PFS construction as specified by 3.4.3.
- **KFEncrypt** uses the master keys of PKS and PFS to encrypt free text.
- **KFCompute** applies auxiliary keys to encrypted free text to compute NLP algorithms.

Now, we will describe each module in detail.

4.3.1 PKS

As described in 3.3.3, the **PKS** implements the four algorithms $(Setup, Extract, Hide, Check)$. Below we provide the implementation details of each algorithm. Note that the implementation uses AES with key size 256 and block length 128.

PKS.Setup **and** PKS.Extract

PKS.Setup and PKS.Extract, shown in algorithms 1 and 2, follow the mathematical definition of 3.3.3 closely.

Algorithm 1 Generate the master secret key

```
1: procedure PKS.Setup
2:     msk ← rand(256)
3:     return msk
```

Algorithm 2 Extract a secret key for a keyword

```
1: procedure PKS.Extract_msk(w)                    ▷ w ∈ {0,1}*
2:     sk ← SHA256(w||msk)
3:     return sk
```

PKS.Hide

PKS.Hide, shown in algorithm 3, follows 3.3.3 but also encodes the ciphertext in Base 36.

Algorithm 3 Hide a word

```
1: procedure PKS.Hide_msk(w)                                   ▷ w ∈ {0,1}*
2:     sk ← PKS.Extract_msk(w)
3:     v ← 1^128
4:     c ← Encrypt_sk(v)
5:     c' ← b36Encode(c)
6:     return c'
```

PKS.Check

PKS.Check, shown in algorithm 4, follows 3.3.3 closely, but first decodes the ciphertext from Base 36.

Algorithm 4 Check if a ciphertext encrypts a plaintext keyword

```
1: procedure PKS.Check_sk(c)                                   ▷ c ∈ α_b36
2:     c' ← b36Decode(c)
3:     v_0 ← 1^128
4:     v_1 ← Decrypt_sk(c')
5:     if v_0 = v_1 then
6:         return true
7:     else
8:         return false
```

4.3.2 PFS

As described in 3.4.3, the **PFS** implements the three algorithms

$$(Setup, Disguise, Recognize)$$

Below we provide the implementation details of each algorithm and we also add two new algorithms PFS.Extract, which simply returns the secret frequency key, and PFS.Uncover which enables anyone with the msk to fully decrypt an ciphertext. Note that the implementation uses AES with key size 256 and block length 128.

PFS.Setup **and** PFS.Extract

PFS.Setup, shown in algorithms 5 and 6, follows the mathematical except generates an additional secret key sk_r. Additionally, we provide the PFS.Extract algorithm to pull out the frequency key, sk_f, from msk.

Algorithm 5 Generate the master secret key

1: **procedure** PFS.Setup
2: $sk_d \leftarrow \mathsf{rand}(256)$
3: $sk_r \leftarrow \mathsf{rand}(256)$
4: $sk_f \leftarrow \mathsf{rand}(256)$
5: $msk \leftarrow (sk_d, sk_r, sk_f)$
6: **return** msk

Algorithm 6 Extract the frequency secret key

1: **procedure** $\mathsf{PFS.Extract}_{msk}$
2: **return** $msk.sk_f$

PFS.Disguise

PFS.Disguise, shown in algorithm 7, follows 3.4.3 except it uses the additional secret key sk_r to produce a standard AES-CBC encryption of the plaintext words. The ciphertext components are concatenated and encoded in Base 36.

Algorithm 7 Hide a word

1: **procedure** $\mathsf{PFS.Disguise}_{msk}(w)$ ▷ $w \in \{0,1\}^*$
2: $c_0 \leftarrow \mathsf{Encrypt}_{msk.sk_r}(w)$
3: $c_1 \leftarrow \mathsf{Encrypt}_{msk.sk_f}(\mathsf{SHA256}(w||msk.sk_d))$
4: $c \leftarrow \mathsf{b36Encode}(c_0||c_1)$
5: **return** c

PFS.Recognize

PFS.Recognize, shown in algorithm 8, first decodes the ciphertext from Base 36 and separates the components. Note that in our implementation of PFS.Recognize, it does not take two ciphertexts as input. Instead it returns the y value from 3.4.3 that can be compared to the y' value of any other ciphertext. The reason for this is to enable an algorithm to avoid invoking PFS.Recognize to check equivalence of the underlying plaintexts for every pair of ciphertexts. Instead, the algorithm can do native "equals" comparisons just as it would do if the data was not encrypted.

PFS.Uncover

Given the master secret key msk, PFS.Uncover, uses the the additional sk_r to fully decrypt any ciphertext using the first component that is unused in 8.

4.3.3 KFEncrypt

The **KFEncrypt** module combines the implementations of the PKS and PFS schemes to encrypt free text words. The module is comprised of the following components.

Algorithm 8 Recognize a ciphertext by removing the randomized encryption layer

1: **procedure** $\mathsf{PFS.Recognize}_{sk}(c)$ ▷ $c \in \alpha_{b36}$
2: $c_0||c_1 \leftarrow \mathsf{b36Decode}(c)$
3: $y \leftarrow \mathsf{Decrypt}_{sk}(c_1)$
4: **return** y

Algorithm 9 Uncover a ciphertext by decrypting and returing the underlying plaintext

1: **procedure** $\mathsf{PFS.Recognize}_{msk}(c)$ ▷ $c \in \alpha_{b36}$
2: $c_0||c_1 \leftarrow \mathsf{b36Decode}(c)$
3: $n \leftarrow \mathsf{Decrypt}_{msk.sk_r}(c_0)$
4: **return** n

- data structures for storing master keys and extracted keys
- methods for generating and extract keys (via $\{\mathsf{PKS}, \mathsf{PFS}\}\mathsf{Setup}$ and $\{\mathsf{PKS}, \mathsf{PFS}\}\mathsf{Extract}$)
- in-place encryption (hiding and disguising) of free text words in custom data structures

Storing and Generating the Master Key

The master key data structure follows the format specified in 4.2, with the FrequencyMasterKey substructure. MasterKey is generated by invoking the PKS.Setup, PFS.Setup functions.

$$\begin{aligned} &\mathsf{MasterKey}\ \{ \\ &\quad \mathsf{msk}_{kw} : \{0,1\}^{256} \\ &\quad \mathsf{msk}_{fq} : \mathsf{FrequencyMasterKey} \\ &\} \end{aligned} \qquad \begin{aligned} &\mathsf{FrequencyMasterKey}\ \{ \\ &\quad d : \{0,1\}^{256} \\ &\quad r : \{0,1\}^{256} \\ &\quad f : \{0,1\}^{256} \\ &\} \end{aligned} \tag{4.2}$$

And each component of the msk structure is filled by invoking the statements in 4.3.

$$\begin{aligned} \mathsf{MasterKey.msk}_{kw} &\leftarrow \mathsf{PKS.Setup}(256) \\ \mathsf{MasterKey.msk}_{fq} &\leftarrow \mathsf{PFS.Setup}(256) \end{aligned} \tag{4.3}$$

Storing and Extracting the Auxiliary Keys

The keyword auxiliary key data structure has the format in 4.4, whereas the frequency auxiliary key is simply a bit string $\mathsf{FrequencyKey} \in \{0,1\}^{128}$ and does need a specific storage structure. Auxiliary keys are extracted using PKS.Extract and PFS.Extract.

$$\begin{aligned} &\mathsf{KeywordKey}\ \{ \\ &\quad \mathsf{word} : \{0,1\}^* \\ &\quad \mathsf{key} : \{0,1\}^{256} \\ &\} \end{aligned} \tag{4.4}$$

$$\begin{aligned} \mathsf{KeywordKey.key} &\leftarrow \mathsf{PFS.Extract}_{\mathsf{MasterKey.msk}_{kw}}(w) \\ \mathsf{KeywordKey.word} &\leftarrow w \end{aligned} \tag{4.5}$$

$$\mathsf{FrequencyKey} \leftarrow \mathsf{MasterKey.msk}_{fq}.f \tag{4.6}$$

In-Place Encryption: Hiding and Disguising Free Text Words

The last component of the **KFEncrypt** module enables a data learner to take an arbitrary data structure, hide and disguise the free text words, and replace the plaintexts with the ciphertexts in-place. This functionality maintains compatibility with existing NLP algorithms that already search over these custom data structures. To this end, we provide the following interface to that can be implemented for any data structure.

Searchable Interface

1. CountFreeTextRecords : $\varnothing \rightarrow \mathbb{N}$
2. GetFreeTextRecordAtIndex : $\mathbb{N} \rightarrow \mathcal{WL}^*$
3. SetFreeTextRecordAtIndex : $\mathbb{N} \times \mathcal{WL}^* \rightarrow \varnothing$

In turn, any data structure that provides an implementation for 4.3.3, denoted as $\mathsf{i} - \mathsf{searchable}$, we provide the HideAndDisguiseFreeText implementation shown in algorithm 10.

The main function of algorithm 10 is to replace every free text plaintext word with a ciphertext consisting of two sub-ciphertexts, one for each scheme. The result is that the structure of the custom data storage type is un-altered.

4.3.4 KFCompute

In 4.3.4 we use extracted keys to modify the encrypted free text data structures to make them searchable by NLP algorithms. Specifically, we will roughly unwind algorithm 10

Algorithm 10 Hide and Disguise plaintext

```
1: procedure HideAndDisguiseFreeText(ds, masterKey)        ▷ ds, a data structure that
   conforms to the i − searchable interface, masterKey as MasterKey
2:     n ← ds.CountFreeTextRecords()
3:     ctr ← 0
4:     while ctr < n do
5:         W ← ds.GetFreeTextRecordAtIndex(ctr)
6:         Let [w_0, ..., w_|W|] ← W
7:         C = []
8:         for all w_i ∈ [w_1, ..., w_|W_ctr|] do
9:             c_0 ← PKS.Hide_masterKey.msk_kw(w_i)
10:            c_1 ← PFS.Disguise_masterKey.msk_fq(w_i)
11:            C[ctr] ← c_0||c_1
12:        ds.SetFreeTextRecordAtIndex(i, C)
13:        ctr = ctr + 1
14:    return ds
```

limited to the set of obtained auxiliary keys, and end up with a partially decrypted free text.

The main function for algorithm 11 is to try to execute PKS.Check with each keyword key, and if the check succeeds then replace the combined ciphertexts with the single plaintext keyword stored in the keyword key structure. Otherwise, use PFS.Recognize with sk_f to replace the combined ciphertexts with the single, recognizable deterministic function of the plaintext. If the keyword key decrypts, the plaintext gives us both full knowledge of the ciphertext and the ability to compute frequency counts with it. If not, then at the least we replace a randomized ciphertext with a deterministic pseudorandom function of the plaintext. This allows a machine learning algorithm to always compute frequency counts with the possibility of discovering the keyword if there exists a compatible keyword key.

Note that algorithm 11 can also be invoked without the auxiliary key for frequency. The algorithm would just skip that step in the procedure.

In the next chapter we provide a case study in the medical field to show exactly how this mechanism can be used to run unmodified real NLP algorithms on encrypted data.

4.4 Performance

One of the key features of a system like ours must be performance. Scanning over tens of millions of plaintexts is computationally intensive, but when factoring in ciphertext expansion and decryption for each plaintext, a small slowdown at the lower level can make the system unusable. Since we rely solely on well known, popular cryptography primitives, our system performs well. In fact, on many machines some of our cryptographic primitives

Algorithm 11 Check and Recognize plaintext

```
1: procedure CheckAndRecognizeFreeText(ds, K, sk_f)        ▷ ds, a data structure that
   conforms to i − searchable, K is a set of extract keys as KeywordKey structures, and sk_f
   is the frequency secret key
2:     n ← ds.CountFreeTextRecords()
3:     ctr ← 0
4:     while ctr < n do
5:         C ← ds.GetFreeTextRecordAtIndex(ctr)
6:         Let [c_0, ..., c_|C_ctr|] ← C
7:         W = []
8:         for all c_i ∈ [c_1, ..., c_|C_ctr|] do
9:             Let c_i,0||c_i,1 ← c_i
10:            for all sk_i ∈ K do
11:                Let c_i,0||c_i,1 ← c_i
12:                if PKS.Check_{sk_i.key}(c_i,0) then
13:                    W[ctr] = sk_i.word
14:                    continue
15:            if W[ctr] = "" then
16:                W[ctr] = PFS.Recognize_{sk_f}(c_i,1)
17:                continue
18:        ds.SetFreeTextRecordAtIndex(i, W)
19:        ctr = ctr + 1
20:    return ds
```

are implemented as hardware instructions, increasing performance substantially. Next, we provide a detailed breakdown on the performance of the functions in our implementation.

4.4.1 Runtime Performance

We ran our benchmarks on a "General Purpose" Amazon Web Services (AWS) `m4.large` Virtual Machine, that run on 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) processors [3]. Benchmarks were run 1,000,000 times and averaged to calculate time per operation. All units are in nanoseconds per operation (ns/op).

Note that the key size is 256 bits and block length is 128 bits. The encrypt/decrypt benchmarks in table 4.1 are using $\mathsf{AES}-\mathsf{CBC}$ (2.7.1) on a single block message. The hash function of the concatenation of two strings, $\mathsf{SHA256}(a||b)$, is benchmarked with input lengths $|a| = 128$ and $|b| = 256$ to provide an upper-bound on typical usage lengths.

While we have provided many benchmarks to analyze the performance of our system, primarily two benchmarks actually determine if our system is usable or not: PKS.Check and PFS.Recognize. CheckAndRecognizeFreeText (algorithm 11), is the bulk of the work a data leaner must perform to compute NLP on the encrypted data.

Table 4.1: Cryptographic Primitive Benchmarks

rand	3421	ns/op
AES − CBCEncrypt	3531	ns/op
AES − CBCDecrypt	1225	ns/op
SHA256$(a\|\|b)$	3048	ns/op

Table 4.2: PKS Implementation Benchmarks

PKS.Setup	3427	ns/op
PKS.Extract	3149	ns/op
PKS.Hide	6324	ns/op
PKS.Check	1261	ns/op

For each free text word in the data set, the learner must do a PFS.Recognize operation and then iterate over all keyword search keys in possession and do PKS.Check operation. Assuming that free text words are constant sized (we can approximate the range of English words from $40 - 80$ bits, or $5 - 10$ characters). If n is the number of free text words in the data set, w is the number of keyword keys, then the complexity of this function is $O(n \times (w + 1)) = O(nw)$. Since we designed our system to support many keyword search keys, the PKS.Check operation must be extremely fast. To this end, we also benchmarked its main dependency, a single AES − CBC − Decrypt on one block. Essentially, our goal is to have the PKS.Check benchmark be as close as possible to the baseline AES − CBC − Decrypt benchmark.

As we see in tables 4.2 and 4.3, a AES − CBCDecrypt operation is 1225 ns versus the 1261 ns for a PKS.Check operation. Thus our algorithm spends an additional 36 ns beyond the baseline decrypt operation.

Relative to the benchmarking machine, can do 793,021 PKS.Check operations per second.

4.4.2 Ciphertext Expansion

In addition to the runtime performance, the size of the ciphertexts is very important to performance. We can look at the ciphertext expansion rate, or the size increase, after replacing plaintext words with both PKS and PFS ciphertexts.

A PKS ciphertext is the encryption of a single 128 bit block, followed by an appended

Table 4.3: PFS Implementation Benchmarks

PFS.Setup	10275	ns/op
PFS.Disguise	10065	ns/op
PFS.Recognize	1361	ns/op

IV, and then all encoded in Base36. Thus, 256 bits are encoded in Base 36. By the expansion ratio in 4.2, $1.55 * 256 \approx 397$ bits are needed.

A PFS ciphertext is a 32-byte SHA256 hash. After encryption and Base 36 encoding, it becomes an IV plus 3 blocks (after padding) or $128 * 4 * 1.55 = 794$ bits.

Thus, combining these two ciphertexts into one, we get that the total expansion is 1191 bits or about 150 bytes. For example, for each English word (which ranges between 5 and 10 bytes), ciphertexts are **15** to **30** times bigger.

4.5 Alvis: The Command Line Interface for PKS and PFS

Alvis is the command line interface (CLI) for generating the master key, encrypting (hiding and disguising) arbitrarily structured plaintext data structures, extracting auxiliary keys, and applying these keys to (checking and recognizing) arbitrarily structured ciphertext data structures.

Alvis is a ready-to-use application that data owners can use to encrypt data and issue auxiliary keys to give to data learners. Learners can use Alvis and issued keys to run their NLP algorithms over the encrypted data.

The remainder of this section describes the Alvis interface and how to use it.

```
demo % alvis
NAME:
   Alvis - A command line interface for Private-key {Keyword,Frequency} Search

USAGE:
   alvis [global options] command [command options] [arguments...]

VERSION:
   0.1

COMMANDS:
     setup    Generate the master key
     extract  Extract a secret key for keyword,frequency search
     encrypt  hide and disguise free text in data files
     decrypt  check data files for keywords and recognize repeated plaintexts
     uncover  Uncover a frequency ciphertext
     stats    Number of free text words in data files
     help, h  Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --help, -h     show help
   --version, -v  print the version

demo %
```

Figure 4-1: The Alvis help screen showing all the commands.

4.5.1 Serialization

To serialize data structures and write them to disk we use Java Script Object Notation (JSON) [5]. JSON is natively compatible with data structures in Golang.

4.5.2 Setup

The setup command,

```
alvis setup -out master.key
```

generates the master key for the system and stores it in the structure from 4.2, where the flag `-out` specifies the file path to write the data structure to. As an example, the file contents of `"master.key"` look as follows.

```
{
    "FrequencyKey": {
        "DetachedKey": "53nl8OklBxz9waF0KpB422O2yjSe68sLZvBu8Xvesso=",
        "InnerKey": "EctboHEQof9hcEHGPola6itq/Lu+z+JI3YFesd/Cgww=",
        "OuterKey": "9hOMyLrczDPt9mSi9wFqNoX7xzAx1OVo4RzuloumqSE="
    },
    "KeywordKey": {
        "Key": "Oh6pI6vsDjcs5OAz7QwsS+LlG84YCaCQsshSYr8jP78="
    }
}
```

Note that symmetric keys (in the space of $\{0,1\}^{256}$) are Base 64 encoded.

4.5.3 Extract

The `alvis extract` command uses the $\{\mathsf{PKS}, \mathsf{PFS}\}\mathsf{Extract}$ algorithms as specified in algorithms 2 and 6 to extract keyword and frequency search keys.

Keyword Keys

To extract keyword keys, invoke the following command:

```
alvis extract keyword -msk master.priv -words words.txt -out-dir keys
```

where once again `-msk` specifies the path to the master key. The `-words` flag specifies a text file comprised of keywords (each on a newline) for which an auxiliary key is requested. The `-out-dir` specifies a directory for where to write the KeywordKey (4.5) structures. For example, a words file containing

```
ejection
fraction
is
of
patient
doctor
```

```
paralysis
```

generates a directory of keys as shown in figure 4-2.

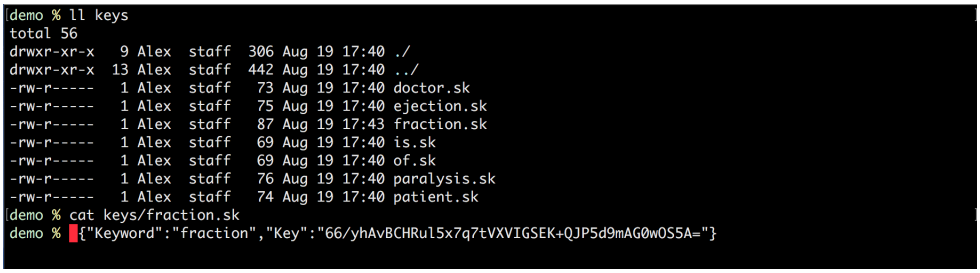

Figure 4-2: First, list the extracted keyword key files. Next, show the contents of **fraction.sk**.

Frequency Keys

To extract the frequency key, invoke the following command:

```
alvis extract frequency -msk master.priv -out freq.sk
```

where `-out` specifies the file path to write the bytes of the extracted frequency key.

4.5.4 Encrypt

The encrypt command,

```
alvis encrypt
        -msk master.priv
        -data-dir /a/path/to/records/
        -out-dir /a/path/to/encrypted/records
```

uses the master key file, as noted above, to run the encryption procedure (otherwise known as hide and disguise or algorithm 10). The `-data-dir` flag specifies which data files to pull free text words from. The `-out-dir` specifies where to write the modified, now free text hidden and disguised, data files.

4.5.5 Decrypt

The decrypt command,

```
alvis decrypt
        -key-dir keys/
        -freq-key freq.sk
        -freq-index /a/path/to/index/file -data-dir /a/path/to/encrypted/data/
        -out-dir /a/path/to/partially-decrypted-searchable/data
```

uses the master key file, as noted above, to run the decryption procedure (otherwise known as check and recognize) in algorithm 11. The arguments in the command are as follows.

- `-key-dir` specifies the directory of keyword key files, as shown in 4-2
- `-freq-key` specifies the path to the frequency key
- `-freq-index` specifies the file path to store the index mapping $c_1 \rightarrow c_0$ (in algorithm 7)
- `-data-dir` flag specifies which data files to pull the hidden and disguised free text words from
- `-out-dir` specifies where to write the modified, now checked and recognized, data files.

Note that `freq-index` is necessary to map the deterministic part of the frequency ciphertext to the randomized part. The randomized part is need to decrypt back to the plaintext (Uncover - algorithm 9).

4.5.6 Uncover

The uncover command,

```
alvis uncover   -msk master.priv
                -freq-index /a/path/to/index/file
                -file /a/path/to/recognized/ciphertexts/file
```

uses the master key file, as noted above, to run the uncover operation as specified by algorithm 9. The arguments in the command are as follows.

- `-freq-index` specifies the file path to store the index mapping $c_1 \rightarrow c_0$ (in algorithm 7)
- `-file` specifies a file that has recognized frequency ciphertexts, that is the decrypted c_1 from algorithm 7.

In the next chapter we document our case study of using Alvis on a real medical patient data set for which we run a suite of NLP algorithms that utilizes the auxiliary keys in our system to compute on partially encrypted data.

Chapter 5

Computing on Encrypted Patient Data

In this chapter we describe how we applied our command line interface, Alvis, to a suite of natural language processing algorithms that were successful on unencrypted medical patient records. Our main results is that we can use Alvis to run the unmodified suite of NLP algorithms on a encrypted patient data while achieving the same results as if the algorithms were run on unencrypted data.

The remainder of the chapter is outlined as follows. First, we give the background on the patient data and the high-level summary of what the NLP algorithm was able to achieve. Next, we explain the privacy problems that prevent wider-scale access to this kind of data for machine learning researchers. Then we dive into details of the patient data structure and specific classification goals of the NLP algorithm. Finally, we explain how the machine learning researcher and the hospital data administrator fit our two party model of data owner and data learner. We then conclude by demonstrating the usage of Alvis to both encrypt and compute on patient data using the unmodified suite of NLP algorithms.

5.1 Background

The NLP algorithm in this case study seeks to learn about patients with heart failure problems that received Cardiac Resynchronization Therapy (CRT). While this a successful therapy for a majority of patients, about one third of CRT patients do not experience positive results [14]. More interestingly, the causes of failure are not well understood [14].

One reason for the difficulty in understanding CRT failure conditions is the way that the clinical results are recorded in patient records. Record keeping, while electronic, leaves a lot to be desired for recording specific patient results, especially during CRT treatment.

Data is stored in many formats, and structured data often only contains a limited number of important metrics. This means that the bulk of information, which could potentially reveal CRT failure reasons, is hidden in free text doctor notes. Thus, clinical researchers would need to manually read this data to determine causes, a task that is infeasible for a large number of patient records [16].

5.1.1 NLP on Patient Data Reveals Important Information

Natural Language Processing (NLP) and is used to solve this exact problem. With NLP, clinical researchers can programmatically process the free text doctor's notes and extract common threads over many patient record files.

In our specific case study, this is exactly what a team of machine learning researchers, Freel, Haimson, and Traub from the Massachusetts Institute of Technology (MIT), and a clinical research doctor, Lindvall from the Massachusetts General Hospital (MGH), accomplished [16]. Specifically, the team's suite of NLP algorithms and analysis improved the prediction accuracy for the success of CRT treatment by 9% [16]. While this doesn't completely solve the problem, it could save millions of dollars for both hosptials and patients and possibly prevent a patient from undergoing an intensive treatment that will ultimately fail. We abbreviate the team's suite of algorithms as **FHTL**.

The team ran their algorithm on only about 900 patients from which the above results were achieved. A larger data set would likely strengthen the results and improve the prediction success rate.

5.1.2 Privacy Regulations Prevents Large Scale Data Access

One major issue that prevents access to larger patient data sets is the inherent privacy problem that comes with releasing data to an external research group, such as MIT machine learning researchers. Hospitals are bound to protect the PII of patients, and in particular, free text doctor's notes could contain a lot of sensitive data that is hard to anonymize programmatically. For example, the note could mention details about where the patient lives or the patients family members. Such information is difficult to sensor, and manual anonymizing is infeasible. Thus, hospital's are hesitant to give out large patient data sets which in turn stunts the discoveries made by good-intentioned external research parties.

This is the primary motivation for introducing cryptography as a possible solution to the problem. The main question, as seen by hospitals and external researchers is, can hospitals encrypt patient data such that external researchers can learn from the data without discovering the PII of patients?

In the remainder of the chapter, we explain how we used Alvis to facilitate search on encrypted data alongside the FHTL machine learning algorithm. We show that our system is an important, practical step to answering the question above.

5.1.3 Hospital: Data Owner, Researcher: Data Learner

First, we frame the medical case study in our two party model, data owner versus data learner. It is easy to see that the hospital is the data owner, maintaining a set of patient records. We note that this is a slight simplification, as in some cases there exists a company (like Partners Healthcare) that administers patient record keeping systems for multiple hospitals. In this case, the administering party is the data owner, as they are responsible for keeping the PII of patients private.

The external (machine learning) researchers are the data learning party. We emphasize that the data learning party is specifically separate from the hospital and thus is not able to view PII of patients.

As the results of 5.1.1 indicate, the collaboration between computer scientists like machine learning researchers and medical institutions is essential to transforming the large amounts of raw patient data siting in hospital databases into knowledge.

Therefore, this perfectly fits our model. The data learning party or the MIT machine learning researchers wish to collaborate with a medical institution, like MGH, to learn about why CRT treatments fail and how to predict their success.

5.2 Patient Data Records

First, we will introduce the structure of the patient data records to better explain the kinds of information they contain and the format they use. Each patient record is stored in a JSON file (4.5.1), containing both structured and unstructured (free text) information about each patient visit to the hospital. In table 5.1 we include the data type breakdown provided by Freel et al [16] for their sample of patients.

We note that the the structured segments of the file are not encrypted as they have already been de-identified. Structured fields are often computer generated and therefore consistent which makes it simple to programmatically remove PII. Thus, we continue by focusing only on unstructured free text data.

Freel et al [16] note that patient records contain roughly ten times more structured data than free text documents, but when comparing the number of sentences in free text to the number of structured entries, they estimate that roughly one third of CRT related infor-

Table 5.1: Breakdown of Patient Data Types [16]

Data Type	Contained Information	# Documents	# Entries/Sentences
Structured	Inpatient/Outpatient Duration of stay ICD-9 Code Lab type Value High/Low indicator	3,100,000	44,000,000
Free Text	Relevant labs (EF, QRS) Clinical characters (LBBB, sinus rhythm) Cardiologist notes Summary of lab values Symptoms Family history Social history	245,000	26,000,000

mation is stored in free text sentences. They offer the following examples to highlight this point.

Example 5.2.1. "This is a 54-year-old woman with end stage heart failure secondary to Chagas disease. Her main symptoms are shortness of breath, chest discomfort, anxiety, and existential distress." [16]

Example 5.2.2. "A very pleasant 68-year-old gentleman with a history of ischemic cardiomyopathy presented with class III symptoms of heart failure, has had an upgrade of his device to biventricular implantable cardioverter-defibrillators, currently in sinus rhythm." [16]

We see that both of the 5.2.1 and 5.2.2 examples contain a lot of information about each patient's respective conditions. We also note that example 5.2.2 contains the doctor's, likely irrelevant commentary, that the patient was "very pleasant". While this information is likely innocent and unrevealing, it is easy to see how other similarly irrelevant information could indicate PII.

5.3 FHTL's Free Text Search Methods

The FHTL machine learning algorithm utilizes several NLP models, summarized below, to extract information from free text patient notes [16].

1. Clinical Value Extraction (CVE). Use of regular expressions to extract already-known prediction factors for CRT like NYHA Class, LVEF, QRS, LBBB, and Sinus Rhythm.

2. Bag of Words (BOW). Use frequency analysis of bigrams or n-grams to reduce the number of relevant sentences across patient records by locating similarities and overarching themes. No previous knowledge expected.

3. Paragraph Vectors. Represent free text paragraphs as variable length sequences of words to form fixed-sized feature vectors. These are used to predict a given word in a sentence (or sequence of words) given the fixed dimensional feature vectors.

4. Stop Words. Remove frequently, often meaningless, English words from the free text notes such that the remaining words are more meaningful and more likely to be related to underlying data analysis goals.

Next, we discuss how the PKS and PFS schemes are compatible and manage to maintain the information obtained in the above methods while hiding PII.

5.3.1 CVE

The regular expressions used in FHTL are simple enough such that they can be approximated with a modest-sized set of keywords to be extracted using PKS. Below we give some example regular expressions and a list of keywords used to approximate them.

- **Left Ventricular Ejection Fraction (LVEF)**:

  ```
  (?:ef|ejection fraction)\s*(?:of|is)?[:\s]*([0-9]*\.?[0-9]*)
  ```

  can be approximated requesting keywords

  ```
  ef, ejection, fraction, of, is, \%
  ```

  and keywords $\forall i \in \{0, \ldots, 100\}$, i and $i\%$.

- **Ischemic**:

  ```
  non(?:-| )ischemic
  ```

  can be approximated requesting

  ```
  ischemic, non-ischemic, non
  ```

- **New York Heart Association Classification (NYHA Class)**:

  ```
  class (i+v*|[1-4])(?:(?:/|-)(i+v*|[1-4]))?
  nyha", "nyha(?: class)? (i\+v*|[1-4])(?:(?:/|-)(i+v*|[1-4]))?
  ```

can be approximated requesting

```
class, of, nyha, I, II, III, IV
```

5.3.2 BOW and Paragraph Vectors

PFS, our general bag-of-words (frequency analysis) scheme, perfectly fits the bag of words model and the representation of paragraphs as length-N word sequences.

As noted in our discussion of PFS, the PFS.Recognize operation transforms a ciphertext into a deterministic string to enable counting plaintext repetitions. While the underlying FHTL algorithm might not see the plaintext words, only a deterministic function of them, later the PFS.Uncover can be used by the hospital administrator to inspect the output of a BOW search.

As shown in CheckAndRecognize, algorithm 11, the deterministic function of the ciphertext is used to replace the randomized ciphertexts in-place. This maintains that the "partially decrypted" resulting free text, which FHTL will search over, is still in paragraph form, translating the original plaintexts into either the same plaintexts (for extracted keyword keys) or deterministic functions of the plaintexts (for an extracted frequency key.)

5.3.3 Stop Words

Stop words are English words that contain little to no information like "about, above, across, after, afterwards, again". Stop words can easily be filtered by requesting keyword keys for each stop word. We note that this is a perfect example of the data owner party (the hospital) releasing or white listing certain words known to be harmless for the data learner to filter out. That is, FHTL will see stop words the same way they do as when searching over plaintext data and can remove them to leave behind the more meaningful, likely "covered" but recognizable deterministic ciphertexts.

5.4 Alvis Implements the Searchable Interface for Patient Data

The first step is to describe how Alvis interacts with patient data files as they are in a custom format. To this end, we implemented the Searchable interface as described in 4.3.3 and added it to the Alvis implementation.

The Searchable methods, CountFreeTextRecords and {Get, Set}FreeTextRecordAtIndex, are implemented by iterating over every hospital visit entry in the patient data file, ignoring all

entries except those that contain a "free_text" field name. To get or set free text words at an index i, simply find the i^{th} "free_text" field, and either read or write the free text words in that entry. Note that as a precursor, we implemented a function to remove all unwanted characters, make all text lower-case, and split the free text paragraphs into an ordered list. Thus the list of words taken from a free text paragraph is cleansed of all bad characters prior to processing for encryption.

5.5 Using Alvis in Practice

Our main result is that we can use our software to run the unmodified FHTL program on encrypted patient record files, and specifically, based on trials over a small subset of patient records we have found that executions of FHTL on the encrypted patient files using Alvis, produce the same outputs as executions of FHTL on the unencrypted patient files.

We note that our results are based on running trials of our software and the FHTL algorithm on only a small subset of the original patient data files. The original work by Freel et al [16] used about 900 patient data files, while we were limited to 10 (deceased) patient files due to access restrictions and privacy regulations. We note that FHTL was designed to run on small datasets [16] and that our trials indicate that the our software will work on larger data sets, however more auxiliary keyword keys may be needed to show that the execution of FHTL on encrypted data classifies as well as it would on plaintext data.

For the remainder of the chapter we will walk through the process of using Alvis the run the FHTL machine learning algorithm on encrypted free text patient data files. This workflow closely follows that of section 4.5. In most cases Alvis integrates as pre-processing or post-processing step, meaning it is not necessary to make any changes to FHTL.

5.5.1 Hospital Generates Master Key

The first step is for the hospital administrator to execute the setup command to generate a master secret key. Note that this master key must be kept in a safe location and its secrecy essential because it can be used to decrypt all encrypted patient data. Figure 5-1 shows an example invocation of the setup command and the associated master key that is created.

```
hospital % alvis setup -out "hospital.master.key"
hospital % cat hospital.master.key
{"KeywordKey":{"Key":"4mLJ7mMZqoCzLy+trEyDt5e7ynqL4jiNjS/PoSxFad8="},"FrequencyKey":
{"InnerKey":"YmWOxpFoKFktilXjiiggB3kWiuYKwtJcByWtaMhch2I=","OuterKey":"DvPV3MbrgijujWYaZX+5Zx
+0ez3oL3d8TuaV7zm8nDg=","DetachedKey":"jnx+mXNV2CoDvjHJsA34YT98dkS+b3c/OB7eLFee70c="}}
```

Figure 5-1: The hospital generates the master secret key.

5.5.2 Hospital Encrypts

In the encryption step, the hospital administrator encrypts patient record files. Specifically, Alvis uses the searchable interface for patient data files, as explained in 5.4, to extract free text words from the patient file. The hospital executes the command figure 5-2 to transform every ordered list of free text words into an ordered list of PKS and PFS ciphertexts using the HideAndDisguiseFreeText algorithm, as explained in section 10.

```
hospital % alvis encrypt -msk "hospital.master.key" -data-dir "data/patients/" -out-dir "data/encrypted_patients"
hospital % head -n 10 data/encrypted_patients/EMPI_1.json.enc
{
    "Car": [
        {
            "date": "Report_Date_Time",
            "doctor": null,
            "free_text": {
                "frequency_enc": [
                    "270mb5gy128v70t9ep3aval5w98q5fshvznvd5m5s1lexvukascplf4jw7j5oieafkegblnpaluk5ezql95ropgua555k4f5ggd8",
                    "1sha7flyemnbwu851a8kk6bef4f1tflf89en21e7ettmptz14xae9sr4u0oexo34ymaz82eoylo4kemxy96cooyfrk64vastjivs",
                    "1n556nw6gbx42smvdjj3x30p314ob4ro6p3xxxlalrqfp7bnq0hborqng86joyqce3to0hnk35n63wppgujaix0hnpzxmf0htnyj",
hospital % grep "keyword_enc" -A4 data/encrypted_patients/EMPI_1.json.enc | head -5
                "keyword_enc": [
                    "9ui2g8z849r1likb4rba5rrp2u9ml5c2kmews13yncvusvo4f9",
                    "ajz4pibganlybz99nn3u9mqmcoc9slz6i1jeazz5d8w541enj2",
                    "768wdxejedoceuvd6z9o5spe9vay6hmlwl2akre1yqefwr0o5j",
                    "bbl8wkmwdv51sc9fahflwna8wfknrdj6amyafznf8hdvzh051n",
hospital %
```

Figure 5-2: The hospital encrypts patient data files. The next two commands show excerpts from an encrypted patient file.

Figure 5-2 also shows the two types of ciphertexts (PKS and PFS) in the resulting encrypted free text notes. Afterwards, the encrypted patient files are published to some storage service where the machine learning researchers can download them.

5.5.3 Researcher Requests Auxiliary Keys

Next, in the auxiliary key request step, the researcher requests keys for keywords and the frequency key. Specifically for the keywords, in our case study, the MIT machine learninger team would request search keys for approximating the regular expressions in section 5.3.1 and the stop words as explained in section 5.3.3.

The machine learning researchers can package their requested keywords in a plaintext file, denoted `keywords.txt`, where each keyword is on a new line. Figure 5-3 shows an excerpt of what such a file could look like. Finally, the researcher sends `keywords.txt` to the hospital for approval.

5.5.4 Hospital Grants or Rejects Search Keys

Upon approving the list keywords in `keywords.txt` and approving the ability to compute a frequency count, the hospital administrator executes the commands in figure 5-4 and 5-5 to extract all the corresponding keyword keys and a frequency key.

The hospital administrator then sends the resulting directory of keyword key files and the frequency key file back to the researcher.

```
hospital % head -10 data/keywords.txt
ef
ejection
fraction
is
of
the
0%
1%
2%
3%
hospital % tail -10 data/keywords.txt
with
within
without
would
yet
you
your
yours
yourself
yourselves
hospital %
```

Figure 5-3: The researcher compiles a list of requested keywords for the hospital.

```
hospital % alvis extract keyword -msk "hospital.master.key" -words "data/keywords.txt" -out-dir "keyword_keys/"
hospital % ls keyword_keys/
%.sk            41%.sk          75.sk           and.sk          fire.sk         neither.sk      thence.sk
0%.sk           41.sk           76%.sk          another.sk      first.sk        never.sk        there.sk
0.sk            42%.sk          76.sk           any.sk          five.sk         nevertheless.sk thereafter.sk
1%.sk           42.sk           77%.sk          anyhow.sk       for.sk          next.sk         thereby.sk
1.sk            43%.sk          77.sk           anyone.sk       former.sk       nine.sk         therefore.sk
10%.sk          43.sk           78%.sk          anything.sk     formerly.sk     no.sk           therein.sk
10.sk           44%.sk          78.sk           anyway.sk       forty.sk        nobody.sk       thereupon.sk
100%.sk         44.sk           79%.sk          anywhere.sk     found.sk        none.sk         these.sk
100.sk          45%.sk          79.sk           are.sk          four.sk         noone.sk        they.sk
11%.sk          45.sk           8%.sk           around.sk       fraction.sk     nor.sk          thick.sk
11.sk           46%.sk          8.sk            as.sk           from.sk         not.sk          thin.sk
12%.sk          46.sk           80%.sk          at.sk           front.sk        nothing.sk      third.sk
12.sk           47%.sk          80.sk           back.sk         full.sk         now.sk          this.sk
13%.sk          47.sk           81%.sk          be.sk           further.sk      nowhere.sk      those.sk
13.sk           48%.sk          81.sk           became.sk       get.sk          of.sk           though.sk
14%.sk          48.sk           82%.sk          because.sk      give.sk         off.sk          three.sk
```

Figure 5-4: The hospital extracts keywords from the approved list and writes each keyword key to a file in the specified keyword key directory.

5.5.5 Researcher Partially Decrypts Patient Files

Upon receiving the directory of keyword keys and the frequency key, the researcher executes the command shown in figure 5-6 on the published, encrypted patient files (5.5.2) using the keyword and frequency keys. This command invokes the CheckAndRecognizeFreeText algorithm described in 11, using the same searchable interface explained in above in 5.4. As we can see the excerpt from the partially decrypted patient file in figure 5-6, contains a paragraph with mixed deterministic ("recognized") ciphertexts and plaintext keywords for which the researcher has a keyword key to "check".

5.5.6 Researcher Runs FHTL on Partially Decrypted Patient Files

After "partially" decrypting as described above, the researcher can now run the unmodified FHTL machine learning algorithm on the patient files in the created `searchable_patients/`

```
hospital % alvis extract frequency -msk "hospital.master.key" -out "frequency.sk"
hospital % cat frequency.sk
?????(?fe?g?{=?/w|N??9??8
hospital %
```

Figure 5-5: The hospital extracts the frequency key and writes it to a specified file.

```
hospital % alvis decrypt -key-dir "keyword_keys/" -freq-key "frequency.sk" -data-dir "data/encrypted_patients/" -out-dir "data/
searchable_patients/"
^C
hospital % head -20 data/searchable_patients/EMPI_1.dec.json
{
    "Car": [
        {
            "date": "Report_Date_Time",
            "doctor": null,
            "free_text": "o1h5bweowvkvt314gvw6ds7nq jqh0ko6bstqilxa4eh66yes7k l2tgsh0cse4hlrkcykyu263hn tnbap5mmxqs6i8czyhqxkemwm
krnon9bc03pj1hz0zv42lmqu2 93wwcw86y0ragj8ct34qqpryyzaig6acyv0v6glayon27tw8sp 8wk6gavauorgnnq3uvrdzv2zasfsajn90pk9z2ikv46662rs2xo
ds0h5l2dmsu6re0grqejmj96 lo5zxbyvuglt8p32yuvqufz11 sdj2pyhfsdq99d8clmv018kbs the jivrx4y4bhvfkmxjyq1xg3vtf ejection fraction is 10%
rw2agbw2m5tr10f2a3h4gi626 g3nidf7ry9350dqq43bwm43ar and frkhyetjy815adceekd2kihbn gwhcmao9bogwygz77cxret9u1 there is no
l8bxergf6i52elhjr82xrf18b of rw2agbw2m5tr10f2a3h4gi626 i41zgki17k7ztqknunpu7twuy there is l8bxergf6i52elhjr82xrf18b of a
ro3s9qp3y0wgpccjathzdd59e i41zgki17k7ztqknunpu7twuy there is kefk8a25bmhpqru4k4kipdib2 sdj2pyhfsdq99d8clmv018kbs
kefk8a25bmhpqru4k4kipdib2 patient tlgmpk5qg6a0tvw16dj7uab3o jb1ssfz5jsiv3x061x7fkwd2i f7d8qm2zwpr8ro3539n6s0kir
l2tgsh0cse4hlrkcykyu263hn i8rpgzuwtk1hya5uclwdyhzzr p8fbcde8n4b4oc5lkej1skkrz 7 31% pyo5cj6x6f7ns4see0mewpaq7 2 50
tzc8mj1hipthcb4o358iq0yy7 gwdufs1aqak9mgau29dx85jza pku7v7jmxdwbljl4oba53eops nn0ro6on48bl64z3wwbw2wr5i in8bojesduh12sfcpqn2dpmeb
k6ebtl1jbdektw9j6bug9010m mp3fdr2zem1iaea70m6l001qz all i8rpgzuwtk1hya5uclwdyhzzr mp3fdr2zem1iaea70m6l001qz
anything in8bojesduh12sfcpqn2dpmeb kq95qkmodnvunfqyoc6edacz0 cardiac nn0ro6on48bl64z3wwbw2wr5i alright
```

Figure 5-6: The researcher "partially" decrypts the encrypted patient files. An excerpt of a partially decrypted patient file is then shown.

directory. Since free text notes still "look" like plaintext words, due to the unchanged paragraph structure, deterministic Base 36 encoded ciphertexts, and decrypted keywords, the FHTL algorithm works as it would on completely plaintext data except now it outputs partially encrypted results. An excerpt of the output computed by running FHTL is shown in figure 5-7. While the excerpt in figure 5-7 shows a bag of words output, other NLP

```
2016-08-07 23:30:06,588 - DaemonLog - INFO - Number of column names: 5724
2016-08-07 23:30:06,604 - DaemonLog - INFO - 100 biggest theta components of last CV run:
2016-08-07 23:30:06,604 - DaemonLog - INFO - -0.0650191768284 car_ngram__or3nqn65uxsbrjuct6c5m0x2s aortic
2016-08-07 23:30:06,605 - DaemonLog - INFO - -0.0650191768284 car_ngram__stenosis after
2016-08-07 23:30:06,605 - DaemonLog - INFO - -0.0650191768284 car_ngram__tc2pzkb3xgqmu7kytjjmga2kp or3nqn65uxsbrjuct6c5m0x2s
2016-08-07 23:30:06,605 - DaemonLog - INFO - -0.0650191768284 car_ngram__thtijhlnej2zqx5dng0w2lmpd systolic
2016-08-07 23:30:06,606 - DaemonLog - INFO - -0.0557307229958 car_ngram__l8eq4b79ct209iw4xw4h5r86v qqv9hwo97i6ph32gta05hkjb0
2016-08-07 23:30:06,606 - DaemonLog - INFO - -0.0557307229958 car_ngram__non systemic
2016-08-07 23:30:06,606 - DaemonLog - INFO - -0.0464422691631 car_ngram__tyb8w2sqht9j7nym8m1yah3x5 ub4yqse4nrk1eweoj1qxfvb99
2016-08-07 23:30:06,606 - DaemonLog - INFO -  0.04439433262   car_ngram__19 fn7br9il1ug5ao727d5n4vuh0
2016-08-07 23:30:06,606 - DaemonLog - INFO -  0.0416740781883 car_ngram__go2o6sd8cepbjqv174ergfcw9 is
2016-08-07 23:30:06,607 - DaemonLog - INFO -  0.0359242414064 car_ngram__mitral ftl1h58wgqx0payre240v5yd4
2016-08-07 23:30:06,607 - DaemonLog - INFO -  0.0333392482058 car_ngram__l5jn8k4322mj78l5jiv2mtvq1 go2o6sd8cepbjqv174ergfcw9
```

Figure 5-7: An example output of running FHTL on the partially decrypted patient data.

methods, like those described in section 5.3, were used to in conjunction with bag of words to filter out irrelevant data and extract certain feature vectors.

Most importantly, the partially encrypted bag-of-words output can usually remain encrypted as the underlying plaintext is not needed for the correctness of the algorithm. When FHTL is executed on a set of data it's goal is to classify each record in some category. In the case of the research study, the goal is to decide if the patient will have a successful CRT treatment or not. The bag-of-words output is shown in figure 5-7 is to highlight the FHTL algorithm traffics in partially encrypted data.

For reference, figure 5-8 shows the result of running FHTL on the plaintext patient data. These results match those of figure 5-7.

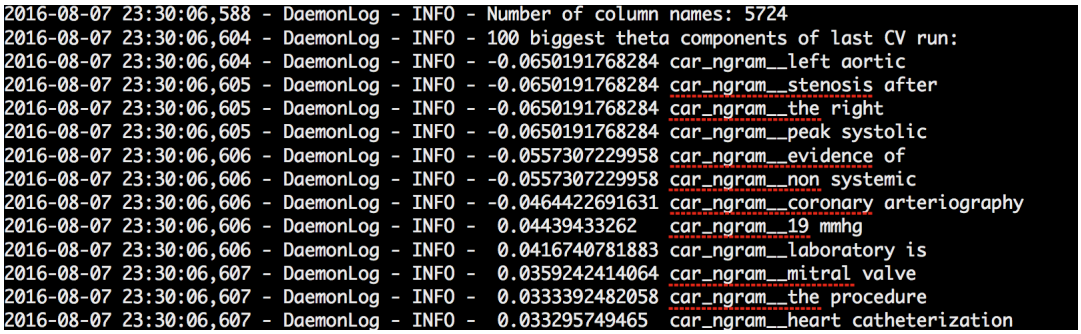

Figure 5-8: The execution of FHTL on unencrypted data.

However, in some cases the bag-of-words output can be useful while an NLP algorithm is being developed or modified. Therefore, it is possible that the machine learning research might occasionally desire to ask the hospital administrator to reveal the mapping of certain placeholders to corresponding plaintexts.

5.5.7 Auxiliary Information: Uncovering Deterministic Ciphertexts for BOW Output

The excerpt output shown in figure 5-7 is a bag of words (bi-gram, in this case) output, showing the most frequently occurring pairs of words. While some of these words are decrypted (via keyword keys), many others are unknown to the learner. That is, the output is partially encrypted.

From an honest execution of a well constructed machine learning algorithm like FHTL, most bag of words outputs will not contain PII, as PII will likely not repeat across patient files. Thus, it might be the case that the researcher should be able to uncover these unknown words, as long as the hospital administrator approves.

To this end, the researcher can copy the output of FHTL to a file `output.txt` and send it to the hospital. In turn, a hospital administrator can use algorithm 9) to uncover the underlying plaintext words. After manually inspecting the short list of uncovered plaintext bi-gram words, the hospital can decide if the uncovered words contain PII and if they do not, return the plaintext back to the researcher.

Chapter 6

Conclusion

In this work we designed and implemented a system where one data owning party can outsource keyword searches, approximate regular expression matching, and bag-of-words computations on encrypted free text data to a group of computationally more powerful, partially untrusted data learning parties. The first phase of our system enables the data owner to encrypt their data one time. The auxiliary key request phase allows data learning parties to request specific auxiliary keys and the compute phase uses these keys to perform basic NLP techniques like keyword search, approximate regular expression matching, and bag of words computations on encrypted free text data.

We provided formal privacy definitions and constructions for our schemes, and proved that our constructions achieve our privacy definitions using simple cryptographic primitives.

We implemented our schemes to provide both a software library to be integrated into existing applications, and Alvis, a command line interface that is ready to be deployed in real systems. We also created a searchable interface for data learners to integrate their custom data structures, enabling search on arbitrarily encrypted free text data structures.

Finally, we presented our work on a real-world case study, the data analysis of encrypted patient data. We used a pre-existing suite NLP algorithms that already proved to be fairly effective searching on patients with records of Cardiac Resynchronization Therapy. Without modifying the underlying algorithms, we were able to use Alvis to compute these NLP algorithms on encrypted free text patient data.

6.1 Code

All of our code, both the software library and the command line interface, is available for review at `https://github.mit.edu/agrinman/alvis`.

6.2 Future Work

While our work exhibits a fully-operable implementation that can be deployed today, more work is required to improve the usability and usefulness of the system.

6.2.1 Graphical User Interface

An important step is to add a graphical user interface (GUI) to act as substitute to the command line interface. While it is likely that most data learning parties are technical, hospital administrators will need a friendly GUI to select patient data files for encryption, approve requested keywords and frequency keys, and "uncover" ciphertexts like the special case in section 5.5.7.

6.2.2 Pilot for Computing on Encrypted Patient Data

A necessary next step is to introduce our system to both hospitals and machine learning researchers through a collaborative pilot. This pilot will be helpful to further validate our two party model, confirm that the interaction in our system is manageable, and that the search results prove to be useful.

While our implementation makes almost no assumptions about the data and operates on the very simplest input-output interface using files, it is likely that hospitals will need to integrate this tool as some pre/post-processing step deeper in their data management systems and a pilot would reveal the necessary next steps for a complete integration.

6.2.3 Testing on More Machine Learning and NLP Algorithms

Finally, our case study and primary motivation throughout this work has been searching on CRT-related patient data using the FHTL machine learning algorithm. However, our implementation is not specific to this study and therefore could be used with almost any computational study on patient data. Therefore, our next steps include finding more machine learning algorithms in the medical space to test alongside our implementation.

To understand where else our system could be practical and deployable, we need to look at other case studies in different fields. To this end, we also plan to look at financial algorithms to understand if there are certain companies like banks that need to outsource computing to collaborative, but not fully trusted, machine learning researchers.

Appendix A

Software Dependencies

A.1 Internal Golang Dependencies

bytes
crypto/aes
crypto/cipher
crypto/rand
crypto/sha256
encoding/json
errors
fmt
io/ioutil
log
math/big
net/http
net/http/pprof
os
path
runtime
strings
sync
unicode

A.2 External Golang Dependencies

https://github.com/fatih/color
https://github.com/urfave/cli

References

[1] Crypto - the go programming language. https://golang.org/pkg/crypto/.

[2] Documentation - the go programming language. https://golang.org/doc/.

[3] Ec2 instance types amazon web services (aws). https://aws.amazon.com/ec2/instance-types/.

[4] Fips 197, advanced encryption standard (aes). http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[5] Json. http://www.json.org/.

[6] A new secure hash standard - schneier on security. https://www.schneier.com/blog/archives/2007/02/a_new_secure_ha.html.

[7] The rijndael block cipher. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[9] Mihir Bellare and Phillip Rogaway. Introduction to modern cryptography. *Ucsd Cse*, 207:207, 2005.

[10] Alexandra Boldyreva, Serge Fehr, and Adam ONeill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *Annual International Cryptology Conference*, pages 335–359. Springer, 2008.

[11] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 506–522. Springer, 2004.

[12] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual International Cryptology Conference*, pages 213–229. Springer, 2001.

[13] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[14] Neal A Chatterjee and Jagmeet P Singh. Cardiac resynchronization therapy: past, present, and future. *Heart failure clinics*, 11(2):287–303, 2015.

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

[16] Austin Freel, Josh Haimson, Michael Traub, and Charlotta Lindvall. Predicting the effectiveness of cardiac resynchronization therapy using natural language processing.

[17] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[18] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 555–564. ACM, 2013.

[19] Ashish K Jha, Catherine M DesRoches, Eric G Campbell, Karen Donelan, Sowmya R Rao, Timothy G Ferris, Alexandra Shields, Sara Rosenbaum, and David Blumenthal. Use of electronic health records in us hospitals. *New England Journal of Medicine*, 360(16):1628–1638, 2009.

[20] Omkant Pandey and Yannis Rouselakis. Property preserving symmetric encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 375–391. Springer, 2012.

[21] FIPS PUB. Secure hash standard (shs). 2012.

[22] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 47–53. Springer, 1984.

[23] Brent Waters. Functional encryption for regular languages. In *Advances in Cryptology–CRYPTO 2012*, pages 218–235. Springer, 2012.