9-2018

Private-Key Fully Homomorphic Encryption for Private Classification of Medical Data

Alexander N. Wood
The Graduate Center, City University of New York

PRIVATE-KEY FULLY HOMOMORPHIC ENCRYPTION FOR PRIVATE CLASSIFICATION OF MEDICAL DATA

by

ALEXANDER NICOLAS WOOD

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

2018

Private-Key Fully Homomorphic Encryption for Private Classification of Medical Data

by

Alexander Nicolas Wood

This manuscript has been read and accepted by the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Date

Delaram Kahrobaei
Chair of Examining Committee

Date

Robert M. Haralick
Executive Officer

Supervisory Committee:
Robert M. Haralick
Delaram Kahrobaei
Ali Mostashari
Kayvan Najarian
Vladimir Shpilrain

THE CITY UNIVERSITY OF NEW YORK

ABSTRACT

Private-Key Fully Homomorphic Encryption for Private Classification of Medical Data

by

Alexander Nicolas Wood

Advisor: Professor Delaram Kahrobaei

A wealth of medical data is inaccessible to researchers and clinicians due to privacy restrictions such as HIPAA. Clinicians would benefit from access to predictive models for diagnosis, such as classification of tumors as malignant or benign, without compromising patients' privacy. In addition, the medical institutions and companies who own these medical information systems wish to keep their models private when used by outside parties.

Fully homomorphic encryption (FHE) enables practical polynomial computation over encrypted data. This dissertation begins with coverage of speed and security improvements to existing private-key fully homomorphic encryption methods. Next this dissertation presents a protocol for third-party private search using private-key FHE. Finally, fully homomorphic protocols for polynomial machine learning algorithms are presented using privacy-preserving Naive Bayes and Decision Tree classifiers. These protocols allow clients to privately classify their data points without direct access to the learned model. Experiments using these classifiers are run using publicly available medical data sets.

These protocols are applied to the task of privacy-preserving classification of real-world medical data. Results show that private-key fully homomorphic encryption is able to provide fast and accurate results for privacy-preserving medical classification.

Acknowledgments

This work would not have been possible without the financial support of the Office of Naval Research, the CUNY Computer Science Fellowship, the CUNY Mathematics Fellowship, the CUNY Summer Fellowship, and the Hunter College University Fellowship. I am indebted to Dr. Delaram Kahrobaei, Professor of Computer Science at The Graduate Center, CUNY, and the chairperson of my committee. Her guidance and expertise were fundamental to my success and taught me the importance of bridging theory and practice. I am forever grateful for the research opportunities she offered and the mentoring she provided.

Furthermore, I would like to thank each of the members of the committee for their invaluable feedback and unwavering support. Professor Kayvan Najarian has been a dedicated mentor throughout the research process. I am indebted to him for his confidence in my ability as a researcher as well as for hosting me at the University of Michigan. I am grateful and indebted to Professor Vladimir Shpilrain for generously sharing his mathematical insights and expertise. Furthermore, I would like to thank Professor Robert Haralick for providing crucial feedback and comments, which led to a great deal of improvements in my final dissertation. I am grateful for the support of Dr. Ali Mostashari, an industry collaborator and committee member whose generous advice helped shape my research.

Moreover, I would like to thank Jonathan Gryak for his advice and support through the years and his generous feedback which led to improvements in my dissertation. Last but not least, I would like to thank my family and friends who have offered invaluable support during

pursuit of this project. I would like to thank Professor Katia Perea, whose kind and patient friendship made this work possible. Finally, I would like to thank George and Roberta Tabb for always believing in me, for their unconditional love and support, and for their assistance in proofing and editing.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The fields of machine learning and cryptography rose to prominence and experienced rapid development in the past few decades. Cryptography, in particular, revolutionized the world, from Whitfield Diffie and Martin Hellman's asymmetric encryption scheme in 1976 to the current prevalence of online shopping. Machine learning was spearheaded in the 50s with Alan Turing's "Turing Test," Arthur Samuel's checkers program, and Frank Rosenblatt's perceptron. Now, machine learning permeates our lives via automatic online recommendations, self-driving cars, and more. Machine learning techniques applied to medical data have led to great leaps forward in the medical field, with applications in personalized treatment, disease diagnosis, radiology, and more [13, 24].

Despite this the two sub disciplines have remained relatively separate. As postulated by Rivest, the fields of cryptography and machine learning at first appear to be opposites; cryptography, on the one hand, seeks to *hide* information, while machine learning looks to *discover* information [35, 58]. However antithetical it may appear at first, machine learning and cryptography are bound to be intertwined. The more easily data is available, the greater the need for privacy, and data is gathered faster than ever before.

The need for privacy has a direct application to medical data. Information storage costs continue to decrease while personal medical applications, such as genome sequencing, are increasingly accessible. Websites such as 23andme, LifeNome, and Ancestry.com provide the first wave of commercially available personalized genetic analysis. A breach in the privacy of this data could leave a subject particularly vulnerable, as we are uniquely identified by our genetic code. As technology advances so do the insights gained from genetic analysis, and as time goes on the risk involved with sharing your genetic data could increase. The implications of this data being shared in a publicly identifiable way could have an unforeseen negative impact upon a person's life. Strict privacy guidelines should be applied to genetic information.

Furthermore, considered collectively these patients' records represent a wealth of data that has already been collected by various research institutions and hospitals. The ability to use this information without compromising the privacy of these patients would impact the field of computational medicine. In particular, training classification models on a single-source data set can lead to over-fitting. This yields a learned model with excellent results on the testing data but unpredictable results when applied to new data [51]. Medical researchers could use private classification methods to verify that their model can generalize to an external database.

In addition, privacy is a growing concern in medical applications as more assisted decision making and diagnosis systems become commercially available. Owners of these systems do not wish to share their models. Similarly, hospitals and clinicians are unwilling to share their patients' data due to privacy restrictions such as The Health Insurance Portability and Accountability Act of 1996 (HIPAA) [47]. Clinicians would benefit from access to private classification protocols which allow them to access these diagnosis systems without having to reveal patient information. Thus, there are two opposing forces at work: the desire to analyze all available data in order to increase knowledge and sophistication of machine learning techniques, as well as the need for privacy and control over what information we share.

The field of fully-homomorphic encryption (FHE) seeks to bridge this gap. Theoretically, a fully homomorphic encryption scheme allows for computation of arbitrary functions over encrypted data without first performing decryption. Current research into encrypted computation over medical data employs fully homomorphic public-key cryptosystems, which enable secure communication between multiple parties over insecure channels via asymmetric key distribution. Private-key cryptosystems require prior knowledge of the encryption/decryption key(s). In other words, if multiple parties wish to perform decryption in a private-key setting, they must first exchange keys over a secure channel. While this is considered a disadvantage of private-key cryptosystems when the goal is purely communication, these cryptosystems are suitable for medical applications. Due to HIPAA restraints and the personal nature of genomic and medical data, it makes sense that those who hold medical data would not, in fact, want anyone besides themselves to have the option of encrypting the data [36].

1.1 Contribution

This dissertation addresses the use of private-key fully homomorphic encryption for design of efficient private classification algorithms in medical applications. Security of these classification algorithms, simply put, corresponds a two-party protocol between a Client and a Model Owner. The Client should learn no unnecessary information at the end of the protocol about the model owned by the Model Owner, and the Model Owner should learn no information about the Client's input. "Unnecessary information" is a vague term which is clarified in the formal security discussions. Put abstractly, this qualification references the fact that privacy-preserving classification necessitates the sharing of *some* information about the Model Owner's model – for instance, the final classification of the Client's data within that model. Beyond their classification, Clients should learn no unnecessary information about a Model Owner's model. Some privacy-preserving classification protocols may take place with an intermediary Server between the Client and Model Owner. In this case, the Server should only perform computation, and should learn no unnecessary information about the Client's data point(s) or the Model Owner's model(s).

The research in this dissertation provides privacy-preserving classification algorithms using private-key fully homomorphic encryption for Naive Bayes and decision tree classifiers. Implementation of these algorithms requires the construction of additional protocols. In particular, this dissertation presents algorithms for the private computation of the argmax function, a third-party private search protocol, and algorithms for efficiently implementing private-key FHE.

Experimental results on real-world medical data sets show that these classifiers are able to provide fast and accurate classification results. Specifically, information gathered from breast tumor biopsies [45] is classified as malignant or benign using the proposed privacy-preserving protocols. Experimental results on the efficiency third party private search protocol are also presented.

The organization of this dissertation is as follows. Terminology is defined in Chapter 2. Chapter 3 provides an overview of the history of fully homomorphic encryption and privacy-preserving classification as well as current state-of-the-art techniques. Chapter 4 proposes methodology for implementation of private-key fully homomorphic encryption, such as encoding methods and parallelization techniques. Chapter 5 presents a privacy-preserving Naive Bayes protocol using private-key FHE as well as experimental results. Chapter 6 presents a third-party private search protocol, and Chapter 7 presents a privacy-preserving decision tree classifier as well as experimental results. Chapter 8 concludes the thesis by summarizing the contributions.

Chapter 2

Terminology

2.1 Machine Learning

Machine learning broadly seeks to learn new information from a given data set [39]. A data set has some *features* which are used to use to predict some *quantitative* or *categorical outcomes*. A *supervised learning problem* in machine learning operates by using a set of *training data* to draw conclusions about the relationship between features of the data and outcomes. These conclusions are used to create a *learner*, which predicts the outcome of any new data points.

When selecting parameters for a supervised learning problem, it is important to keep in mind the *bias-variance tradeoff*. While the goal of machine learning is to model the specifics of the training data in a method that generalizes well to other data, it is often not possible to do both simultaneously. As the complexity of the model is decreased, the variance tends to decrease while the bias increases. This means that the learned model may be simpler and will not overfit on new data, but it will underfit on the training data. On the other hand, a more complicated model tends to have increased variance with decreased bias. This means that there is a higher risk of overfitting the model on training data. Thus, with any supervised learning problem it is important to seek a balance between bias and variance.

Classification is the task of predicting a qualitative output value, often called a *class*, from a set of input values. These input values, the dependent variables, may be *discrete* or *continuous*. A discrete variable takes its value from a discrete, or countable, set. A continuous variable can take on an infinite number of possible values [39]. In computational medicine we often deal with discrete class values. A common application in computational medicine is *binary* classification, where there two categories in which the data points reside, represented as $\{0, 1\}$ or $\{-1, 1\}$, often called *targets*. Binary classification is used for disease prediction, for example, "has cancer" or "does not have cancer."

Feature selection is a pre-processing method which identifies the most relevant features of a data set. By restricting learning to these useful features and eliminating irrelevant or redundant features, classification models' performance may be increased [37]. Because implementation of fully homomorphic encryption and other privacy-preserving measures can lead to a large increase in classification time, it is important that learning is carried out on only this relevant data.

2.1.1 Performance Measures

The performance of binary classification algorithms is evaluating using a variety of performance measures. Let TP, TN, FP, and FN denote True Positive, True Negative, False Positive, and False Negative, respectively. Let P and N denote the total number of positive and negative data points in the testing set. One measure called *accuracy* is calculated as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$

and yields the proportion of data points that were correctly classified.

Other performance measures include *sensitivity*, *precision*, *specificity*, and *negative pre-*

dictive value (NPV). The first two are given by

$$\begin{aligned}\text{Sensitivity} &= \frac{\text{TP}}{\text{TP}+\text{FN}}\\ \text{Precision} &= \frac{\text{TP}}{\text{TP}+\text{FP}}.\end{aligned}$$

Sensitivity provides a measure of what proportion of positive cases were classified correctly as positive, while precision provides a measure of what proportion of cases classified as positive were positive in reality. Sensitivity is especially important in medical applications as it is critical to correctly identify all true positive cases [52].

Specificity and NPV are also known as inverse recall and inverse precision, as they provide similar information for negative classifications. Specificity describes the proportion of negative cases that were classified as negative and NPV describes the proportion of cases classified as negative that were negative in reality. They are computed as

$$\begin{aligned}\text{Specificity} &= \frac{\text{TN}}{\text{TN}+\text{FP}}\\ \text{NPV} &= \frac{\text{TN}}{\text{TN}+\text{FN}}.\end{aligned}$$

The *F1 score* provides the harmonic mean of precision and sensitivity and is given by

$$\text{F1} = \frac{2\cdot\text{TP}}{2\cdot\text{TP}+\text{FP}+\text{FN}}.$$

Other performance measures occasionally seen include the *false positive rate (FPR)*, also known as *fallout*, and the *false negative rate (FNR)*. These measures are calculated as

$$\begin{aligned}\text{Fallout} &= \frac{\text{FP}}{\text{FP}+\text{TN}}\\ \text{FNR} &= \frac{\text{FN}}{\text{TP}+\text{FN}}.\end{aligned}$$

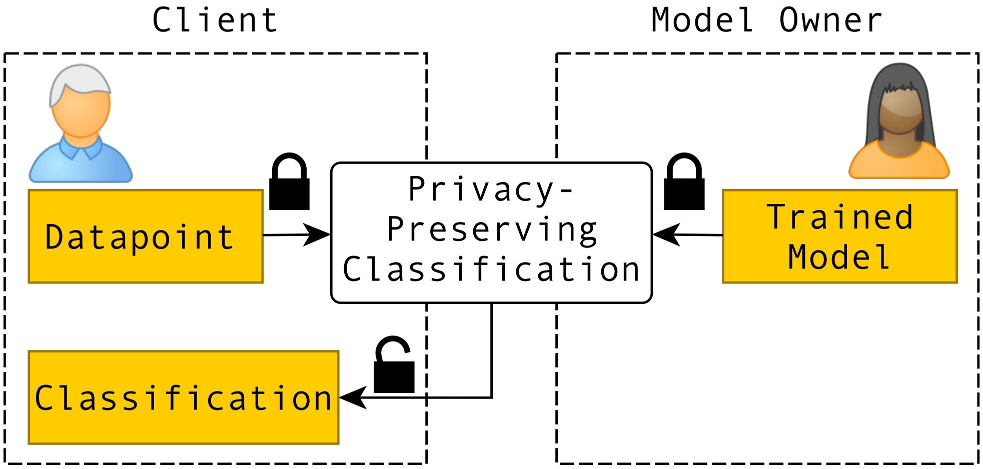


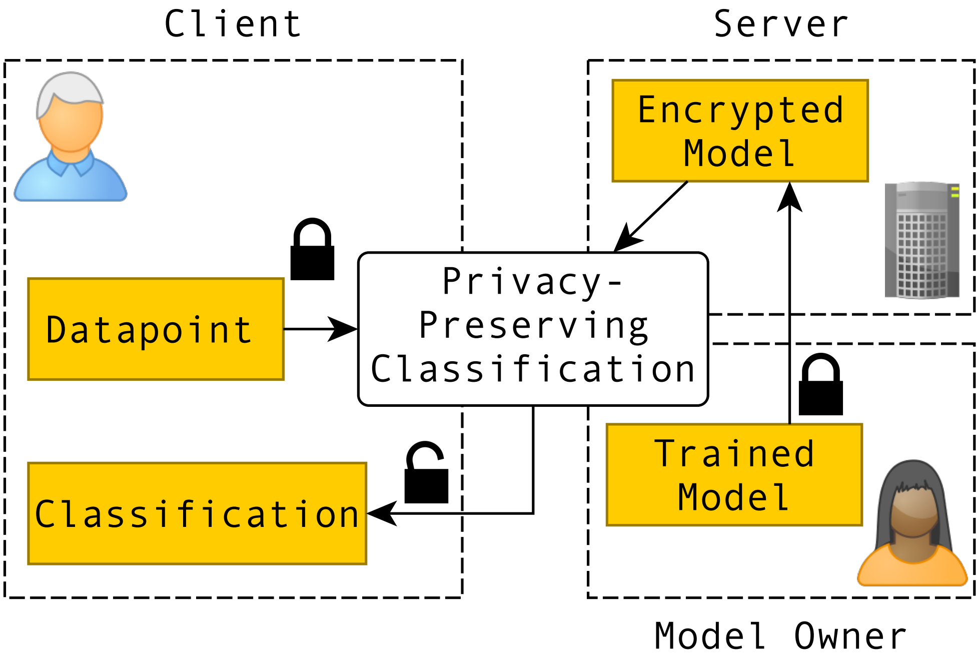
Figure 2.1: PPC Between the Client and Model Owner

These calculate the proportion of negative values which are wrongly classified as positives and the proportion of positive values which are wrongly classified as negatives.

2.2 Privacy-Preserving Classification

We wish to specifically look at machine learning applications in the medical field. Privacy restrictions such as the Health Insurance Portability and Accountability Act (HIPAA) necessitate the development of private classification algorithms. *Privacy-preserving classification* describes the collection of efficient methods for privately performing the classification stage of a machine learning algorithm [6], while *privacy-preserving data-mining* describes the task of training a model entirely over encrypted data [1]. Privacy-preserving classification is the focus of this work. Introductory background on privacy-preserving data-mining is available in the literature review in Chapter 3.

During privacy-preserving classification, Clients classifies their data vector using a model owned by the Model Owner. Each party would like to keep their information private – Clients do not want the Model Owner to learn any partial information about their data vectors, and



t

Figure 2.2: PPC with Computation on Server

the Data Owner does not want the Clients to learn any unnecessary information about her model.

For example, the Model Owner could be an institution that used its own collected data to create a model. The Client could then be clinicians who use the privacy-preserving classification in order to assist with the treatment of their patients. The Model Owner may or may not wish to delegate computation to a cloud service provider, which we will call the Server. Figure 2.1 shows the outline of the protocol carried out between the Client and the Model Owner. Figure 2.2 shows the protocol as carried out between the Client, Model Owner, and an intermediary Server.

2.2.1 Model

The discussion that follows is based off of the notation and presentations given by Hastie [39] and Bost et al. [6]. Throughout this paper notation is as follows: The Client has data in

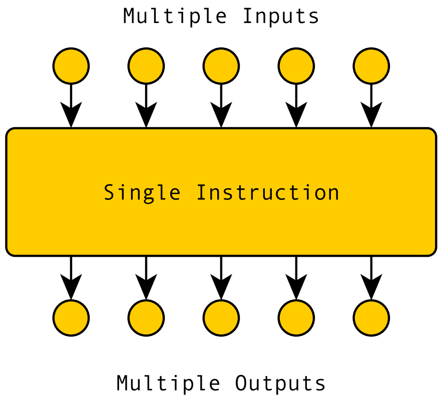


Figure 2.3: The SIMD Paradigm

the form of d-dimensional vectors $X = (X_1, X_2, \ldots, X_d)$. X is called a *feature vector*, while each X_i in X is called a *feature*.

The Client wishes to classify his data using a *classification function* f, known only by the *Model Owner*, into a set of discrete classes $\mathcal{G}$. Observed classes will be denoted G whereas predicted classes denoted with a hat, $\hat{G}$. A large amount of *training data* is used to construct a classifier, say N inputs, each written as a feature vector-class pair (X, G_i).

2.3 Parallelization via SIMD

The *Single-Instruction Multiple-Data (SIMD)* paradigm is a class of parallel computers. SIMD allows for computation of multiple values under a single instruction. Figure 2.2 shows the general concept, where multiple inputs are encoded within a single vector. A single instruction operates on this vector, and decoding the output vector yields the output of the instruction on each of the individual inputs. A common example is that of image processing, where a filter is applied to every pixel in an image [18].

2.4 Fully Homomorphic Encryption

One major approach to the task of privacy-preserving classification uses *fully homomorphic encryption* (FHE). Homomorphic encryption was first conceptualized in 1978 by Rivest, Adleman, and Dertouzos, envisioned originally as a 'privacy transformation,' which would enable computation of functions on encrypted data without first having to decrypt the information [57]. This concept is known today as a homomorphic encryption scheme, informally defined as allowing for computation of functions over encrypted data. A homomorphic encryption scheme is called *fully* homomorphic if it allows for computation of *arbitrary* functions over encrypted data.

The algorithms that comprise a public key homomorphic encryption scheme $\mathcal{E}$ are defined as follows:

I. $(\mathrm{pk}, \mathrm{sk}) = \texttt{KeyGen}_{\mathcal{E}}(n)$, the key generation algorithm, which distributes public and private keys pk and sk (respectively) to all necessary parties given some security parameter n.

II. $c = \texttt{Encrypt}_{\mathcal{E}}(m, \mathrm{pk})$, the encryption algorithm, which takes as input pk as well as a message m. The output is a ciphertext c.

III. $m = \texttt{Decrypt}_{\mathcal{E}}(c, \mathrm{sk})$, the decryption algorithm, which uses the private key(s) sk to recover the plaintext m given a ciphertext c.

As $\mathcal{E}$ is a homomorphic public key encryption system, it is able to carry out computations over some set of circuits $\mathcal{C}$ by utilizing an additional algorithm:

IV. $c = \texttt{Evaluate}_{\mathcal{E}}(c_1, c_2, \ldots, c_n, C, \mathrm{pk})$, an algorithm to perform computation over encrypted data. The input to this function is a collection of ciphertexts $c_1, c_2, \ldots, c_t$, a circuit $C \in \mathcal{C}$, and the public key(s) from the key generation algorithm.

Formally, the encryption scheme $\mathcal{E}$ is called homomorphic on $\mathcal{C}$ if it is correct on $\mathcal{C}$ and the decryption algorithm can be expressed as a circuit of size poly(n) [27]. The scheme is called *fully homomorphic* if $\mathcal{C}$ is the set of arbitrary circuits. Because a Boolean circuit can describe arbitrary computations, a scheme only needs to be homomorphic over addition and multiplication to be described as fully homomorphic [50]. A scheme $\mathcal{E}$ is called additively homomorphic if

$$\texttt{Encrypt}_{\mathcal{E}}(x+y) = \texttt{Encrypt}_{\mathcal{E}}(x) \oplus \texttt{Encrypt}_{\mathcal{E}}(y)$$

for some operation $\oplus$ in the ciphertext space. Similarly, a scheme is called multiplicatively homomorphic if

$$\texttt{Encrypt}_{\mathcal{E}}(x \cdot y) = \texttt{Encrypt}_{\mathcal{E}}(x) \otimes \texttt{Encrypt}_{\mathcal{E}}(y)$$

for an operation $\otimes$ in the ciphertext space. This functionality is shown in Figure 2.4. Two plaintexts that are first encrypted, added (or multiplied), and then decrypted, yield the same result as adding (or multiplying) over the original plaintexts.

While computation of arbitrary functions is *theoretically* possible using addition and multiplication as described above, this in itself is not sufficient for *practical* computation of arbitrary functions. Any fully homomorphic encryption scheme that is suited for practical use will only be able to compute polynomial functions and polynomial approximations of functions, known as *polynomial machine learning* [35].

2.4.1 Private-Key Fully Homomorphic Encryption

The majority of previous work focuses on public-key fully homomorphic encryption. This work implements *private-key* fully homomorphic encryption. A private-key cryptosystem generates only one key, the secret key, during the `KeyGen` algorithm. This private key is used for both encryption of plaintexts and decryption of ciphertexts.

A private key homomorphic encryption scheme $\mathcal{E}$ over a set of circuits $\mathcal{C}$ is defined via

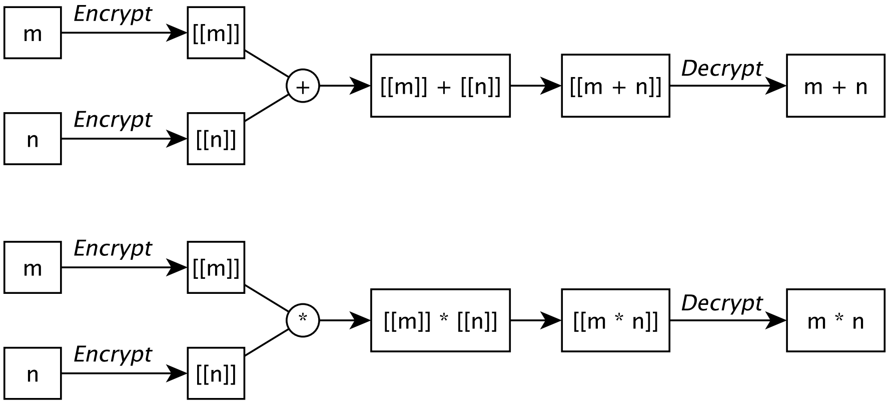


Figure 2.4: Homomorphic addition (top) and multiplication (bottom)

the following algorithms:

I. $\text{sk} = \texttt{KeyGen}_{\mathcal{E}}(n)$, the key generation algorithm, which distributes a private key sk to the necessary party given some security parameter n.

II. $c = \texttt{Encrypt}_{\mathcal{E}}(m, \text{sk})$, the encryption algorithm, which takes as input sk as well as a message m. The output is a ciphertext c.

III. $m = \texttt{Decrypt}_{\mathcal{E}}(c, \text{sk})$, the decryption algorithm, which uses the private key sk to recover the plaintext m given a ciphertext c.

IV. $c = \texttt{Evaluate}_{\mathcal{E}}(c_1, c_2, \ldots, c_n, C)$, an algorithm to perform computation over encrypted data. The input to this function is a collection of ciphertexts $c_1, c_2, \ldots, c_t$ and a circuit $C \in \mathcal{C}$.

Encryption and decryption can only be carried out by keyholder(s), while evaluation can be performed by any party possessing a ciphertext. As before, a private-key homomorphic encryption scheme is called fully homomorphic if $\mathcal{C}$ is the set of arbitrary circuits, and arbitrary computation can be reduced via Boolean circuits to homomorphic addition and multiplication.

2.4.2 Leveled, Somewhat, and Partially Homomorphic Encryption

A scheme that is homomorphic over one operation is called *partially* homomorphic. A partially homomorphic scheme can be additively homomorphic or multiplicatively homomorphic. Partially homomorphic encryption schemes have existed for some time, such as the multiplicatively homomorphic ElGamal scheme [23] and the additively homomorphic Pallier scheme [48].

Schemes that can perform homomorphic addition and multiplication over encrypted data up to some computational limit are called *somewhat homomorphic* encryption (SHE) schemes. These schemes contain *noise* terms, which grow exponentially during homomorphic addition and multiplication operations, and correct decryption is not possible once these noise terms exceed a noise bound. The first FHE schemes were constructed from SHE schemes using a technique called *bootstrapping*, which manages this noise, discussed in detail in Chapter 3.

A *leveled* homomorphic encryption (LHE) scheme is a scheme in which noise growth is polynomial in the homomorphic multiplication operation. Therefore, these schemes can be implemented to perform FHE up to some pre-specified depth without bootstrapping. Specific LHE schemes are discussed in Chapter 3.

2.4.3 Notation

For the remainder of this work let $[\![x]\!]_{\mathcal{S}}$ denote the encryption of a plaintext x within an encryption scheme $\mathcal{S}$. For brevity of notation, the subscript is omitted for values encrypted under the Gribov-Kahrobaei-Shpilrain (GKS) scheme. The fully homomorphic addition and multiplication operations are denoted by $[\![x]\!] + [\![y]\!] = [\![x \oplus y]\!]$ and $[\![x]\!] \cdot [\![y]\!] = [\![x \otimes y]\!]$, respectively. Let $a \overset{\$}{\leftarrow} A$ denote the selection of a value a from a set a uniformly at random.

Chapter 3

Background

This chapter provides an overview of current methods in the field of privacy-preserving classification, machine learning, and computational medicine. These methods include differential privacy, fully homomorphic encryption, and various non-fully homomorphic encryption methods.

Leveled homomorphic encryption schemes have found some success with classification algorithms. The Yet Another Somewhat Homomorphic Encryption scheme (YASHE) was used in an application of neural networks to encrypted data called CryptoNets [33], and has been implemented in the Simple Encrypted Arithmetic Library (SEAL) as well with bioinformatics computation in mind [21]. ML Confidential used leveled homomorphic encryption (LHE) to run classification using Linear Means and Fisher's Linear Discriminant classifiers [35].

Some private classification methods do not implement homomorphic encryption. Differential privacy is one non-cryptographic approach that has been implemented, although it lacks the utility of FHE schemes [33]. Non-fully homomorphic cryptographic methods have been used to apply Naive Bayes and decision tree classification [5]. Vaidya et al. describe protocols for constructing support vector machine (SVM) models using horizontally, vertically, or arbitrarily partitioned data while maintaining the privacy of their data without FHE [63].

This chapter begins with discussion of differential privacy and privacy-preserving classification techniques. The chapter concludes with an overview of the history of public-key and private-key fully homomorphic encryption methods as well as the current techniques.

3.1 Differential Privacy

Differential privacy is a non-cryptographic approach to private data mining that has seen some success with training various classification algorithms [22]. Differential privacy is a method of security that applies to databases. Intuitively, if there are two databases that differ on only one row, a query satisfies differential privacy if there is a very high probability that the query will produce the same result regardless of which database is queried. This method's security depends upon having a large database. This method loses its utility when the goal is to classify a single data point [33].

One example of differential privacy for data analysis is given by Wang, Mohammed, and Chen, who present a method for distributing genetic data that satisfies differential privacy [66]. The authors provide the following formal definition of differential privacy.

Definition 3.1.1. *Let $\mathcal{A}$ be a randomized algorithm and let D and D' be two databases such that*

$$|D \Delta D'| \leq 1.$$

In other words, there is at most one row in D' that does not appear in D, or vice versa. Say that $\mathcal{A}$ is ϵ-differentially private *if for all possible anonymized data sets $\hat{D}$,*

$$\Pr[\mathcal{A}(D) = \hat{D}] \leq re^{\epsilon} \times \Pr[\mathcal{A}(D') = \hat{D}].$$

In sum, one effectively cannot tell from looking at the output of $\mathcal{A}$ on D and on D' whether or not one specific line of data was included in the data set. This is often achieved by adding in random noise. The results of a query are disguised by giving an answer that is not quite exact, but is "close enough" for the intended analyses.

The goal of differential privacy, while similar to that of encryption, is different in key ways. The goal of encrypting information is to hide it completely. With differential privacy, the goal is not to hide data but rather to anonymize it. Furthermore, in the setting explored in this work, the goal is to hide not only the database from the user, but also to hide the query from the database holder. Differential privacy only seeks to perform the former function.

3.2 Privacy-preserving Classification

3.2.1 ML Confidential

Graepel, Lauter, and Naehrig suggest a framework for private training and classfication via machine learning that they title ML Confidential [35]. This protocol is proposed for both the training and classification phases of machine learning and is carried out entirely over encrypted data using an LHE or SHE scheme. The authors describe a protocol that operates over a class of machine learning algorithms that they designate polynomial learning. A proof-of-concept is given for the classification phase of machine learning.

ML Confidential operates between three parties. There is the *Data Owner*, who holds the data to be processed, as well as the *Content Provider*, which uploads data to the *Cloud Service Provider* on the Data Owner's behalf. The Data Owner wishes to perform both the training phase, `ML.Train`, and classification phase, `ML.Classify`, of some machine learning algorithm on the cloud without giving the cloud access to their data. The protocol may be either private key or public key. Furthermore, the homomorphic encryption scheme used may be either fully homomorphic, somewhat homomorphic, or a leveled homomorphic encryption scheme. The algorithms provided by the protocol are

- `HE.Keygen` for generation of (public or private) keys,
- `HE.Enc` for (somewhat/leveled/fully) homomorphic encryption,
- `HE.Dec`, the corresponding decryption algorithm,
- `HE.Eval` for homomorphic computation and utilizes `HE.Add` for homomorphic addition and `HE.Mult` for homomorphic multiplication.

Note that the specifics regarding the functionality of these functions depend upon whether the scheme is private or public key as well as the method of homomorphic encryption utilized. In particular, the authors provide examples of a leveled homomorphic encryption scheme in which `HE.Eval` computes polynomial functions of a bounded degree.

ML Confidential Protocol, Private Key Version

- *Key Generation*: The Data Owner runs `HE.Keygen` to generate a private key sk, securely stored locally, and shares this key with the Content Provider.
- *Encryption & Upload, Training Data:* For all training vectors $\mathbf{x}$ the Content Provider sends `HE.Enc`$(sk, \mathbf{x})$ to the Cloud Service Provider.
- *Training:* The algorithm `HE.Eval` runs the training phase `ML.Train` on the encrypted training vectors. This computes an encrypted Learned Model that is stored by the Cloud and available to the Data Owner.
- *Classification:* Next, a previously unused vector $\mathbf{x}$ is encrypted and `HE.Enc`$(sk, \mathbf{x})$ is sent to the cloud, which carries out `ML.Classify` and returns the encrypted classification to the data owner.

- *Verification:* The Data Owner tests the encrypted Learned Model probabilistically by sending encryptions of test vectors to the Cloud and verifying that they are returned with the correct classifications.

For the public key version, only the first algorithm is significantly modified:

ML Confidential Protocol, Public Key Version

- *Key Generation*: The Data Owner runs `HE.Keygen` to generate a private key sk securely stored locally and a public key pk. It publishes the public key pk.

The algorithm `HE.Encrypt` uses the public key, pk, while `HE.Decrypt` can only be carried out by the Data Owner using the secret key, sk.

This protocol allows a diverse range of sources to provide data while all computation takes place on the cloud. Its security model is designed for a cloud that is honest-but-curious, meaning it will look at the available data but will not deviate from the set protocol. The authors point out this is a reasonable assumption for any commercial cloud service, as once the Cloud's reputation is damaged it will not be able to acquire new clients and hence it has strong motivation to behave honestly. The authors describe their verification step as a naive version of Proof-of-Storage protocols. The Data Owner must store and test enough samples to either determine the test error of the Cloud or determine the location of any accidental error, but has no reason to suspect that the Cloud is purposefully manipulating their data in any way.

Furthermore, the Cloud gains access to a certain amount of information during this protocol. The Cloud must learn the number of vectors trained upon, the number of vectors tested, the number of vectors in each class, and an upper bound on the number of entries in each class.

Next, the authors define a *polynomial learning algorithm* as follows:

Definition 3.2.1. *Let* $A : (\mathbb{R}^n \times \mathcal{Y})^m \times \mathbb{R}^n \to \mathcal{Y}$ *be a learning algorithm that takes a training sample* $(\mathbb{R} \times \mathcal{Y})^m$ *and a test input* $\mathbf{x} \in \mathbb{R}^n$ *and returns a prediction* $y \in \mathcal{Y}$*. Call the learning algorithm* D-polynomial *if the function* A *is polynomial of degree at most* D *in all of its arguments.*

The authors describe a leveled homomorphic encryption scheme based off of the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [11], discussed in detail in Section 3.5.2. Its security is based in the Ring Learning With Errors (RLWE) problem, which provides strong hardness guarantees [46]. The scheme involves a noise term that grows during homomorphic operations. This scheme can only compute D-polynomial functions. Any other function results in noise growth that obstructs decryption.

This scheme was used to perform binary classification with inputs in $\mathbb{R}^n$. The authors test linearizations of the linear means classifier and Fisher's linear discriminant classifier on publicly available breast cancer data using a public-key SHE scheme based on ring-LWE. Because the cryptosystem described is unable to perform any computations that are not D-polynomial this method is unable to perform many of the common machine learning algorithms, including perceptron, support vector machine, k-nearest neighbors, decision trees, exact logistic regression, and more.

3.2.2 The Simple Encrypted Arithmetic Library (SEAL)

The Simple Encrypted Arithmetic Library (SEAL) was developed by researchers in the Cryptography Research Group at Microsoft Research [21]. It is a homomorphic encryption library that was made specifically with Bioinformatics research in mind. SEAL uses LHE with parameters chosen to perform a predetermined number of computations. Initial implementations of SEAL used a variant on the YASHE scheme described in [5]. The most recent version [43] implements the "FullRNS" variant [4] of the Fan-Vercauteren somewhat homomorphic scheme [25].

The authors describe methods in which their software can be used for various biomedical applications. Possible applications mentioned are computing minor allele frequencies, χ^2-statistics, and tests for association of a genotype with disease, among others.

Because of the nature of leveled homomorphic encryption, it can reduce the cost of computation to compute an approximation in place of a more costly function. An example provided by the authors is that of logistic regression. The authors approximate the logistic regression function using polynomial approximations, and hence are able to use this approximation with SEAL.

3.2.3 SEAL for Classification via Neural Networks

Dowlin et al. provide a methodology they term *CryptoNets* in order to carry out private classification over neural networks. They use the YASHE leveled homomorphic encryption scheme to implement their protocol [5], implemented via SEAL [21].

A neural network consists of layers containing *nodes* that compute functions over values fed from the previous layer. These functions are often not polynomial functions, making the direct application of homomorphic encryption unfeasible. Therefore, the authors describe a class of polynomial functions that can be implemented at each layer during the classification stage instead.

Specifically, the authors include the *sigmoid* function in their network, which computes

$$z \mapsto \frac{1}{1 + \exp(-z)}$$

for a value z from a node in the previous layer. The sigmoid function is used in the final layer of the authors' training network. During the testing stage, this step was removed from the network altogether, as it is a monotone function and hence does not affect the final prediction once training is complete. The authors replace the more commonly used rectified linear activation function $z \mapsto \max\{0, z\}$ with the square activation function $z \mapsto z^2$ in both the training and testing stage.

Max pooling is another common layer seen in neural networks that computes the average value of a subset of components from the previous layer. The authors replace the max pooling layer in the classification stage with a polynomial approximation by simply taking the sum of the components instead of the average. As a result, the output in this layer is scaled by some factor and this scaling propagates to subsequent layers. The authors use this *scaled mean-pool* function in their network in both the training and testing stages instead of max pooling layers.

The authors train a sample network on images of $60,000$ handwritten digits then test on the $10,000$ remaining images. They achieve an accuracy rate of 99%.

3.2.4 Other Cryptographic Methods

Bost et. al. construct protocols for privacy-preserving classification using hyperplane detection, Naive Bayes, and decision trees [6]. The protocols are constructed using two additively homomorphic encryption schemes and one leveled homomorphic encryption scheme, HElib [59]. The additively homomorphic schemes are Goldwasser and Micali's Quadratic Reciprocity (QR) cryptosystem [34], as well as the Paillier cryptosystem [48]. Let pk_Q, sk_Q denote a public and secret key pair in the QR cryptosystem and pk_P, sk_P denote the same in Paillier's system. Let square brackets $[\![a]\!]_{\mathcal{P}}$ and $[\![a]\!]_{QR}$ denote the encryption of a value a under Paillier and QR, respectively. The security of these cryptosystems provides semantic security for the authors' protocols, and an honest-but-curious adversary model is used.

This subsection discusses the privacy-preserving Naive Bayes protocol presented by the authors. In order to build their privacy-preserving classification protocol, the authors first describe efficient protocols to perform comparison and argmax operations over encrypted data. They compose these operations within their schemes to construct their encrypted machine learning protocols.

The protocols for comparison and argmax are designed by the authors to work for additively homomorphic public-key cryptosystems and do not apply directly to this paper. Therefore, this section discusses the authors' private Naive Bayes algorithm in full, and reserves discussion of comparison and argmax for Chapter 5.

Comparison

Consider two data holders, A and B, who wish to compare their data, as well as the classifiers, a client C and server S. The authors describe several separate cases in that a comparison protocol can be carried out.

In the first scenario, A and B wish to privately compare their values a and b, respectively. The user B randomly selects a masking bit c and its sends its encryption under QR, $[\![c]\!]_{QR}$, to A. The authors construct a scheme that uses a garbled circuit combined with oblivious transfer to allow A to compute $(a < b) \oplus c$. Because A also knows the value $[\![c]\!]_{QR}$, A can use the additively homomorphic property of QR to compute $[\![a < b]\!]_{QR}$.

The next cases discussed by the authors involve comparison on encrypted inputs. User A has two encrypted inputs $[\![a]\!]$ and $[\![b]\!]$ that she would like to compare and user B has the decryption key. The users construct a method using a modification on the protocol designed by Veugen [64]. The last case of comparison the authors consider again has user A with two encrypted inputs. This one proceeds as the last case, but reversed – thus B is left with the result of the comparison while A is the one who holds the (encrypted) data.

argmax

The authors consider private computation of the argmax function, which returns the argument (position) of the maximum value in a vector. User A has access to values $[\![a_1]\!], \ldots, [\![a_k]\!]$, encrypted under user B's secret key. A wants B to learn the index of the largest of the k values without learning any other information, including other information on the ordering of the values.

User A begins by randomizing the order of the k elements using a random permutation π and sets the maximum index value equal to $a_{\pi(1)}$. The idea is to iterate through each value in the permutation, comparing $[\![a_{\pi(1)}]\!]$ to $[\![a_{\pi(2)}]\!]$ and setting the maximum value equal to the index of whichever is larger. Note that this is, in fact, a sequence of k comparisons.

The authors begin each comparison by running the previously described privacy-preserving comparison protocol. This alone is not sufficient for the privacy goal because user A will learn the ordering of the permuted inputs. Therefore, B must implement another randomization step, which the authors call `Refresh`. The `Refresh` procedure is carried out by the Paillier system's method for randomization of ciphertexts.

This is still not sufficient, because user B holds the secret key for the Paillier system. Therefore, instead of B performing `Refresh` on $[\![a_{\pi(i)}]\!]$, the authors use the additive homomorphic property of the Paillier scheme. User A sends B the value $[\![a'_i]\!]$, which is $[\![a_{\pi(i)}]\!]$ perturbed by A adding random noise. User B then performs the `Refresh` procedure on $[\![a'_i]\!]$.

Private Naive Bayes

The authors implement the above protocols to carry out private Naive Bayes classification. Let $\mathcal{G}$ denote the set of classes and assume there is a finite number of values each attribute can take. Let P denote the vector of class probabilities, where $P_i = \Pr(G_i)$, and let T denote the collection of tables given by $T_{i,j}(X) = \Pr(X = X_i | G = G_i)$. Assume each vector X has d features, and that there are c possible classes. The authors' private Naive Bayes protocol runs as follows.

1: The service provider encrypts the tables P and T using Paillier.

2: The server sends the encrypted tables to the client.

```
3: The client computes $[\![p_i]\!] = [\![P_i]\!] \prod_{j=1}^{d} [\![T_{i,j}(x_j)]\!]$ for $i = 1, \ldots, c$.
4: The client uses the server to compute $i = \text{argmax}_i p_i$.
5: Client outputs $i$.
```

Note that step three of the above protocol requires only that the encryption scheme be multiplicatively homomorphic. It is Step 4 that would ultimately require an additively homomorphic scheme. The authors in [6] use multiple encryption methods, combined with an algorithm for changing the encryption scheme, in order to compute the argmax.

Private Polynomial Decision Tree

The authors describe a protocol by which a user can classify her data point using a binary decision tree without giving away any information about her data, while also not learning any information about the path her data point took on the tree. To achieve this, the authors use the polynomial representation of the decision tree as their learned model and describe an algorithm, which uses both the QR scheme and a public-key FHE scheme, to classify a data point based on this representation. Specifically, the authors used the FHE scheme in HElib [59] to run their protocol.

3.3 Fully Homomorphic Encryption Schemes

The study of fully homomorphic encryption schemes has proceeded in three phrases described by Peikert in *A Decade of Lattice Cryptography* as follows [50]. The first phase consisted of the first publications of groundbreaking, yet impractical, FHE schemes beginning with Gentry's seminal thesis [27]. From there, the second wave of FHE began with a rapid series of efficiency improvements including bootstrapping and the development of SHE and LHE schemes. The third generation of FHE began around 2013, where practical improvements simplified and raised the efficiency of FHE schemes [2, 9, 32]. The following sections provide an overview of the major breakthroughs throughout these three waves as well as security assumptions underlying the schemes.

3.4 The First FHE Schemes

Gentry's thesis described the first fully homomorphic encryption scheme [27]. While Gentry's seminal work was not a practical scheme in terms of applications, it laid the foundations for a wealth of future work in public-key FHE. Subsequent works follow his method of first constructing a SHE scheme then bootstrapping this scheme into a FHE scheme.

Improvements on this original framework occurred rapidly in the years following this first publication. While improvements upon Gentry's initial scheme provide some schemes that are conceptually simpler, one common thread among public-key FHE schemes is the the time and difficulty it takes to describe them. Therefore, this chapter provides an overview of the concepts of these public-key FHE schemes and leave the details to the referenced papers.

Gentry's first FHE scheme is built around a mathematical construct called a *lattice*. A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^n$ [50]. More specifically, let $\mathcal{L}$ be a subset of $\mathbb{R}^n$, and let $\mathbf{0}$ denote the zero vector in $\mathbb{R}^n$. Say that $\mathcal{L}$ is a subgroup of $\mathbb{R}^n$ if $0 \in \mathcal{L}$, and $-x \in \mathcal{L}$ and $x + y \in \mathcal{L}$ for every $x, y \in \mathcal{L}$. This subgroup is called *discrete* if for every element $x \in \mathcal{L}$ there exists some neighborhood $N_x \subset \mathbb{R}^n$ such that $\mathcal{L} \cap N_x = \{x\}$. For instance, the integers $\mathbb{Z}^n$ form a discrete subgroup of $\mathbb{R}^n$, hence $\mathbb{Z}^n$ forms an n-lattice known as the integer lattice.

3.4.1 Gentry's Fully Homomorphic Encryption

Gentry's seminal work proceeded in three main steps. First, a somewhat homomorphic encryption scheme was constructed that is able to compute only a limited class of functions homomorphically. From here, a bootstrapping method was developed in order to enable arbitrary computation based on the limited class of functions in the previous step. Finally, bootstrapping is applied to the somewhat homomorpic encryption scheme in order to make it fully homomorphic by "squash(ing) the decryption circuits" [27].

What follows is an overview of Gentry's original scheme, which omits some of the finer details and proofs. Gentry's concept of a "bootstrapping" is described first, followed by an overview of his original somewhat homomorphic scheme and the bootstrapping procedure applied to it.

Consider a ring $R = \mathbb{Z}[x]/f(x)$ for monic degree n polynomials $f(x) \in \mathbb{Z}[x]$. This means $f(x)$ is of the form

$$f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

with coefficients $a_i \in \mathbb{Z}$. Gentry describes that elements $\mathbf{v} \in R$ can be thought of as (coefficient) vectors $\mathbf{v} \in \mathbb{Z}^n$, and the ideal generated by $\mathbf{v}$ yields the ideal lattice $(\mathbf{v})$ generated by

$$\{\mathbf{v} \times x^i \mod f(x) : i \in [0, n-1]\}.$$

Let I denote an ideal of R and $\mathbf{B}_I$ a basis of I. The `KeyGen` algorithm runs a sub-algorithm

$$\left(\mathbf{B}_J^{\text{pk}}, \mathbf{B}_J^{\text{sk}}\right) = \texttt{IdealGen}(R, \mathbf{B}_I)$$

where $\mathbf{B}_J^{\text{pk}}$ and $\mathbf{B}_J^{\text{sk}}$ are bases of an ideal J such that $I + J = R$.

Furthermore, let $\texttt{Samp}(\mathbf{x}, \mathbf{B}_I, R, \mathbf{B}_J)$ represent an algorithm that takes samples from $\mathbf{x}+I$. Gentry shows that this coset has a unique representative with respect to $\mathbf{B}_I$ that can be computed efficiently, meaning the value $\mathbf{x} \mod \mathbf{B}_I$ is unique. This holds for any $\mathbf{x} \in R$ and any ideal of R.

With this in mind, the encryption scheme $\mathcal{E}'$ runs as follows, where $R = \mathbb{Z}[x]/f(x)$ and ideal correspond to lattices as above.

I. $(\text{pk}, \text{sk}) = \texttt{KeyGen}_{\mathcal{E}'}(R, \mathbf{B}_I)$, where $\text{pk} = (R, \mathbf{B}_I, \mathbf{B}_J^{\text{pk}}, \texttt{Samp})$ and $\text{sk} = (\text{pk}, \mathbf{B}_J^{\text{sk}})$.

II. $c = \texttt{Encrypt}_{\mathcal{E}'}(\mathrm{pk}, m)$, the encryption algorithm. This algorithm encrypts a plaintext m by computing

$$c' = \texttt{Samp}(m, \mathbf{B}_I, R, \mathbf{B}_J^{\mathrm{sk}})$$

and returning $c = c' \mod \mathbf{B}_J^{\mathrm{pk}}$.

III. $m = \texttt{Decrypt}_{\mathrm{sk},c}$, the decryption algorithm, which computes

$$m = (c \mod \mathbf{B}_J^{\mathrm{sk}}) \mod \mathbf{B}_I$$

and returning m.

IV. $c = \texttt{Evaluate}_{\mathcal{E}'}(\mathrm{pk}, C, c_1, c_2)$ for a circuit C and ciphertexts c_1 and c_2. This algorithm evaluates C over c_1 and c_2 using sub-algorithms

$$\begin{aligned} \texttt{Add}(\mathrm{pk}, c_1, c_2) &= c_1 + c_2 \mod \mathbf{B}_j^{\mathrm{pk}} \\ \texttt{Mult}(\mathrm{pk}, c_1, c_2) &= c_1 \times c_2 \mod \mathbf{B}_j^{\mathrm{pk}} \end{aligned}$$

The class of circuits $\mathcal{C}_{\mathcal{E}'}$ that can be computed are also described by Gentry. Let X_{Enc} and X_{Dec} be subsets of $\mathbb{Z}^n$, where X_{Enc} is the set of all samples from the coset $\mathbf{x} + I$ and $X_{\mathrm{Dec}} = R \mod \mathbf{B_J}^{sk}$. Let $B(r)$ denote the ball of radius r in R. Then, there are values

$$\begin{aligned} r_{Enc} &= \min\{r : X_{Enc} \subset B(r)\} \\ r_{Dec} &= \max\{r : X_{Dec} \supset B(r)\} \end{aligned}$$

and permitted circuits are ones where an input in $B(r_{Enc})^t$ yields an output in $B(r_{Dec})$. Gentry shows that this constitutes the set of circuits with depth at most

$$\log\log r_{Dec} - \log\log n_{\mathrm{Mult}(R)} \cdot r_{Enc}$$

for some constant factor $n_{\mathrm{Mult}(R)}$ where $\|\mathbf{u} \times \mathbf{v}\| \leq n_{\mathrm{Mult}(R)} \cdot \|\mathbf{u}\| \cdot \|\mathbf{v}\|$.

Gentry concludes by showing that increasing the depth of the circuits that can be homomorphically evaluated requires minimizing $n_{\mathrm{Mult}(R)}$ and r_{Enc} and maximizing r_{Dec}.

A series of complex bootstrapping operations is carried out by Gentry in order to bring the scheme above scheme up to a FHE scheme. Broadly, he lowers the complexity of the decryption circuit of $\mathcal{E}'$ by reducing the size of r_{Dec} and alters the decryption algorithm to run simplified computations. Furthermore, he reduces the work the decryption algorithm must carry out by adding in preprocessing steps to the encryption algorithm that reduce the amount of work that must be performed during decryption.

This original scheme is impractical in multiple ways. First of all, it is conceptually dense and requires a large breadth of knowledge of advanced mathematics to understand. Beyond this, and more importantly, the computation time required to implement the fully homomorphic properties of this scheme is highly impractical.

3.5 Second-Generation FHE and Beyond

A number of papers were published in the years immediately following Gentry's original publication. Van Dijk, Gentry, Halevi, and Vaikuntanathan published a scheme in 2010 that avoided the average-case assumptions of lattice-based cryptography and instead was based simply on modular arithmetic [20]. As above, they started with a somewhat homomorphic scheme and applied bootstrapping to achieve a fully homomorphic scheme. The security of this scheme was based on the hardness of the *approximate-GCD* problem, where you must find an integer p given a set of randomly chosen integers that lie "close" to some multiple of p. Further work based on the approximate-GCD problem was carried out by Coron et al., who reduced the public key size of the scheme of Van Dijk et al. [17].

In another paper, Smart and Vercauteren describe a somewhat homomorphic encryption scheme, which supports SIMD (Single-Instruction Multiple-Data) operations, thus enabling a level of parallelization in FHE [60]. As before, bootstrapping methods were used to raise this scheme from SHE to FHE. Their scheme is lattice-based, the security of the scheme is based on a variant of the *Bounded Distance Decoding Problem* (BDDP), and the security of their bootstrapping procedure is based on the *Sparse Subset Sum Problem* (SSSP). BDDP seeks to find the closest lattice vector to a given vector (not necessarily in the lattice), and SSSP seeks to find a sparse subset of a larger set A whose sum is equal to a given number s modulo a given N.

A number of other papers contribute to this "second generation" of public-key FHE that follow Gentry's SHE-bootstrap-FHE blueprint [7, 9, 8, 10, 17, 28, 60]. A series of papers Zvika Brakerski and Vinod Vaikuntanathan published in 2011, "Fully Homomorphic Encryption from ring-LWE and Security for Key Dependent Messages" [9] and "Efficient Fully Homomorphic Encryption from (Standard) LWE" [8], were of particular impact. These papers use the SSSP as well as LWE and ring-LWE. The results in these papers were improved in a joint work with Gentry [11]. This paper, "Fully Homomorphic Encryption Without Bootstrapping," removed the need for expensive bootstrapping procedures entirely. Although bootstrapping is available as an optimization procedure in this scheme, it is not necessary to implement it to achieve FHE.

3.5.1 The Learning With Errors Problem

A major shift seen in this second generation of schemes is the use of the *learning with errors (LWE)* and *ring-learning with errors (ring-LWE)* problems as cryptographic foundations. The LWE problem was introduced by Regev [56] and is seen as ideal for cryptographic applications as its average-case hardness is as hard as the worst-case, up to a polynomial factor. Learning with errors asks us to find a vector $\mathbf{s} \in \mathbb{Z}_q^n$ given m samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where $\mathbf{a}_i$ is uniformly random in $\mathbb{Z}_q^n$, e is an error term taken from some error distribution, and

$$(\mathbf{a}_i, b) = \langle \mathbf{s}, \mathbf{a}_i \rangle + e \pmod{q}.$$

Abstractly, the goal is to recover the vector $\mathbf{s}$ when given some set of approximate linear equations over $\mathbf{s}$. The fact that these equations are approximate rather than exact is what leads to the presumed hardness of this problem. Furthermore, LWE is as hard in the average case as it is in the worst case, albeit up to a polynomial factor, an ideal situation for cryptography.

Ring-LWE is a ring-based variant on LWE introduced in [46]. Ring-LWE takes m samples (a_i, b_i) from $R_q \times R_q$ for a ring R of degree n over $\mathbb{Z}$, where $R_q = R/qR$. Again, there is an error term e taken from an error distribution χ over R (typically an embedded discrete Gaussian). The ring-LWE problem asks us to distinguish whether the given m samples were taken from the uniform distribution or from the ring-LWE distribution containing samples of the form

$$(a, b) = s \cdot a + e \pmod{q}$$

for an unknown term $s \in R_q$. The ring-LWE problem was shown to be at least as hard as quantumly solving the shortest vector problem (SVP) [46]. SVP is a hard problem based on ideal lattices, which has a best known solution taking exponential time in the quantum setting [50].

Brakerski, Gentry, and Vaikuntanathan define a further problem, the general learning with errors problem (GLWE), as a generalized and combined version of LWE and ring-LWE. In this construction, the ring is given by $R = \mathbb{Z}[x]/(x^d + 1)$ where d is a power of 2. LWE is considered as a sub-case of ring-LWE, which occurs when $d = 1$.

3.5.2 The Brakerski-Gentry-Vaikuntanathan (BGV) Scheme

The Brakerski-Gentry-Vaikuntanathan (BGV) scheme is a ring-based scheme based on the assumption of the infeasibility of the GLWE problem [11]. It uses either the ring $R = \mathbb{Z}$ or $R = \mathbb{Z}[x]/(x^d + 1)$, where d is a power of 2, as a platform. The BGV scheme proved to be a major step forward from Gentry's original scheme. It provided a reduction in noise growth via a technique called modulus-switching, smaller key sizes, and the introduction of SIMD parallelization to FHE.

This section provides an overview of the algorithms in the authors' basic scheme in the ring-LWE case, leaving out some of the more technical mathematical details.

I. $(\text{params}, \text{pk}, \text{sk}) = \texttt{KeyGen}(1^\lambda, 2^L)$, the key generation algorithm over some security parameters λ and L. This key generation algorithm contains two sub-algorithms called $(\{\text{params}\}) = \texttt{Setup}(1^\lambda, 1^L)$ and $(\text{sk}, \text{pk}) = \texttt{KeyGen}(\{\text{params}\})$. Defining these algorithms in detail is very technical, and details on parameter and key construction can be found in the original paper [11]. It suffices to note that the private key consists of L vectors $\mathbf{s}_1, \ldots, \mathbf{s}_L \in R_q^{n+1}$ and the public key consists L matrices $\mathbf{A}_1, \ldots, \mathbf{A}_L \in R_q^{N\times(n+1)}$.

II. $\mathbf{c} = \texttt{Encrypt}(\text{params}, \text{pk}, m)$, the encryption algorithm, which takes as input the security parameters, public key pk, and a message $m \in R_q^{n+1}$. To encrypt, take a random sample $\mathbf{r} \leftarrow R_2^N$. The ciphertext $\mathbf{c} \in R_q^{n+1}$ is given by

$$\mathbf{c} = (m, 0, \ldots, 0) + \mathbf{A}_j^T \mathbf{r}.$$

for some j such that $1 \leq j \leq L$.

III. $m = \texttt{Decrypt}(\text{params}, \text{sk}, \mathbf{c})$, the decryption algorithm, which outputs

$$m = [[\langle \mathbf{c}, \mathbf{s}_j \rangle]_q]_2.$$

assuming $\mathbf{c}$ is encrypted under public key $\mathbf{A}_j$. Note that the above notation represents modular reduction in $(-q/2, q/2]$ followed by reduction modulo 2.

IV. $\mathbf{c}_4 = \texttt{Add}(\text{pk}, \mathbf{c}_1, \mathbf{c}_2)$, the homomorphic addition algorithm, which takes as input the public keys along with two ciphertexts encrypted under the same secret key $\mathbf{s}_j$. Note that the `Refresh` algorithm below will allow us to perform key switching for ciphertexts *not* encrypted under the same $\mathbf{s}_j$. Compute homomorphic addition:

$$\mathbf{c}_3 = \mathbf{c}_1 + \mathbf{c}_2 \pmod{q_j}$$

where $q_j \in$ params. The ciphertext $\mathbf{c}_3$ is associated with the private key $\mathbf{s}'_j = \mathbf{s}_j \otimes \mathbf{s}_j$, the tensor of $\mathbf{s}_j$ with itself. Output $\mathbf{c}_4 = \texttt{Refresh}(\mathbf{c}_3, \text{params})$.

V. $\mathbf{c}_4 = \texttt{Mult}(\text{pk}, \mathbf{c}_1, \mathbf{c}_2)$, the homomorphic multiplication operation, which again takes as input ciphertexts encrypted under the same decryption key $\mathbf{s}_j$. Compute $\mathbf{c}_3$ as the coefficients of the linear equation given by

$$\langle \mathbf{c}_1, \mathbf{x} \rangle \cdot \langle \mathbf{c_2}, \mathbf{x} \rangle.$$

Output $\mathbf{c}_4 = \texttt{Refresh}(\mathbf{c}_3, \text{params})$.

VI. $\texttt{Refresh}(\mathbf{c}, \text{params})$, an algorithm to perform key switching. The details of this algorithm are left for [11]; the procedure involves a modulus switching procedure and a key switching procedure to change a ciphertext from being encrypted under $\mathbf{s}'_j$ to encryption under $\mathbf{s}_{j-1}$ with modulus q_{j-1}.

The `Refresh` procedure must be carried out during multiplication in order to perform noise reduction. The noise added during addition is always negligible enough for this procedure to be unnecessary. Observe furthermore that bootstrapping is not required in order for this scheme to be fully homomorphic. A bootstrapping procedure is provided by the authors, however, in order to increase efficiency in terms of performance.

3.5.3 HElib

Shai Halevi, a researcher at IBM, wrote a software library called HElib, which implements a version of the BGV scheme in C++, and is publicly available on GitHub [59]. HElib implements a number of optimizations on the BGV scheme in order to decrease runtime of algorithms [38]. Specifically, HElib uses what is called the single instruction multiple data (SIMD) paradigm to perform parallel computations and increase runtimes. HElib also implements optimizations published by Gentry, Halevi, and Smart [30, 29, 31] as well as ciphertext packing techniques published by Smart and Vercauteren [60].

3.5.4 Yet Another Somewhat Homomorphic Encryption Scheme (YASHE)

Another popular somewhat homomorphic encryption scheme introduced around this time was YASHE, or "yet another somewhat homomorphic encryption scheme" [5]. While the scheme still requires key-switching methods, this does not utilize any modulus-switching methods such as seen in the BGV scheme; the authors call this property *scale-invariance*. Furthermore, YASHE uses a smaller ciphertext size than BGV as ciphertexts are given by just one ring element (as opposed to multiple ring elements in BGV). YASHE is secure under the ring-LWE assumption.

An overview of the scheme is provided. The ring R is given by $R = \mathbb{Z}[x]/(\Phi_d(x))$, where $\Phi_d(x)$ is the dth cyclotomic polynomial. The dth cyclotomic polynomial is the unique irreducible polynomial in $\mathbb{Z}[x]$ that divides $x^d - 1$ but does not divide $x^\ell - 1$ for any $\ell < d$, and is of degree $n = \varphi(d)$, where φ denotes Euler's totient function. Also assume there is some error distribution χ as well as a parameters q and t, $1 < t < q$.

I. $(\mathrm{pk}, \mathrm{sk}, \mathrm{evf}) = \texttt{KeyGen}(\lambda)$, the key generation algorithm over some security parameter λ. Run a sub-algorithm to generate the parameters d, q, g, and distributions χ, and χ'. Given samples $f', g \leftarrow \chi'$, set $f = [tf' + 1]_q$ and verify that f is invertible modulo q. (If not, repeat.) The private key is f and the public key is given by h, where $h = [tgf^{-1}]_q$. A further evaluation key γ is computed for use in the `KeySwitch` algorithm. Output $(\mathrm{pk}, \mathrm{sk}, \mathrm{evf}) = (f, h, \gamma)$.

II. $c = \texttt{Encrypt}(\mathrm{pk}, m)$, which encrypts a message m as a ciphertext $c \in R$. Let $[m]_t$ be a representative of $m + tR$ and let $s, e \leftarrow \chi$. Output $c = [\lfloor q/t \rfloor [m]_t + e + hs]_q$.

III. $m = \texttt{Decrypt}(\mathrm{sk}, c)$, which outputs

$$m = \left[\left\lfloor \frac{t}{q} \cdot [fc]_q \right\rceil \right]_t .$$

IV. $c_3 = \texttt{Add}(c_1, c_2)$, which computes addition as

$$c_3 = [c_1 + c_2]_q .$$

V. $c_3 = \texttt{Mult}(c_1, c_2, \mathrm{evk})$, which computes multiplication as

$$\tilde{c} = \left[\left\lfloor \frac{t}{q} c_1 c_2 \right\rceil \right]_q .$$

for a function P which embeds $c_1 \in R$ in R^ℓ for a parameter ℓ. The value $\tilde{c}$ is an encryption under f^2. The output is given by $c_3 = \texttt{KeySwitch}(\tilde{c}, \mathrm{evk})$, which transforms the ciphertext $\tilde{c}$ into a ciphertext c_3 that is can be decrypted under the original key secret key, f.

This leveled fully homomorphic scheme is brought up to a fully homomorphic scheme via bootstrapping procedures as described by Gentry in his original publication [26]. As with the other public-key FHE schemes, homomorphic computation in YASHE results in noise that grows with each computation and imposes a limit on the number of computations that may be carried out before the noise grows too large.

3.5.5 Fan-Vercauteren (FV) Encryption

Fan and Vercauteren introduced a scheme, commonly referred to as the FV scheme [25]. This scheme is a version of Brakerski's scheme [7], which is based off of the ring-LWE problem rather than LWE. The bootstrapping procedure introduced by the authors used modulus switching in order to create a simpler method.

In this scheme the plaintext ring is given by $R_t = \mathbb{Z}_t[x]/(\varphi(x))$ for an integer $t > 1$ and monic, irreducible polynomial $\varphi \in \mathbb{Z}[x]$ of degree d. Let R denote the ring $\mathbb{Z}[x]/(\varphi(x))$.

I. $(\mathrm{pk}, \mathrm{sk}) = \texttt{KeyGen}(\lambda)$, the key generation algorithm with security parameters λ. Let χ be a distribution on $\mathbb{Z}[x]/(\varphi(x))$, described by the authors. The public key pk is given by

$$(f_0, f_1) = ([-(a \cdot g + e)]_q, a),$$

where $a \leftarrow R_q$, $e, g \leftarrow \chi$, and $[x]_q$ denotes reduction of all coefficients of $x \in R$ modulo q. Output $(\mathrm{pk}, \mathrm{sk}) = ((f_0, f_1), g)$.

II. $c = (c_0, c_1) = \texttt{Encrypt}(\mathrm{pk}, m)$, which encrypts a message $m \in R_t$ as the ciphertext $c \in R_q^2$ given by

$$(c_0, c_1) = \left(\left[f_0 \cdot u + e_1 + \left\lfloor \frac{q}{t} \right\rfloor \cdot m \right]_q, [f_1 \cdot u + e_2]_q \right)$$

where $u, e_1, e_2 \leftarrow \chi$.

III. $m = \texttt{Decrypt}(\mathrm{sk}, c)$, which decrypts by computing

$$m = \left[\left\lfloor \frac{t \cdot [c_0 + c_1 \cdot s]_q}{q} \right\rceil\right]_t$$

where $s = \mathrm{sk}$ is the secret key.

IV. $e = \texttt{Add}(c, d)$ adds ciphertexts $c = (c_0, c_1)$ and $d = (d_0, d_1)$ to obtain the ciphertext $e = (e_0, e_1)$ via the operation

$$(e_0, e_1) = \left([c_0 + d_0]_q, [c_1 + d_1]_q\right).$$

V. $e = \texttt{Mult}(c, d)$ multiplies ciphertexts $c = (c_0, c_1)$ and $d = (d_0, d_1)$ to obtain the ciphertext $e = (e_0, e_1)$. First, compute

$$\begin{aligned}
\bar{e}_0 &= \left[\left\lfloor \frac{t \cdot (c_0 \cdot d_0)}{q} \right\rceil\right]_q \\
\bar{e}_1 &= \left[\left\lfloor \frac{t \cdot (c_0 \cdot d_0 + c_1 \cdot d_1)}{q} \right\rceil\right]_q \\
\bar{e}_2 &= \left[\left\lfloor \frac{t \cdot (c_1 \cdot d_1)}{q} \right\rceil\right]_q
\end{aligned}$$

Next, there are two variants on the final computation of multiplication, details of which are highly technical and left to the original paper [25]. The first method minimizes error due to noise accumulation, while the second method is similar to modulus-switching methods and reduces the time and space used during computation.

This leveled fully homomorphic encryption scheme is brought up to a FHE scheme via bootstrapping procedures.

3.5.6 Third-Generation Public-Key FHE: Recent Developments

Around late 2012 or early 2013 marked the switchover to what Peikert terms the third generation of FHE, marked by a shift in the internal methods used within the schemes [50]. Gentry, Sahai, and Waters published a FHE scheme in 2013 based on LWE that introduced the approximate eigenvector method for multiplication [32]. This method reduces homomorphic multiplication in most cases to matrix multiplication, a much more efficient procedure. Gentry, Halevi, and Smart revisited bootstrapping procedures to reduce the computational bottleneck involved in these algorithms [30].

3.6 Gribov-Kahrobaei-Shpilrain (GKS) Encryption

Alexey Gribov, Delaram Kahrobaei, and Vladimir Shpilrain presented a novel approach to fully homomorphic encryption called the Gribov-Kahrobaei-Shpilrain (GKS) cryptosystem [36]. This ring-based scheme is able to avoid much of the computational overhead required for fully homomorphic public key encryption. The authors define the platform ring S_n as

$$S_n = \langle x_1, x_2, \ldots, x_n | p \cdot 1 = 0, x_i^2 = x_i, \text{ and } x_i x_j = x_j x_i \text{ for all } i, j \rangle. \tag{3.1}$$

This ring contains a super-exponential number of *idempotent* elements, or elements $g \in S_n$ such that $g^2 = g$. Some idempotent elements are given by

$$e_F := \prod_{i \in F} x_i \cdot \prod_{j \notin F} (1 - x_j)$$

for any set of indexes $F \in \mathcal{P}(\{1, 2, \ldots, n\})$. The above construction yields 2^n idempotent elements that are pairwise orthogonal, meaning $e_F e_G = 0$ whenever $F \neq G$, and represent a linear basis of S_n over $\mathbb{Z}_p$.

The scheme is ring-based and involves a private plaintext ring P and a ciphertext ring C, where P is a retract of C, i.e., $P \subset C$ and at the same time P is a factor ring of C. Both P and C are rings of the form of Equation 3.1. Let I be an ideal of C such that C/I is isomorphic to P.

The data owner, D, generates C and I during the key generation algorithm. Parameters n, r, and p are chosen such that p is a large prime, $r > n$, and let $P := S_n$. The ciphertext ring $C = S_r$ is randomly generated from $P = S_n$ as $\{x_i\}_{i=1}^{s}$, where $s > n$. In other words, the ring $C = S_r$ expands upon the generators of $P = S_n$. The data owner then randomly chooses an ideal I by randomly selecting elements

$$x_m - w_m(x_1, x_2, \ldots, x_{m-1})$$

for $m = n+1, \ldots, r$, where $w_m(x_1, x_2, \ldots, x_{m-1})$ represents an idempotent element of S_{m-1}. Observe that C/I is isomorphic to P by construction. The data owner next rewrites C in terms of its orthogonal basis $\{e_i\}_{i=1}^{2^r}$ and applies a random permutation π to the orthogonal set generators. D concludes the key generation algorithm with a final transformation between this permuted basis, given by

$$C = \langle e_1, e_2, \ldots, e_{2^r} | p \cdot 1 = 0, e_i^2 = e_i,$$

$$\text{and } e_i e_j = 0 \text{ for all } i, j \rangle,$$

and a *triangular basis*. The triangular basis as described by the authors applies an invertible linear transformation of the form

$$t_j = \sum_{i \in F_j} e_{ij},$$

where F_j denotes a set of indices. These index sets satisfy the property that $F_j \subseteq F_k$ whenever $j < k$. As the elements in this basis are not orthogonal, homomorphic multiplication does not occur component-wise. Rather, it follows the property that $t_i t_k = t_k$ whenever $j \leq k$.

A plaintext element $x \in P$ is encrypted as

$$\texttt{Encrypt}(x) = x + E_0$$

where

$$E_0 = \sum_{j=n+1}^{r} (x_j - w_j(x_1, \ldots, x_{j-1})) \cdot h_j(x_1, \ldots, x_r)$$

for random $h_j \in C$. This plaintext element is then converted to the published triangular basis. To decrypt, the data owner simply converts the ciphertext back to the basis $\{x_i\}_{i=1}^{r}$ then replaces x_j with $w_j(x_1, \ldots, x_{j-1})$ for $j = n+1, \ldots, r$.

The authors show that this scheme is secure against a ciphertext-only attack. This scheme has the advantage of not accumulating noise terms during computation. Thus, the only limit on computation in the GKS scheme lies in the value of the prime modulus p in the definition of the ring S.

3.7 Encoding Data for Fully Homomorphic Computation

Fully homomorphic computation can only be completely utilized over data encoded in a fully homomorphic manner. Various types of data such as strings, integers, and floats must be encoded such that computation over these values preserves the fully homomorphic properties of the scheme. For instance, the ASCII encoding for 0 is `48` – but `48` + `48` = `96`, the ASCII code for the grave accent. Encoding via ASCII will not suffice when encrypted addition is required.

In addition to a fully homomorphic encoding method it is important that the method chosen is efficient in terms of memory usage. While the precise methods of encoding depend on the structure of the encryption scheme, often what is required is an encoding with integer values. Fully homomorphic encoding methods used in the FHE software SEAL are described below. The fully homomorphic encoding method for GKS encryption are discussed in Chapter 4

3.7.1 Fully Homomorphic Encoding in SEAL

SEAL, or the Simple Encrypted Arithmetic Library, is a software for performing machine learning using FHE developed by Microsoft. A previous version of SEAL implemented a variant of the YASHE scheme developed by researchers at Microsoft's Cryptography Research Group [21]. The most recent version of Microsoft's Simple Encrypted Arithmetic Library (SEAL) [43] implements the "FullRNS" variant [4] of the Fan-Vercauteren somewhat homomorphic scheme [25].

Encoding Integers

Plaintexts and ciphertexts in the YASHE implementation of SEAL are polynomials in the ring $R_t = \mathbb{Z}_t[x]/(x^n + 1)$ and $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, respectively. The authors provide three methods for encoding integer values.

The first method is the least memory efficient, but the most straightforward. A plaintext value $y \in \mathbb{Z}$ is encoded as the constant polynomial $p(x) = y$. This encoding method requires the moduli t and q to be incredibly large to avoid overflow. The second method they present is to encode a value $y \in \mathbb{Z}$ as using its binary representation $y = \sum_i b_i 2^i$ for bits b_i, and encoding a plaintext polynomial

$$p(x) = \sum_i b_i x^i.$$

Decoding is performed by evaluating $p(2)$. The authors explain that this method may be applied to base-b encodings for larger integers b, with higher base values resulting in shorter polynomials with larger coefficients.

The last encoding method described is based off of the Chinese Remainder Theorem (CRT). A plaintext is encoded multiple times under co-prime moduli $t_1, \ldots, t_k$. The CRT allows for decoding by combining these individual plaintexts under a single modulus $\prod t_i$. While this encoding method requires more space, it does allow for much smaller moduli.

Encoding Reals

Real values can be encoded in the YASHE version of SEAL by scaling them to integer values and performing scaling after every use of homomorphic multiplication. However, this often results in prohibitively large integers that must be encoded. Instead, the authors suggest encoding floating point values again via binary representation. A floating point value y can be represented as $y_+ + y_-$, where $y_+ = \sum_{i=0}^{B} b_i 2^i$ and $y_- = \sum_{j=1}^{C} c_j 2^{-j}$ for bits b_i and c_j. The plaintext y may then be encoded as

$$\sum_{i \leq B} b_i x^i - \sum_{j < C} x^{n-j} c_j.$$

This method requires reserving the coefficients for the integer and fractional parts of y before of encoding.

Plaintext Packing

The authors suggest a method called plaintext packing to decrease the number of computations that must be performed. This method involves encoding multiple small messages as one large message in order to reduce overall run time, using what is called the single instruction multiple data (SIMD) paradigm.

The method for plaintext packing in YASHE SEAL involves using co-prime polynomials

$Q_1, \ldots, Q_k$ such that

$$x^n + 1 = \prod_{i=1}^{k} Q_i(x) \pmod{t}.$$

There is an isomorphism between the ring R and a co-prime polynomial version of the ring given by

$$\frac{\mathbb{Z}_t[x]}{(x^n+1)} \cong \prod_{i=1}^{k} \frac{\mathbb{Z}_t[x]}{(Q_i(x))} \pmod{t}.$$

Multiple values can be encoded at once as the constant coefficient of each of the k factors.

Above is a formulation of the Chinese Remainder Theorem over rings. The most recent version of SEAL [43] based off of the CV encryption scheme similarly uses the Chinese Remainder Theorem in order to provide SIMD operations, which they call "batching."

3.7.2 Fully homomorphic encoding in GKS

Gribov, Kahrobaei, and Shpilrain provide a method for encoding integer values within the platform ring of the GKS cryptosystem [36]. A fully homomorphic embedding of $\mathbb{Z}_p$ into the platform ring S_n is chosen. This corresponds to letting the element $1 \in \mathbb{Z}_p$ be mapped to any idempotent in S.

Methods for encoding real numbers as well as implementing SIMD batching within the GKS cryptosystem are presented in this work. Chapter 4 discusses methods for these encodings, as well as various other implementation methods for the GKS cryptosystem.

3.8 Conclusions

Current methods in privacy-preserving classification span non-FHE encryption methods, FHE methods, and differential privacy. Fully homomorphic encryption developed quickly from its first presentation by Gentry to the more efficient forms available today. Popular schemes such as YASHE [5] and FV [25] have been implemented for machine learning tasks using HElib [59] and SEAL [43], respectively. The GKS scheme [36] is a conceptually simple scheme without the noise seen in other FHE schemes. Methods for the efficient implementation of GKS encryption for real-world classification tasks will be proposed throughout the remainder of this dissertation.

Chapter 4

Implementation of the Gribov-Kahrobaei-Shpilrain (GKS) Scheme

There are a number of implementation issues that must be addressed when performing fully homomorphic encryption over real world data. The encoding method used for integers, floats, and rationals needs to result in fully homomorphic computations within the ring.

In addition to a fully homomorphic encoding method it is important that the method chosen is efficient in terms of memory usage. While the precise methods of encoding depend on the structure of the encryption scheme, often what is required is an encoding with integer values. The GKS scheme requires all elements to be embedded as ring elements with coefficients in $\mathbb{Z}_p$. This ring structure can be taken advantage of in order to implement single instruction, multiple data (SIMD) instructions to maximize the amount of computations carried out in a single instruction.

This chapter outlines parameter selection for applications using the GKS scheme as well as implementation algorithms, fully homomorphic encoding methods, and how to implement

SIMD instructions within GKS in order to maximize the efficiency of the memory used by the program. This chapter concludes by providing data on the speed of the described algorithms in a C++ implementation.

4.1 Fully Homomorphic Encoding of Real-World Values in GKS

Because the GKS encryption scheme is based off of rings elements with coefficients in the field $\mathbb{Z}_p$, any encoding used will have to be based off of integer values. Gribov, Kahrobaei, and Shpilrain provide a straightforward method of implementing a fully homomorphic encoding [36]. Any fully homomorphic embedding of $\mathbb{Z}_p$ into the ring S_n will suffice. In particular, it is sufficient to map the element 1 in $\mathbb{Z}_p$ to any idempotent element of S.

Encoding Reals

Real-valued variables must also be encoded as elements of the plaintext ring. This method should encode floating point values as integer values, be fully homomorphic, avoid overflow in $\mathbb{Z}_p$, and not lose any accuracy compared to the unencrypted operations.

To achieve this, scale floating point values to integer values with a fixed level of precision. Encode with n digits of precision using an encoding function $\texttt{Encode}(x) := \lfloor x \cdot 10^n \rfloor$. For instance, if $n = 3$ encode $x = 0.29128$ as

$$\texttt{Encode}(x) = \lfloor 0.29128 \cdot 10^3 \rfloor = 291.$$

Extend this to an encoding in the ring S_n by mapping x to a fixed idempotent value in S. Decode simply by retrieving the coefficient of the fixed idempotent value in S and multiplying by 10^{-n}.

This certainly yields an additively homomorphic embedding of a floating-point value in the ring. Maintaining a multiplicatively homomorphic property requires keeping track of the *depth* of each element. Consider first an example: the floating-point value 0.291 multiplied by 0.895 yields 0.260445. However, the depth 3 encoded value 291 multiplied by 895 yields 260445, which decodes to 260.445. Hence, to achieve a fully homomorphic encoding, keep track of the depth of each encoded value and decode accordingly. The depth of an encoded value is initialized to $d = 1$, and each time multiplication is performed, it's depth is incremented. To decode, multiply the coefficient by 10^{-dn}.

Note furthermore only elements which share a depth may be multiplied. To increase the depth of an element, simply multiply by the scalar 10^n. Keep proper track of the depth of elements during implementation in order to perform homomorphic multiplication.

Furthermore, the prime modulus p must be selected properly to avoid overflow over the modulus during computation. Note that this problem is not unique to the GKS cryptosystem, as all of the FHE schemes surveyed encode floating points as integer values in the range $(-p/2, p/2]$. The prime modulus must be chosen during parameter selection to be large enough to avoid overflow during computation.

4.2 Parallelization via SIMD

SIMD, introduced in Chapter 2.3, is implemented in the GKS scheme by encoding multiple plaintext values within a single plaintext ring element, i.e., by mapping each to a unique idempotent element in the ring basis. Note, however, that not every selection of idempotent elements results in a fully homomorphic encoding. Consider the plaintext ring S_n,

$$S_n = \langle x_1, x_2, \ldots, x_n | p \cdot 1 = 0, x_i^2 = x_i, \text{ and } x_i x_j = x_j x_i \text{ for all } i, j \rangle.$$

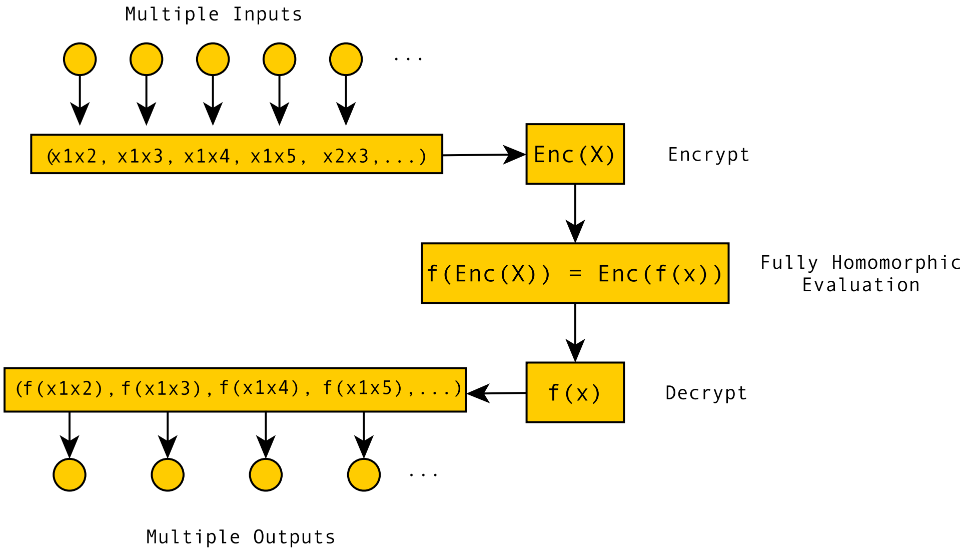

Multiple Inputs
...
(x1x2, x1x3, x1x4, x1x5, x2x3,...)
Enc(X)
Encrypt
f(Enc(X)) = Enc(f(x))
Fully Homomorphic Evaluation
f(x)
Decrypt
(f(x1x2),f(x1x3),f(x1x4), f(x1x5),...)
...
Multiple Outputs

Figure 4.1: SIMD Implementation in the GKS scheme

Two values are encoded simultaneously in a ring element by mapping each to fixed idempotent element, e.g. x_1 and x_1x_n. Then,

$$(a_1x_1 + b_1x_1x_n) + (a_2x_1 + b_2x_1x_n) = (a_1 + a_2)x_1 + (b_1 + b_2)x_1x_n$$

and addition is homomorphic over each element. However, multiplication via this encoding is not homomorphic component-wise, as

$$(a_1x_1 + b_1x_1x_n)(a_2x_1 + b_2x_1x_n) = a_1a_2x_1 + (a_1b_2 + b_1a_2 + b_1b_2)x_1x_n \tag{4.1}$$

when a true SIMD operation would yield $a_1a_2x_1 + b_1b_2x_1x_n$. Therefore, implementing SIMD requires a method of encoding multiple values within a single vector that is multiplicatively homomorphic not just over the ring elements, but also over the encoded coefficients.

In order to implement SIMD, the properties of the ring elements are used in order to describe a multiple embedding. With that in mind as the ultimate goal, consider the following claim.

Proposition 4.2.1. *Let $I, J \subseteq \{1, 2, \ldots, n\}$ be two subsets of indexes, and consider the two idempotent elements in S_n generated by these subsets defined as*

$$x_I = \prod_{i \in I} x_i \text{ and } x_J = \prod j \in J x_j.$$

Encoding two elements within a vector as coefficients of x_I and x_J will yield a fully homomorphic encoding if $I \cap J \neq I$ and $I \cap J \neq J$.

Proof. Assume $I \cap J \neq I$ and $I \cap J \neq J$. Then,

$$(a_1 x_I + b_1 x_J)(a_2 x_I + b_2 x_J) = a_1 a_2 x_I + b_1 b_2 x_J + (a_1 b_2 + a_2 b_1) x_I x_J.$$

Since $I \cap J \neq I$ and $I \cap J \neq J$, then $x_I x_J \neq x_I$ and $x_I x_J \neq x_J$. Therefore, decoding the above product yields $a_1 a_2$ and $b_1 b_2$, the correct product, and $x_I x_J$ contains a value which may be ignored. Inductively, this extends to any collection $I_1, I_2, \ldots, I_k \subseteq \{1, 2, \ldots, n\}$ satisifying $I_i \cap I_j \neq I_i$ and $I_i \cap I_j \neq I_j$ for any $1 \leq i, j \leq k$. □

While there are many subsets of elements that satisfy this criterion, a straightforward approach is to select distinct subsets $I_i, \ldots, I_k$ such that $|I_1| = |I_2| = \cdots = |I_k|$. If $|I_j| = s$ for all $1 \leq j \leq k$, then the number of SIMD slots available via this method is maximized whenever $\binom{n}{s}$ achieves its maximum over s. This corresponds to the maximum value for a binomial coefficient and occurs when $s = \lfloor n/2 \rfloor$ or $s = \lceil n/2 \rceil$.

Proposition 4.2.2. *Assume that $\binom{n}{\lfloor n/2 \rfloor}$ integer values are simultaneously encoded in the GKS scheme by mapping each element as the coefficient of the ring element generated by a set of indexes I satisfying $|I| = \lfloor n/2 \rfloor$. This embedding remains fully homomorphic when random values are encoded as the coefficients of elements generated by index sets J where*

$|J| > |I|$, *but fails to be fully homomorphic if there are any nonzero coefficients for elements with index sets where* $|J| < |I|$.

Proof. For the former case, consider two sets of indexes I and J where $|I| = \lfloor n/2 \rfloor$ and $|J| > |I|$. Then, the product $x_I x_J = x_K$ has an index set K where $|K| > |I|$. This value will have no impact on the values of the coefficients of the encoded elements.

An example of the latter case is available in Equation 4.1. More generally, say there is a nonzero coefficient for an element with index set $|J| < |I|$, called x_J. Let x_I be a word where $|I| = \lfloor n/2 \rfloor$ and $J \subset I$, which must exist for some set of indexes as $|J| < |I|$. Then,

$$\begin{aligned}(a_1 x_I + b_1 x_J)(a_2 x_I + b_2 x_J) &= a_1 a_2 x_I + b_1 b_2 x_J + (a_1 b_2 + a_2 b_1) x_I x_J \\ &= (a_1 a_2 + a_1 b_2 + a_2 b_1) x_I + b_1 b_2 x_J,\end{aligned}$$

since $x_I x_J = x_I$. □

Example 4.2.3. *When the plaintext ring is generated by* $n = 5$ *elements in the GKS scheme, the described encoding method allows for* $\binom{5}{2} = 10$ *elements to be simultaneously encoded in SIMD slots as coefficients of the elements* x_1x_2, x_1x_3, x_1x_4, x_1x_5, x_2x_3, x_2x_4, x_2x_5, x_3x_4, x_3x_5, *and* x_4x_5. *Furthermore, the values generated by* 3, 4, *and* 5 *elements may be assigned random coefficients during encoding.*

SIMD via Orthogonal Elements

A second fully homomorphic encoding utilizing SIMD slots is attained using the orthogonal representation of ring elements. Recall that ring elements are initially embedded in the standard basis in the plaintext ring

$$S_n = \langle x_1, x_2, \ldots, x_n | p \cdot 1 = 0, x_i^2 = x_i, \text{ and } x_i x_j = x_j x_i \text{ for all } i, j \rangle$$

which is extended to the ring S_r for $r > n$ via the method described in Chapter 3. Elements are then converted to a representation in the orthogonal basis as

$$S_n = \langle e_1, e_2, \ldots, e_{2^r} | p \cdot 1 = 0, e_i^2 = e_i, \text{ and } e_i e_j = 0 \text{ for all } i \neq j \rangle$$

via the change-of-basis

$$e_F := \prod_{i \in F} x_i \cdot \prod_{j \notin F} (1 - x_j)$$

for all sets of indexes $F \subset \{1, 2, \ldots, n, \ldots, r\}$.

A set $A = a_i$ of 2^n integers can be simultaneously encoded in the GKS scheme using this transformation over the original plaintext ring. Let a_F denote the element of A with index $i = \sum_{f \in F} 2^{f-1}$. In other words, use the binary representation of the selection of indexes F in order to perform a one-to-one mapping of the elements of A. This transformation is given by

$$\texttt{Encode}(A) = \sum_{F \subseteq \{1,2,\ldots,n\}} a_F \left(\prod_{i \in F} x_i \cdot \prod_{j \notin F} (1 - x_j) \right).$$

Values are decoded by inverting the orthogonal transformation in S_n. This method requires more effort to implement encoding but allows for 2^n elements to be encoded simultaneously. With the settings recommended by Gribov, Kahrobaei, and Shiplirain, a total of $2^5 = 32$ values may be encoded in one ciphertext via this method. Note that when implementing encryptions via this encoding, restrictions on an acceptable encryption of zero must be put in place to require that no coefficient in the encryption of zero is equal to 0.

4.3 Generating the Triangular Basis Transformation

Encryption in the GKS scheme, discussed in Chapter 3, begins with elements expressed in terms of the standard basis in the ring

$$C = \langle x_1, x_2, \ldots, x_r | p \cdot 1 = 0, x_i^2 = x_i, \text{ and } x_i x_j = x_j x_i \text{ for all } i, j \rangle. \tag{4.2}$$

Elements are then converted to the orthogonal basis via the transformation given by

$$e_F := \prod_{i \in F} x_i \cdot \prod_{j \notin F} (1 - x_j)$$

for each set of indexes F. Assign each orthogonal element e_F a numerical index by setting $e_F = e_i$ where $i = \sum_{f \in F} 2^{f-1}$. As above, this corresponds simply to converting the binary representation of the choice of index subset to decimal.

Lastly the ciphertext is transformed into the triangular basis, which satisfies

$$t_j = \sum_{i \in F_j} e_{ij},$$

for a set of indexes F_j, where $F_j \subseteq F_k$ whenever $j \leq k$ for k from 1 to 2^r.

Implementation of GKS encryption requires a method for randomly generating a triangular basis satisfying the above conditions. The only truly invertible linear transformation of the above form is given by

$$t_m = \sum_{i=1}^{m} e_k$$

for $1 \leq k \leq 2^r$.

In order to introduce randomness to this transformation, randomly repeat m of the rows.

The final, public ciphertext ring is given by

$$\tilde{C} = \langle t_1, t_2, \ldots, t_{2^r+m} | p \cdot 1 = 0, e_i^2 = e_i,$$
$$\text{and } e_i e_j = 0 \text{ for all } i, j \rangle.$$

Note that in the published basis, homomorphic addition occurs component-wise and the homomorphic multiplication operation is given by $t_i t_j = t_i$ whenever $i \leq j$.

Elements in the orthogonal basis are not uniquely represented by elements in the triangular basis. For example, if $t_1 = e_1$, $t_2 = e_1$, $t_3 = e_1 + e_2$, and $t_4 = e_1 + e_2$, then the element e_2 in the orthogonal basis can be written in the triangular basis in four ways: $t_3 - t_1$, $t_3 - t_2$, $t_4 - t_1$, or $t_4 - t_2$. When converting from the orthogonal to the triangular basis, the user should randomly choose which representation to use during each conversion. While this transformation is not invertible in the traditional sense, the key holder can convert back to the orthogonal basis during decryption easily by replacing each t_i with its corresponding orthogonal basis form and combining like terms.

4.3.1 Algorithms for Triangular Basis Implementation

The triangular basis should be generated during the `KeyGen` phase of GKS encryption. The proposed method for creating the triangular basis requires randomly repeating some number of rows, say m rows. The number of rows repeated can be given as a parameter during key generation or can be randomized during key generation. In either case, the number of repeated rows will be published along with the ciphertext ring.

Below the necessary algorithms for conversion between the orthogonal and triangular bases are outlined. Algorithm 1 describes the method for computing the change-of-basis matrix to the triangular basis from the orthogonal basis. Algorithm 2 describes the method for generating the change-of-basis matrix for the orthogonal from the triangular basis. Because this is not a linear transformation, Algorithm 3 describes the process for converting an element from orthogonal to triangular, and is followed by a discussion of the method for converting an element for triangular to orthogonal.

Algorithm 1 The triangular basis in terms of the orthogonal basis

Input: The orthogonal basis $e_1, e_2, \ldots, e_{2^r}$ and a set of m indexes $F \subseteq \{1, 2, \ldots, 2^r\}$.

Output: The triangular basis $t_1, t_2, \ldots, t_{2^{\hat{r}}+m}$ in terms of the orthogonal basis.

```
$k = 1$.                                   ▷ To index over the triangular basis elements.
for $i = 1$ to $2^r + m$ do
    $t_k = \sum_{j=1}^{i} e_i$.
    $k$+=1.
    if  $i \in F$ then                     ▷ Repeat rows listed in $F$.
        $t_k = t_{k-1}$
        $k$+=1.
    end if
end for
```

Recall that the triangular basis, without representation, is given in terms of the orthogonal basis by $t_j = \sum_{i=1}^{m} e_k$. As this is a linear transformation, its inverse is given by $e_1 = t_1$ and

$$e_j = t_j - t_{j-1}$$

for all $2 \leq j \leq 2^r + m$. If row j is repeated then e_j can be represented as

$$e_j = t_j - t_{j-1}$$

$$e_j = t_{j+1} - t_{j-1}$$

and if t_{j-1} is a repetition of the row t_{j-2}, then e_j can additionally be represented by

$$e_j = t_j - t_{j-2}$$
$$e_j = t_{j+1} - t_{j-2}.$$

Each orthogonal element can be represented in one, two, or four ways in terms of the triangular basis depending on the indexes of repeated rows. Because the transformation in Algorithm 1 is not invertible, Algorithm 2 operates by providing four possible representations of each orthogonal generator $e_j = e_{j,1} = e_{j,2} = e_{j,3} = e_{j,4}$ in terms of triangular generators, some of which may be duplicates.

Example 4.3.1. *Consider the set of orthogonal generators* $\{e_1, e_2, e_3, e_4\}$. *Let* $F = \{2, 3\}$ *be the set of indexes to generate the repeated rows in the triangular basis. Then, the triangular basis is generated by Algorithm 2 as*

$$t_1 = e_1$$
$$t_2 = e_1 + e_2$$
$$t_3 = e_1 + e_2$$
$$t_4 = e_1 + e_2 + e_3$$
$$t_5 = e_1 + e_2 + e_3$$
$$t_6 = e_1 + e_2 + e_3 + e_4.$$

Algorithm 2 The orthogonal basis in terms of the triangular basis

Input: The triangular basis $t_1, t_2, \ldots, t_{2^r+m}$, the set of m indexes $F \subseteq \{1, 2, \ldots, 2^r\}$.

Output: The orthogonal basis $e_{1,1}, e_{1,2}, e_{1,3}, e_{1,4}, e_{2,1}, e_{2,2}, e_{2,3}, e_{2,4} \ldots, e_{2^r,1}, e_{2^r,2}, e_{2^r,3}, e_{2^r,4}$ in terms of the triangular basis where $e_{i,j} = e_i$.

```
$k = 1.$                                          ▷ To index over the orthogonal basis elements.
while $i <= 4 \cdot 2^r$ do
    if $i \in F$ then                              ▷ If the currently indexed row is repeated.
        $e_{i,1} = t_k$, $e_{i,2} = t_k$, $e_{i,3} = t_{k+1}$, $e_{i,4} = t_{k+1}$
    else
        $e_{i,1} = t_k$, $e_{i,2} = t_k$, $e_{i,3} = t_k$, $e_{i,4} = t_k$
    end if
    if $i = 1$ then
        continue
    end if
    if $i - 1 \in F$ then                          ▷ If the previously indexed row is repeated.
        $e_{i,1}$-=$t_{k-1}$, $e_{i,2}$-=$t_{k-2}$, $e_{i,3}$-=$t_{k-1}$, $e_{i,4}$-=$t_{k-2}$.
    else
        $e_{i,1}$-=$t_{k-1}$, $e_{i,2}$-=$t_{k-1}$, $e_{i,3}$-=$t_{k-1}$, $e_{i,4}$-=$t_{k-1}$.
    end if
end while
```

Example 4.3.2. *The orthogonal basis is generated from the triangular basis in Example 4.3.1 via Algorithm 2 as*

$$\begin{array}{llll}
e_{1,1} = t_1, & e_{1,2} = t_1, & e_{1,3} = t_1, & e_{1,4} = t_1 \\
e_{2,1} = t_2 - t_1, & e_{2,2} = t_2 - t_1, & e_{2,3} = t_3 - t_1, & e_{2,4} = t_3 - t_1 \\
e_{3,1} = t_4 - t_3, & e_{3,2} = t_4 - t_2, & e_{3,3} = t_5 - t_3, & e_{3,4} = t_5 - t_2 \\
e_{4,1} = t_6 - t_5, & e_{4,2} = t_6 - t_4, & e_{4,3} = t_6 - t_5, & e_{4,4} = t_6 - t_4
\end{array}$$

In Algorithm 3, the ring elements in the orthogonal basis are converted to the triangular basis by randomly selecting one of the four possible representations of that element e_i in the triangular basis as $e_{i,1}$, $e_{i,2}$, $e_{i,3}$, or $e_{i,4}$.

Algorithm 3 Converting an element from orthogonal to triangular

Input: A ring element in the orthogonal basis, $a = a_1 e_1 + a_2 e_2 + \cdots + a_{2^{\hat{r}}} e_{2^{\hat{r}}}$, as well as the triangular in terms of orthogonal basis of Algorithm 1.

Output: The corresponding word in the triangular basis.

```
for $i$ from 1 to $2^{\hat{r}}$ do
    Select $e_i = e_{i,j}$ for random $j \in \{1, 2, 3, 4\}$.
    Replace $e_i$ with $e_{i,j}$ in $a$.
end for
Simplify by combining like terms.
```

The algorithm for conversion from the triangular basis to the orthogonal basis is straightforward and proceeds by replacing each triangular element t_i with its representation in the orthogonal basis as generated in Algorithm 2.

4.4 Homomorphic Evaluation

Assume there are r generators in the ciphertext ring. Then, there are 2^r elements of the ring in the orthogonal basis. After the triangular transformation, with repetition of m rows, the size of the published ring is given by $\hat{r} = 2^r + m$.

Assume there are two ring elements $a, b \in C$ to be added, where $a = a_1 t_1 + \cdots + a_{\hat{r}} t_{\hat{r}}$ and $b = b_1 t_1 + \cdots + b_{\hat{r}} t_{\hat{r}}$. Addition occurs component-wise and is carried out as in Algorithm 4.

Algorithm 4 Homomorphic Addition in the Triangular Basis

Input: $a = a_1 t_1 + \cdots + a_r t_{\hat{r}}$, $b = b_1 t_1 + \cdots + b_{\hat{r}} t_{\hat{r}}$, p

Output: $c = a + b = c_1 t_1 + \cdots + c_{\hat{r}} t_{\hat{r}}$

```
for $i = 1$ to $\hat{r}$ do
    $c_i = a_i + b_i \pmod{p}$
end for
```

Next, the values $a, b \in C$ are to be multiplied. To compute this product use the identity $t_i t_j = t_j$ whenever $i \leq j$. This product can be expressed in terms of the triangular basis as follows:

$$a \cdot b = \left(\sum_{i=1}^{\hat{r}} a_i t_i \right) \left(\sum_{i=1}^{\hat{r}} b_i t_i \right) = \sum_{i=1}^{\hat{r}} \left(\sum_{j=1}^{i-1} (a_i b_j + a_j b_i) + a_i b_i \right) t_i.$$

To show correctness, start by expanding the center term.

$$\begin{aligned} \left(\sum_{i=1}^{\hat{r}} a_i t_i \right) \left(\sum_{i=1}^{\hat{r}} b_i t_i \right) &= a_1 b_1 t_1 + a_1 b_2 t_2 + a_1 b_3 t_3 + \cdots + a_1 b_{\hat{r}} t_{\hat{r}} \\ &+ a_2 b_1 t_2 + a_2 b_2 t_2 + a_2 b_3 t_3 + \cdots + a_2 b_{\hat{r}} t_{\hat{r}} \\ &+ \cdots \\ &+ a_{\hat{r}} b_1 t_{\hat{r}} + a_{\hat{r}} b_2 t_{\hat{r}} + a_{\hat{r}} b_3 t_{\hat{r}} + \cdots + a_{\hat{r}} b_{\hat{r}} t_{\hat{r}}. \end{aligned}$$

This product may be rewritten by grouping over t_i. The coefficient terms may then be reordered for a new expression of the product as a sum.

Grouping terms over t_i yields

$$\begin{aligned}
&= a_1b_1t_1 + (a_1b_2 + a_2b_1 + a_2b_2)t_2 \\
&\quad + (a_1b_3 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3)t_3 + \cdots \\
&\quad + (a_1b_{\hat{r}} + a_2b_{\hat{r}} + \cdots + a_{\hat{r}-1}b_{\hat{r}} + a_{\hat{r}}b_1 + a_{\hat{r}}b_2 + \cdots + a_{\hat{r}}b_{\hat{r}})t_{\hat{r}} \\
&= a_1b_1t_1 + (a_1b_2 + a_2b_1 + a_2b_2)t_2 \\
&\quad + (a_1b_3 + a_3b_1 + a_2b_3 + a_3b_2 + a_3b_3)t_3 + \cdots \\
&\quad + (a_1b_{\hat{r}} + a_{\hat{r}}b_1 + a_2b_{\hat{r}} + a_{\hat{r}}b_2 \cdots + a_{\hat{r}-1}b_{\hat{r}} + a_{\hat{r}}b_{\hat{r}-1} + a_{\hat{r}}b_{\hat{r}})t_{\hat{r}} \\
&= \sum_{i=1}^{\hat{r}} \left[\left(\sum_{j=1}^{i-1} (a_ib_j + a_jb_i) \right) + a_ib_i \right] t_i.
\end{aligned}$$

With this expression of the product, one may implement multiplication via Protocol 5.

Algorithm 5 Homomorphic Multiplication in the Triangular Basis

Input: $a = a_1t_1 + \cdots + a_rt_{\hat{r}}$, $b = b_1t_1 + \cdots + b_{\hat{r}}t_{\hat{r}}$, p

Output: $c = a \cdot b = c_1t_1 + \cdots + c_{\hat{r}}t_{\hat{r}}$

```
for $i = 1$ to $\hat{r}$ do
    $c_i = 1 \cdot a_i \cdot b_i$
    for $j = 1$ to $i - 1$ do
        $c_i$ += $a_i \cdot b_j + a_j \cdot b_i \pmod{p}$
    end for
end for
```

4.5 Computational Complexity

The complexity of the algorithms used for implementation of GKS will have an effect on its performance speed. Algorithm 4 carries out addition component-wise in $\hat{r}$ steps, where $\hat{r}$ is the number of generators in the published ciphertext ring. Therefore, addition occurs with a complexity of $\Theta(\hat{r})$.

Multiplication, seen in Algorithm 5, occurs in a nested loop. Line 2 of this algorithm is a step of complexity $\mathcal{O}(1)$ which occurs in the outer loop. The inner loop contains one step of complexity $\mathcal{O}(1)$, and executes $i-1$ times during iteration i of the outer loop. Therefore, the complexity of Algorithm 5 is derived by

$$\sum_{i=1}^{\hat{r}} (1 + (i-1)) = \frac{\hat{r}(\hat{r}+1)}{2}$$

and yields a complexity of $\mathcal{O}(\hat{r}^2)$.

4.6 Generating Encryptions of Zero

Parameters for the GKS scheme must be selected in order to avoid overflow over the prime modulus p during computation. With this in mind a second parameter, $q \in \mathbb{N}$, must be introduced, where $q \ll p$. During generation of random noise within the encryption algorithm, the random values chosen as coefficients to ring elements should be selected to lie in the range $[0, q)$. The precise size of the variable q is application-dependent and should be selected so that overflow over the prime modulus p does not occur during homomorphic multiplication.

	KeyGen	**Encrypt**	**Decrypt**	**Add**	**Multiply**
32-bit modulus	1.3186	0.0327	0.0585	0.0001	0.0229
64-bit modulus	1.3157	0.0337	0.0616	0.0001	0.0299
128-bit modulus	1.3905	0.0411	0.0665	0.0001	0.0311
256-bit modulus	1.5719	0.0448	0.0738	0.0001	0.0346
512-bit modulus	1.6333	0.0450	0.0758	0.0002	0.0361
1024-bit modulus	1.6373	0.0455	0.0763	0.0002	0.0402

Table 4.1: Average implementation times for the GKS cryptosystem in seconds

4.7 Experimental Performance

Experiments were run to determine average computation speed via GKS using the default parameters combined with the implementation recommendations in this chapter. Operations were carried out over randomly generated values. The times reported in Table 4.1 were generated by repeating the protocol $1,000$ times and calculating the average run time.

Values for random noise generation were uniformly selected from the range $[0, 2^{n-1})$ for each n-bit prime. This is higher than what will generally need to be used, and therefore provides a safe over-estimate of the needed computation time. Tests were run on a Macbook Pro with a 2.3 Ghz processor and 16.0 GB memory using C++ and the GNU MP Bignum Library [62] for arbitrary precision arithmetic. Table 4.1 shows the experimental results for these algorithms.

Figures 4.2 through 4.6 contains several sub-figures which show the bit size of the prime modulus on the horizontal axis and time, in seconds, on the vertical axis. Figure 4.2 shows that key generation is the most costly step, and larger prime modulus values correspond to larger key generation times. However, even with a 1024-bit prime modulus, the average key generation time is below 1.5 seconds. Homomorphic addition, shown in Figure 4.4, is a very fast operation regardless of prime modulus size and should not represent a bottleneck during computation. Figure 4.5 shows homomorphic multiplication, which takes longer than homomorphic addition with an average computation time of between 0.02 and 0.03 seconds. When many multiplication operations are needed, this could pose as a bottleneck to computation.

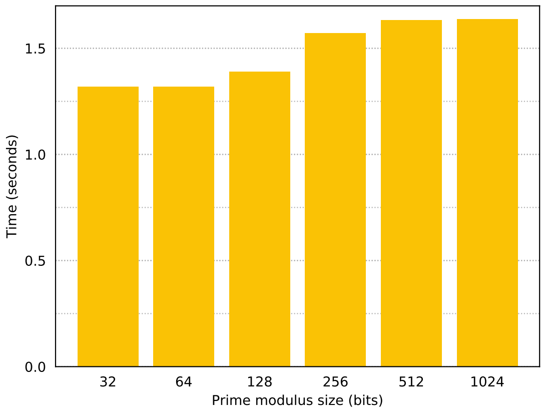


Figure 4.2: Key Generation

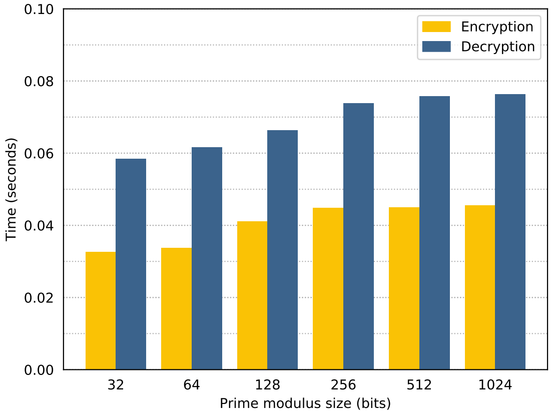


Figure 4.3: Encryption and Decryption

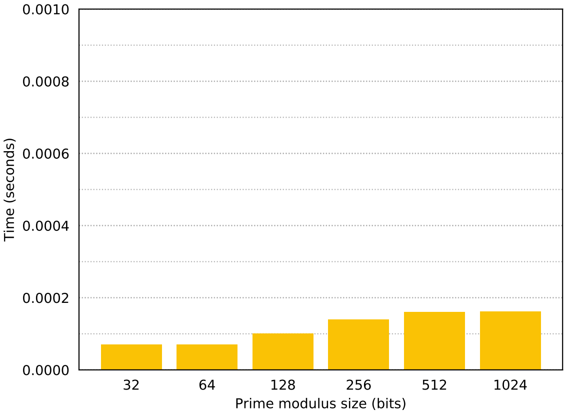


Figure 4.4: Homomorphic Addition

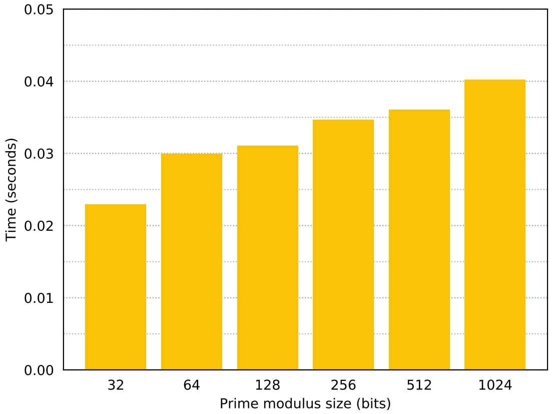


Figure 4.5: Homomorphic Muliplication

Figure 4.6: Average Computation Time in GKS

Figure 4.3 shows the encryption and decryption times for plaintext and ciphertext values, respectively. The plaintext values in these experiments were randomly generated by populating the coefficients in the plaintext ring with values in the range $[0, 2^{n-1})$. Encryption is a more computationally intensive process, as it involves randomization steps and generation of random noise. Decryption consists of a series of substitutions using the private key and the time required remains relatively constant with varying modulus sizes. While encryption and decryption of a value takes longer than addition or multiplication, it should not pose a bottleneck during computation within the proposed setting. Specifically, the algorithms proposed and implemented in this document require a database to be encrypted and stored once *prior* to classification. Decryption is called only a smaller, fixed number of times during these protocols.

4.7.1 Performance Comparison

ML Confidential [35] implements a leveled homomorpic encryption scheme using a modified version of a scheme presented by Brakerski [7]. With a 128-bit prime modulus, the authors report computation times as seen in Table 4.2. While key generation runs faster in this protocol than in the implementation of GKS-FHE in Table 4.1, all other algorithms run more quickly in GKS-FHE. In particular, multiplication takes place in 0.026 seconds in GKS-FHE with a 128-bit prime modulus compared to 0.853 seconds in ML Confidential, and addition takes places in 0.0001 seconds. Key generation is typically run only once per protocol, while addition and multiplication may need to be run many times.

Scheme	**KeyGen**	**Encrypt**	**Decrypt**	**Add**	**Multiply**
ML Confidential [35]	0.279	0.659	0.055	0.001	0.853
SHIELD [41]	0.27	0.383	0.3	0.006	0.372
HElib [59, 41]	85.3	0.59	0.39	0.002	3.6

Table 4.2: Average implementation of existing FHE schemes in seconds

Halevi and Shoup's HElib [59], associated with IBM Research, implements the BGV leveled fully homomorphic encryption scheme [11] with the Smart-Vercauteren SIMD ciphertext packing techniques [60] and additional Gentry-Halevi-Smart optimizations [31]. Authors provide preliminary multiplication times ranging from 25.7 seconds to 473 seconds based on varying sizes of key generation parameters [38]. A March 2018 update to the project's GitHub repository claims forthcoming speedups, which make implementation 15 to 75 times faster.

The implementation times for HElib displayed in Table 4.2 are from an implementation of HElib speed experiments carried out by Khedr et al. in 2015 [41]. The authors in this work also propose an improvement on the Gentry-Sahai-Waters (GSW) scheme [32] within a framework they title Secure Homomorphic Implementation of Encrypted Data-Classifiers (SHIELD). The authors' reported implementation speed data for this scheme is also presented in Table 4.2. These experiments were run on a CPU with 4 cores, 8 threads, and 32 GB of memory. Compared to both of these schemes, GKS encryption performs encryption, decryption, homomorphic addition, and homomorphic multiplication significantly faster. Key generation in SHIELD is faster than in GKS, but significantly slower in HElib. As mentioned before, key generation is performed only once, and fast homomorphic addition and multiplication times are key to avoiding computational bottlenecks in real-world applications.

The most recent version of Microsoft's Simple Encrypted Arithmetic Library (SEAL) [43] implements the "FullRNS" variant [4] of the Fan-Vercauteren somewhat homomorphic scheme [25]. Microsoft reports that encoding and encryption of 4096 25 × 25 pixel gray scale images within a single ciphertext occurs in an average of 44.5 seconds, and decryption of these ciphertexts occurs in 3 seconds [33]. This setting is not directly comparable to the experiments on GKS-FHE outlined in table 4.1. Another application of SEAL reports computation time in SEAL as 0.002 milliseconds for homomorphic addition and 1.514 milliseconds for homomorphic multiplication [3]. Again, this scenario is not directly comparable to the results provided in Table 4.1, as their experimental setup implemented Microsoft Azure's powerful data centers. Specifically, they ran their tests on Azure's H16 instances with 16-core 3.6 GHz processors with 112 GB of memory.

The utility of GKS can be seen in the computational complexity of the multiplication operation. Homomorphic multiplication in GKS corresponds to polynomial multiplication.

Other ring-based schemes that include noise growth, such as FV [25] and SEAL [43], require both a multiplication step and a "relinearization" step. While the multiplication step in schemes of this form may at best correspond to polynomial multiplication, the relinearization step adds additional complexity to the homomorphic multiplication operations.

4.8 Conclusion

All in all, the GKS fully homomorphic encryption scheme can be implemented efficiently for operations over real-world data. Computation speeds for key generation, encryption, decryption, addition, and multiplication are competitive with the state-of-the-art implementations of public-key FHE. The lack of noise during homomorphic operations in the GKS scheme marks an advantage over leveled and somewhat homomorphic schemes.

Furthermore, SIMD may be implemented in GKS for parallelization of computation. Up to 2^n distinct values may be packed within a single GKS plaintext in a ring with n generators while preserving homomorphic addition and multiplication operations. Real values and floating point values are able to be encoded within the scheme.

Chapter 5

Privacy-Preserving Naive Bayes Classification

A well-known machine learning classifier is Naive Bayes, a model which classifies a new data point using probabilities computed during a training phase. Naive Bayes earns its "naive" title due to the strong independence assumptions it makes between the features of the data. This assumption yields a transparent and understandable method for classification which often outperforms more sophisticated methods [39]. Similarly, when diagnosing a patient, clinicians try to define conditionally independent attributes which could be indicative of a disease [40]. Naive Bayes has outperformed more sophisticated methods and state-of-the-art diagnosis systems on 5 out of 8 real-life medical data sets, including localization of a primary tumor, prediction of recurrence of breast cancer, and rheumatological diagnosis [42], as well as in applications predicting heart disease [40].

Naive Bayes is based on *Bayes Theorem.* Given a set of classes G to which a sample X could be assigned, Bayes Theorem states that

$$\Pr(G = G_i | X) = \frac{\Pr(X | G = G_i) \Pr(G = G_i)}{\Pr(X)}.$$

In this formula, $\Pr(G = G_i|X)$ represents the *posterior probability* of a class given an attribute, whereas $\Pr(X|G = G_i)$ is the likelihood of an attribute given a class. Thus the posterior probability is computed using prior knowledge and observed data.

The Naive Bayes model assumes that the features of the data points are conditionally independent in each class. For a class $G = G_i$ and a feature space of dimension d with $X = (X_1, \ldots, X_d)$, the Naive Bayes model assumes that

$$P(X|G = G_i) = \prod_{j=1}^{d} P(X_j|G = G_i). \tag{5.1}$$

This does not often hold in the real world, hence the “naive” title.

Despite this, Naive Bayes is able to provide fast results which are often better than more sophisticated methods when the dimension of the feature space is large. In these situations, assuming that the features of the space are independent can help avoid clumsy density estimations. The Naive Bayes classifier does not consider correlation among features, thus lowering the variance while increasing the bias and hence avoiding over fitting on the training set. Due to this, it has become a benchmark used by the community in data analysis. Furthermore, when diagnosing a patient, clinicians try to define conditionally independent attributes which could be indicative of a disease [40]. Naive Bayes has outperformed more sophisticated methods and state-of-the-art diagnosis systems on 5 out of 8 real-life medical data sets, including localization of a primary tumor, prediction of recurrence of breast cancer, and rheumatological diagnosis [42], as well as in applications predicting heart disease [40].

5.1 Naive Bayes Classification

The classification of a vector X is given simply by taking the maximum posterior probability over all classes and applying Bayes' Theorem.

$$\begin{aligned}\hat{G} &= \operatorname*{argmax}_{G_i \in G} \Pr(G = G_i | X) \\ &= \operatorname*{argmax}_{G_i \in G} \frac{\Pr(X | G = G_i) \Pr(G = G_i)}{\Pr(X)} \\ &= \operatorname*{argmax}_{G_i \in G} \Pr(X | G = G_i) \Pr(G = G_i) \\ &= \operatorname*{argmax}_{G_i \in G} \Pr(G = G_i) \prod_{j=1}^{d} \Pr(X = X_j | G = G_i).\end{aligned}$$

Note that the denominator term $\Pr(X)$ can be removed because X is fixed, while the last step follows from the independence assumption in equation 5.1.

Commonly, in medical applications each attribute X_j of a vector X can take on only a discrete set of values. Then, all of the information needed to classify an arbitrary data point can be succinctly represented in a collection of vectors and matrices. This representation will enable us to collect all data needed in order to perform a classification. The ease with which all this data can be represented will be of great use in the protocol which follows. Let $P = (P_i)$ be a vector where

$$P_i = \Pr(G = G_i)$$

and let T be a matrix where the (i, j)th entry $T_{i,j}$ is equal to

$$T_{i,j} = \Pr(X_j | G = G_i).$$

Given tables P and T, any new data point can be classified by looking up the probability for each attribute given each class and computing the argmax of the resulting values.

Bost et al. implement private Naive Bayes classification between a Client and a Service Provider [6]. The Client has a single data vector X which he would like to keep private. The service provider owns a private model w which she would like to keep private. Given a classification algorithm C the client should be able to learn $C(X, w)$, the classification of his vector X using the model w, without learning any partial information about the model or giving away any information about his input.

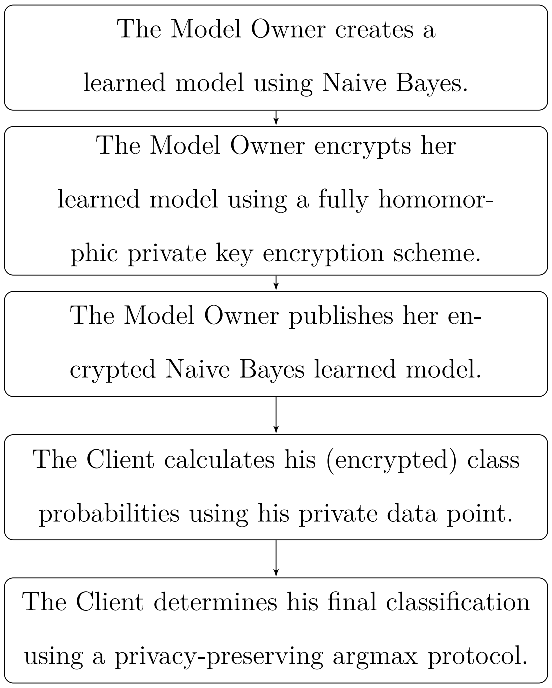

The Model Owner creates a learned model using Naive Bayes.
The Model Owner encrypts her learned model using a fully homomorphic private key encryption scheme.
The Model Owner publishes her encrypted Naive Bayes learned model.
The Client calculates his (encrypted) class probabilities using his private data point.
The Client determines his final classification using a privacy-preserving argmax protocol.

Figure 5.1: Private Naive Bayes Classification

Let pk_Q, sk_Q denote a public and secret key pair in the QR scheme and pk_P, sk_P denote the same in Paillier. Let square brackets $[\![a]\!]_{\mathcal{P}}$ denote the encryption of a value a under Paillier. The security of these schemes provides semantic security for the authors' protocols, and an honest-but-curious adversary model is used.

Assume each vector X has d features, each of which can take on a finite number of possible values, and that there are c possible classes. The authors' protocol runs as follows:

1: The Service Provider encrypts the tables P and T using Paillier.

2: The Server sends the encrypted tables to the Client.

3: The Client computes $[\![p_i]\!]_{\mathcal{P}} = [\![P_i]\!]_{\mathcal{P}} \prod_{j=1}^{d} [\![T_{i,j}(X_j)]\!]_{\mathcal{P}}$ for $i = 1, \ldots, c$.

4: The Client uses the server to compute $i = \text{argmax}_i p_i$.

5: Client outputs i.

Note that Step 3 of the above protocol requires the encryption scheme which is multiplicatively homomorphic. To get around this requirement, Bost et al. implement the model using the logarithm of the probability distributions, where $p_i = \log(\Pr(G = G_i|X))$. Therefore, the posterior probabilities in Step 3 are computed using Paillier's additively homomorphic property. Computing the argmax in Step 4 requires an additively homomorphic scheme. Bost et al. use Paillier and QR, combined with an algorithm for changing the encryption scheme, in order to compute the argmax. In the next section, we describe our adapted version of the protocol which performs classification using a single encryption scheme.

5.2 Proposed Method for Fully Homomorphic Naive Bayes Classification

The protocol presented below varies in several important ways from the previous protocol. Use of a fully homomorphic scheme adds flexibility to the computation by allowing the posterior probabilities to be computed using homomorphic multiplication or in the logarithmic model using homomorphic addition. Furthermore, the presented model assumes direct communication between the Model Owner and the Client. In other words, there is not a trusted server acting as intermediary between the parties. The protocol could easily be adapted to include a Server to carry out computations, if desired.

The protocol is designed as follows. Assume that a Client wishes to classify his vector X which contains d features based off of a learned model w owned by the Model Owner. The group of classes G contains c distinct classes, $G_1, \ldots, G_c$. During this protocol the Model Owner should learn no unnecessary information about the input provided by the Client, and the Client should learn nothing but the predicted class index of X.

There is a certain amount of information which the Client *must* know in order to carry out the protocol. Namely, the Client must know that the data vector X has d features and that there are exactly c classes. However, he should not be able to deduce any information about the conditional class probabilities associated with the d features or the c class probabilities.

Because the Model Owner has already computed a learned model, she prepares two tables. First, the Model Owner prepares table P represented as a column vector of degree c where $P_i = \Pr(G = G_i)$, the prior probability on class G_i. Next she prepares a table T, where entry T_{ij} represents $\Pr(X_j|G = G_i)$. The protocol is given in Algorithm 6.

Algorithm 6 Fully Homomorphic Private Naive Bayes Classifier

Client Input: A data point X.

Model Owner Input: A learned Naive Bayes model with tables P and T of prior class probabilities and likelihoods, respectively.

Client Output: Classification of the data point X under the Model Owner's model.

1: The Model Owner prepares the tables P and T and sends their encryption, $[\![P]\!]$ and $[\![T]\!]$, to the Client.

2: For each class G_i for i from 1 to c, the Client is able to compute

$$\begin{aligned}[\![Pr(G_i|X)]\!] &= [\![Pr(G_i)]\!] \cdot \prod_{j=1}^{d} [\![Pr(X_j|G_i)]\!] \\ &= [\![P_i]\!] \cdot \prod_{j=1}^{d} [\![T_{ij}]\!] \\ &= [\![P_i \cdot \prod_{j=1}^{d} T_{ij}]\!].\end{aligned}$$

3: The Client computes

$$i = \operatorname*{argmax}_{1 \le i \le c} [\![Pr(G_i|X)]\!]$$

using the private argmax protocol described below in Algorithm 8.

If the model was designed using logarithms of the probabilities, then the multiplication operations in Step 2 should be replaced with addition.

There are two parties to consider when discussing the security of this protocol: the privacy of the Client's information as well as the privacy of the Model Owner's learned model. The privacy of the learned model is derived entirely from the security of the encryption scheme used. Discussion of the privacy of the Data Owner's information, however, requires knowledge of the argmax protocol called in Step 3. This argmax protocol is outlined in Section 5.2.1, and the overall security of the protocol is discussed in Section 5.3.

5.2.1 Privacy-Preserving Argmax Protocol

Attempts at the argmax protocol that contain security leaks are first described in order to build intuition for working up to a secure framework. Denote $\mathcal{E}_i = [\![\Pr(G_i|X)]\!]$, and the set of all $\mathcal{E}_i$ as $\mathcal{E}$.

First, suppose the Client sends the set of encrypted probability values $\mathcal{E}$ to the Model Owner. Then, the Model Owner decrypts each value and sends the Client the index of the highest value using asymmetric encryption. While the Client has learned nothing about the model, in this scenario the Model Owner learns not only which class the Client's data point belongs to but also the exact probabilities for each class.

Suppose instead that the Client performs a permutation π on the class probabilities, then sends $\pi(\mathcal{E})$ to the Model Owner. Then, the Model Owner decrypts the values, determines which is largest, and sends that index to the Client, who reverses the permutation to determine his class. It is not necessary to hide the value of the index sent to the Client from any eavesdroppers in this scenario, as the permutation π randomizes the indexes. However, while this method prevents the Model Owner from determining trivially which probability is associated with each class, it does not prevent her from learning the class probabilities themselves.

Another naive attempt at argmax is considered before presenting the secure protocol. Algorithm 7 breaks down the computation of argmax into a series of comparisons.

Algorithm 7 A naive attempt at private argmax

Client Input: A set of indexes $I = \{1, 2, \ldots, c\}$, a family of additively homomorphic monotone functions $\mathcal{F}$, and a set of c values encrypted under the Model Owner's private key.

Client Output: The argmax of the input values, or index of the largest of the c input encrypted values.

```
Set $I = \{1, 2, \ldots c\}$.
while $|I| > 1$ do
    The Client computes a random permutation $\pi$ on $I$.
    The Client computes $\mathcal{E}^* = \mathcal{E}_{\pi(1)} - \mathcal{E}_{\pi(2)}$. Because the encryption scheme is additively homomorphic,
        $$\mathcal{E}_{\pi(1)} - \mathcal{E}_{\pi(2)} = [\![P_{\pi(1)} - P_{\pi(2)}]\!].$$
    The Client sends $\mathcal{E}^*$ to the Model Owner.
    The Model Owner decrypts $\mathcal{E}^*$ and recovers
        $$\begin{aligned} p &= P_{\pi(1)} - P_{\pi(2)} \\ &= \Pr(G_{\pi(1)}|X) - \Pr(G_{\pi(2)}|X). \end{aligned}$$
    If this value is negative, the Model Owner sends the bit $b = 0$ to the Client, otherwise send $b = 1$.
    If $b = 0$, the Client removes $\pi(1)$ from $I$. Otherwise he removes $\pi(2)$.
end while
The Client returns $I$.
```

Because the value computed by the Client is random, the bit 0 or 1 will appear random to any observer. However, the Model Owner recovers some partial information about the Client's data. Namely, she recovers a collection of $c - 1$ values representing the difference between pairs of the posterior probabilities for different $i \in \{1, 2, \ldots, c\}$. While the permutation keeps the Model Owner from knowing which pairs of elements correspond to which values, it is not outside the realm of possibility that an attack on the Client's private data could be carried out by her using this information.

The following problem remains: The Client needs to compare two values encrypted under the Model Owner's private key. Previous approaches to this problem focus on public-key encryption, where the Client can mask his value with random noise encrypted under the Model Owner's public key. However, this approach will not work with private-key encryption because there is no way for the Client to encrypt random noise.

The proposed method for comparison in the private-key setting uses the fully homomorphic properties of the encryption scheme. Using the homomorphic properties of encryption, a family $\mathcal{F}$ of additively homomorphic, *monotone functions* that commute with encryption is constructed. These functions are used to randomize the values sent to the Model Owner. Say that a function $f : R \to R$ *commutes* with encryption if

$$f\left(\llbracket m \rrbracket\right) = \llbracket f(m) \rrbracket.$$

Furthermore, the additively homomorphic property of f guarantees that $f(m+n) = f(m) + f(n)$.

The outline of the protocol that uses additively homomorphic encryption and a monotone function to return the argmax is given in Algorithm 8. Possible families of additively homomorphic monotone functions that can be used with the GKS encryption system are then discussed in Section 5.4.

Algorithm 8 Privacy-preserving argmax

Client Input: A set of indexes $I = \{1, 2, \ldots, c\}$, a family of additively homomorphic monotone functions $\mathcal{F}$, and a set of c values encrypted under the Model Owner's private key.

Client Output: The argmax of the input values, or index of the largest of the c input encrypted values.

```
while $|I| > 1$ do
    The Client computes a random permutation $\pi$ on $I$.
    The Client randomly chooses $f \leftarrow \mathcal{F}$ and computes the values $f\left(\mathcal{E}_{\pi(1)}\right)$ and $f\left(\mathcal{E}_{\pi(2)}\right)$.
    The Client uses the additive homomorphic properties of the encryption scheme and
    $f$ as well as the commutative property of the function to evaluate
        $\mathcal{E}^* = f\left(\mathcal{E}_{\pi(1)}\right) - f\left(\mathcal{E}_{\pi(2)}\right)$
        $= f\left([\![P_{\pi(1)}]\!]\right) - f\left([\![P_{\pi(2)}]\!]\right)$
        $= [\![f(P_{\pi(1)} - P_{\pi(2)})]\!]$
    The Client sends $\mathcal{E}^*$ to the Model Owner.
    The Model Owner decrypts $\mathcal{E}^*$ and recovers
        $f(P_{\pi(1)} - P_{\pi(2)})$
    If this value is negative, the Model Owner sends the bit $b = 0$ to the Client, otherwise
    send $b = 1$.
    If $b = 0$, the Client removes $\pi(1)$ from $I$. Otherwise the Client removes $\pi(2)$.
end while
The Client returns $I$.
```

During this protocol, the Model Owner collects c values representing the result of a monotone function applied to the difference between random pairs of the posterior probabilities. The application of an unknown monotone function to this difference prevents the Model Owner from learning partial information about the Client's values.

5.3 Security

The proposed protocol is secure in the honest-but-curious setting. In this setting, all parties are assumed to take part in the protocol honestly. However, they will use any information they can obtain during an honest execution of the protocol in order to attempt to deduce further information. In this model, parties should not be able to learn any information beyond their protocol output.

During protocol execution, the Client only has access to encrypted values. The private argmax protocol allows the Client to determine which encrypted value is the largest out of a set of values, but does not give him access to the actual values. Therefore, the security of the Model Owner's data is reliant upon the security of the fully homomorphic encryption scheme that is implemented. The GKS scheme is secure against ciphertext-only attack [36].

The only information the Model Owner receives during protocol execution is obtained during the private argmax protocol. The Model Owner receives the value

$$f(P_{\pi(1)} - P_{\pi(2)})$$

during each elimination round of the argmax protocol. Because of the permutation π, she does not know which posterior probabilities are being compared, or, in the case of binary classification, in what order they are being compared. Furthermore, the function f, selected from a family $\mathcal{F}$ of additively homomorphic, monotone functions that commute with encryption, hides the exact value of the difference between the randomized posterior probabilities. Therefore, at the end of protocol execution, the Model Owner is unable to determine any ordering on the posterior probabilities the Client has computed, and cannot determine his final classification.

5.4 Implementation

The proposed protocol was run using the GKS encryption scheme, where coefficients in $\mathbb{Z}_p$ were taken from $(-p/2, p/2]$ and correspond to positive and negative integers, and

$$\mathcal{F} = \{f : R \to R : f(m) = km\}$$

for a randomly chosen integer k up to 20 bits. Implementation used a 198-bit prime p.

Experiments were implemented in C++ on a Mac Book Pro using El Capitan, a 2.3 GHz Intel Core i7, with 16 GB memory. The GNU Multiple Precision Library (GMP) [62] was implemented to allow for integer storage above the built-in data type limits in C++. An implementation of the Naive Bayes algorithm created a learned model. This learned model was encrypted using the GKS Encryption Scheme. Then both encrypted and unencrypted classification of data points, not included in the training set, was carried out. Section 5.5 describes the results of these experiments.

5.5 Evaluation

Data from the UCI Machine Learning Repository was used to test the performance of the protocols [45]. Specifically, the Breast Cancer Wisconsin (Original) Data Set was used, which contains 683 complete data points each containing an ID along with 9 attributes and a binary classification.

	Accuracy	**Sensitivity**	**Specificity**	**Precision**	**NPV**	**F1-score**
Mean	0.96007	0.93398	0.97414	0.95299	0.96546	0.94224
Stand. Dev.	0.02077	0.04861	0.02439	0.04255	0.02469	0.03026

Table 5.1: Naive Bayes Classification Results

The data gives measurements taken from fine-needle aspirate (FNA) biopsies of benign and malignant breast tumors. These nine attributes include clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. Each of these attributes was measured by a clinician on a scale of 1 to 10 at the time it was collected, with lower values corresponding to what you would expect to see in a benign case and higher values corresponding to what you would expect in a malignant case. Previous research has found that while each measurement holds clinical significance in diagnosing a breast tumor as benign or as malignant, a single attribute is not enough to distinguish between the two cases [67].

A Naive Bayes algorithm was implemented to create a learned model which was then encrypted using the GKS Encryption Scheme. Experiments evaluated the performance of the model using 10 by 10-fold cross validation. Furthermore, because the data set is unbalanced with a higher proportion of benign tumors, random oversampling of malignant tumors was implemented while training the models. A positive case denotes a case where the tumor is diagnosed as malignant and a negative case refers to a tumor which is benign.

Additive smoothing on the values in the tables of class and prior probabilities was implemented during both encrypted and unencrypted testing to prevent error in the case of zero or near zero probabilities. Specifically, each probability was increased by 0.1, and any value which was greater than or equal to 1 after smoothing was reset to 0.999. The size of the ciphertext ring in the experiments was $2^8 + 50 = 306$, and the prime modulus p was 198 bits.

Classification was tested on data points not included in the training set in both encrypted

	Mean Time
Unencrypted	0.00001
Encrypted	0.40298

Table 5.2: Mean Classification Time, In Seconds

and unencrypted formats. Tables 5.1 and 5.2 show a comparison of the average performance of the encrypted and unencrypted experiments under 10 by 10-fold cross validation. Results include classification time for a single data point in seconds, accuracy, sensitivity, specificity, precision, negative predictive value (NPV), and the F1 score. The model in the experiments was trained with five decimal points precision, and the unencrypted experimental results were obtained with five decimal points precision. In the encrypted experiments all values were initially encrypted with three decimal points precision. No change occurred in the reported statistics due to this change in decimal precision.

In the case of breast cancer specifically it is crucial to positively identify all malignant tumors for timely medical intervention. The results yield high precision and show higher sensitivity, meaning there is a low rate of false positives and an even lower rate of false negatives. Specificity describes the proportion of negative cases which were classified as negative and NPV describes the proportion of cases classified as negative which were negative in reality. The results show both high Specificity and high NPV, pointing to a high rate of true negatives and a low rate of false negatives. The experiments also yielded a high F_1 score, which takes into account both the precision and the recall of the classifier.

5.6 Discussion

The proposed method performs classification in an average of 0.40298 seconds, compared to 0.479 seconds per classification in the experiments of Bost et al. [6]. Private classification methods for more sophisticated classifiers are even more time consuming. Wu et al. perform private decision tree classification on a tree with 12 decision nodes in approximately 0.545 seconds [68]. Bost et al. report average decision tree classification times on a 4 node tree in approximately 2.085 seconds [6]. Rahulamathavan et al. perform private SVM classification on the same breast cancer data set in an average of 7.71 seconds [54].

The time increase between encrypted and unencrypted computation is expected and occurs in all current fully homomorphic encryption methods. For the example provided above, where a single user wishes to classify their data, classification in under half a second is within a reasonable time range for medical applications.

5.6.1 Computational Bottlenecks

The computational bottleneck within fully homomorphic encryption schemes occurs during homomorphic multiplication operations. Chapter 4 provides results which show that the GKS scheme is relatively efficient at performing fully homomorphic multiplication operations.

The amount of time a clinician may expect this protocol to take will depend upon the number of classes and the number of features within the data set being analyzed, as this determines the number of homomorphic multiplication operations which must be performed. Consider a data set with d features and c classes. For each data point classification, a total of $(d+1)$ homomorphic multiplication operations must be performed in order to calculate the posterior probability, calculated as

$$[\![Pr(G_i|X)]\!] = [\![Pr(G_i)]\!] \cdot \prod_{j=1}^{d} [\![Pr(X_j|G_i)]\!]$$

for each of c classes. This results in a total of $c(d+1)$ multiplication operations per classification.

Communication can also pose a bottleneck to computation, depending on the speed of the connection between the Client and Model Owner. The argmax protocol presented requires that the Model Owner communicates with the Client a total of $c - 1$ times, and that the Client communicates with the Model Owner a total of $c - 1$ times. In the common case of binary classification, this is only one communication from Model Owner to Client and vice versa.

5.6.2 Comparison to Other Classifiers

Naive Bayes often outperforms more sophisticated classification methods for medical diagnosis. Examples include prediction of heart disease [40] and rheumatological diagnosis [42]. Classification on the Wisconsin breast cancer data set using decision trees, support vector machine (SVM), k-nearest neighbors, and logistic regression classification were implemented in Python 3 under 10-fold cross validation for comparison of classifier performance [39].

Decision trees were limited to a maximum depth of 5, a minimum of 3 samples per split, and a minimum of 3 samples per leaf. Gini impurity was used to measure the quality of the split during training. Privacy-preserving decision tree classification is discussed in Chapter 7. Support vector machine (SVM) was implemented with a linear kernel. The k-nearest neighbors classifier was implemented using the ball tree algorithm with 5 neighbors, a leaf size of 30, and uniformly weighted points, using the Euclidean distance measure. The logistic regression model was built using the LIBLINEAR algorithm for optimization of the learning function [44].

Performance of the classifiers is given in Table 5.3. The highest score for each metric highlighted in red. Naive Bayes provided the best performance for the negative predicted value. While other classifiers outperform Naive Bayes on the other metrics, Naive Bayes remains competitive, especially when the relative speed of privacy-preserving Naive Bayes is considered.

	Accuracy	**Sensitivity**	**Specificity**	**Precision**	**NPV**
Naive Bayes	0.96003	0.93389	0.97410	0.95100	0.96476
Decision Tree	0.96628	0.96171	0.97500	0.98699	0.93487
Linear SVM	0.97069	0.96843	0.97500	0.98651	0.94500
***k*-Nearest Neighbors**	0.97065	0.96838	0.97482	0.98646	0.94544
Logistic regression	0.96920	0.97298	0.96232	0.98017	0.95276

Table 5.3: Comparison to other machine learning classifiers, with the highest score for each metric in red

5.7 Conclusions

This first experiment suggests that private-key fully homomorphic encryption can classify medical data efficiently. Similar techniques could enable researchers and clinicians to utilize private medical data and models which they cannot access in the clear. Hospitals and companies with trained assisted diagnosis systems could use this technology to provide access to their models without giving away their parameters, and doctors can use this software without revealing their patient's information. Medical researchers can use these methods to determine whether their models are over fit to their own data sets.

In the next chapter, a protocol for third-party private search is introduced. This protocol is implemented in Chapter 7 for classification via decision trees, further showing the utility of private-key FHE for classifying medical data.

Chapter 6

Third Party Private Search via Fully Homomorphic Encryption

6.1 Introduction

Private search describes a variety of multi-party protocols which involve the querying of encrypted or otherwise privatized data. Private search models follow multiple paradigms. *Data outsourcing* describes the model where a data owner would like to store their database on a server in encrypted form, and wants to search it later themselves. This model does not seek to hide the database contents from the querying party. In the *data sharing* model, also known as *Privacy-Preserving Sharing of Sensitive Information (PPSSI)* [19], *Secure Anonymous Database Search (SADS)* [55] and *Private Information Retrieval (PIR)* [15], a data owner would like to allow a client to search their database without revealing their data. This model is less common seeks to hide the database contents from the querying party [49]. Different contexts require different query outcomes. For instance, a private search protocol can return the count of the number of times a value appears in a database, related content to the query within the database, or a Boolean denoting the presence of a value in the database. The security guarantee needed in the context will vary based on the participants in the scenario and the desired outcome of the protocol.

Private search has a variety of real-world applications. The data outsourcing model can be used by law enforcement for cyber security, allowing an institution to store all sensitive data in an encrypted form while allowing investigators to query the information [14]. If this institution is reluctant to divulge the contents of their search, the data sharing model would enable an investigator to perform a private query on an encrypted database. Similarly, private search has applications in the realm of medical data, where legal restrictions such as HIPAA combined with the sensitive identifying nature of the data make data privacy a priority. For example, private search would enable a medical institution to let an outside researcher query the medical information database for presence of some feature in order to determine if they would like to proceed with the time-consuming process of applying to an institutional review board (IRB) for access to that data. Private search also has application in private medical information system protocols. An application of private search within a Decision Tree classifier is provided in the next chapter, Chapter 7.

Raykova et al. implement what they call exact keyword match in the secure anonymous database search (SADS) setting. Their research focuses on searching for keywords within documents. Therefore, their method has a strong focus on feature extraction and natural language processing in order remove exact match requirements such as capitalization restrictions. The method requires the setup of a *Bloom filter* for each document in the database, which is built from the encryptions of all words in the document. These Bloom filters are then used in conjunction with encryption methods in order to perform private keyword search within the documents. The authors propose a variety of decision function and feature extraction combination, each of which has varying success at locating all documents containing a phrase and adaptable false positive and false negative thresholds. Databases feature extraction time and query time are not discussed explicitly; query time is described as efficient.

Pappas et al. implement querying of protected data with partially trusted parties. Their work focuses on secure document retrieval, and in the process implements an improvement on the SADS system discussed in the previous paragraph. Their model sacrifices some privacy for increased efficiency. Specifically, the authors allow partially trusted parties. Angel et al. use fully homomorphic encryption in the private information retrieval model in order to allow a Client to download a file from a Server without the Server knowing which file they downloaded [3].

This chapter describes a method performing the data sharing method of *third party private search* with a binary outcome denoting presence of an exact match. In this scenario, a Database Owner has a private database which a Client wishes to privately query. Two scenarios are examined. In the first scenario, a third party private search protocol takes place between a Client and a Data Owner. This model has the advantage of performing more quickly, and the disadvantage of requiring the Data Owner to store a cleartext database. The second scenario takes place between a Client and a Server which is storing a Data Owner's encrypted database. While this protocol is less time-efficient, it has the sometimes necessary advantage of storing the data only in encrypted form. The desired outcome in both models is a Boolean value denoting presence (or lack of presence) of the queried value within the database.

With higher security guarantees, a tradeoff between security and efficiency is often unavoidable [55]. Security relaxations are appropriate in some settings such as secure keyword search within a database of documents discussed in the papers referenced above, but these relaxations may not be appropriate when working with medical data due to the high level of privacy required. The protocol proposed in this chapter does not implement any relaxation of security requirements.

Stronger security requirements lead to a higher implementation time than methods with relaxed security requirements. However, implementations show that this scheme can be implemented in an efficient manner for many applications. Specifically, security requirements are kept strong due to potential applications in computational medicine, such as the work in Chapter 7.

6.2 Model

In the unencrypted setting, the Database Owner does not learn the Client's value. This setting is seen in Figure 6.1. A Client privately queries a database held by the Model Owner, who privately provides a query response. The Client does not learn anything about the contents of the database *other than* whether or not his value is contained within it.

The encrypted setting is the same as above, with the additional restriction that the Server does not learn the querying party's value or any unnecessary information about the Data Owner's information. This setting is seen in Figure 6.2. The Server can learn the number of values within the database by observing the number of ciphertexts. The Data Owner can hide the number of values in their database by randomly including different values. Instead of learning the exact number of values in the database the Server learns the maximum number of values which may be contained in the database.

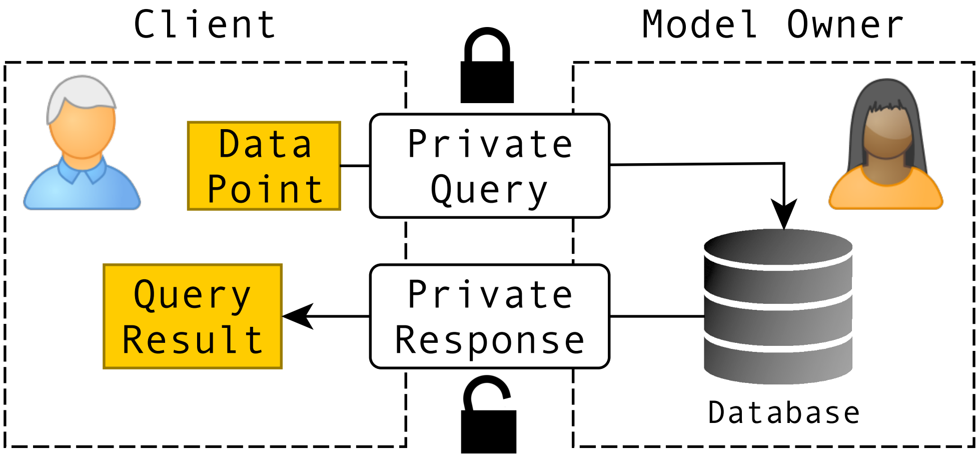


Figure 6.1: Third-Party Private Search

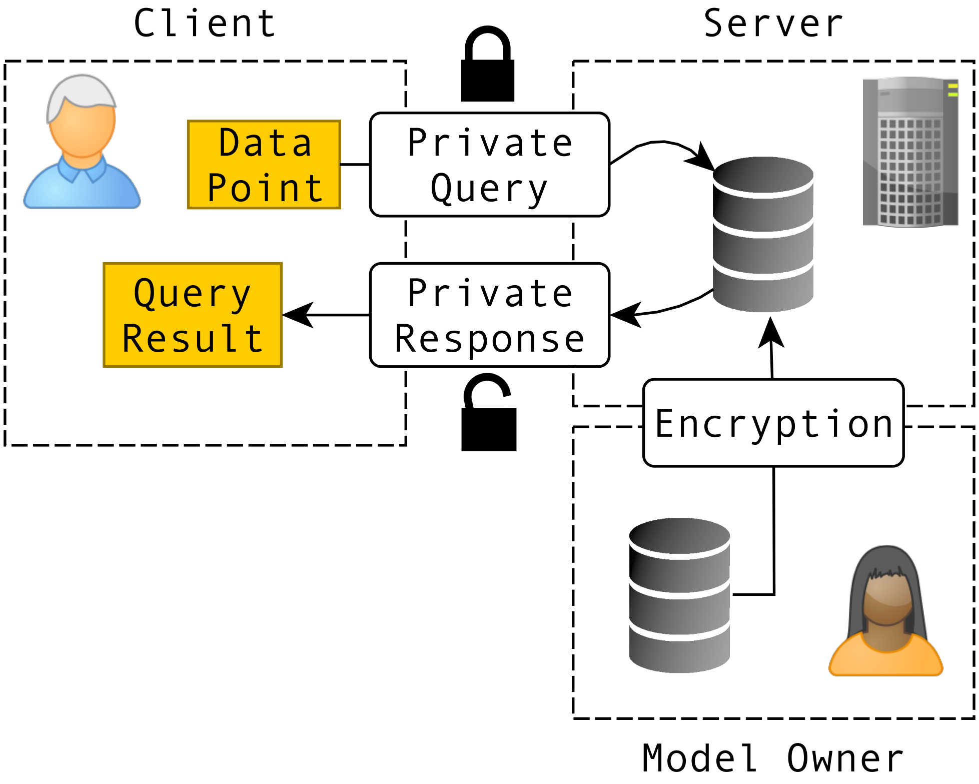


Figure 6.2: Third-Party Private Search via a Server

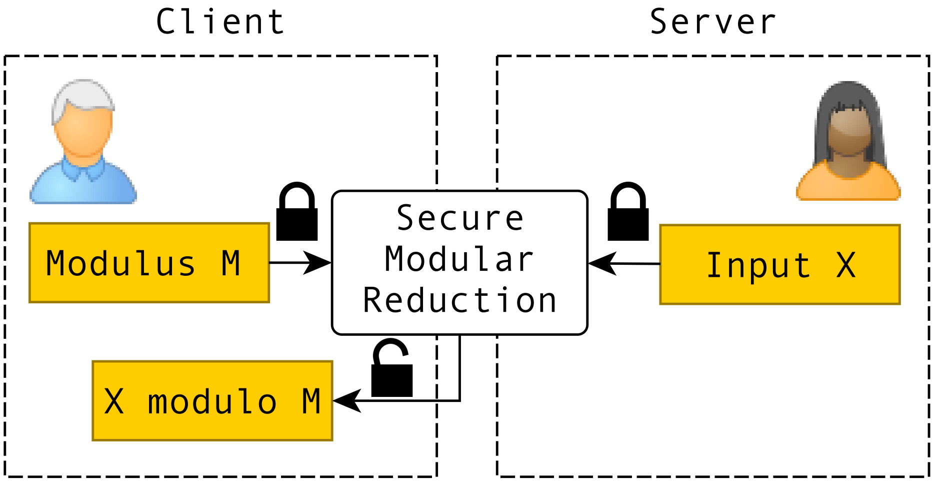


Figure 6.3: Secure Modular Reduction

6.3 Building Blocks

The presented third-party private search protocol implements a number of cryptographic techniques in order to carry out its operations. The TPPS protocol implements fully homomorphic encryption and secure modular reduction during execution. As FHE was detailed in Chapter 2, this section begins by describing a secure modular reduction and finish by defining the proposed TPPS protocol.

6.3.1 Secure Modular Reduction (SMR)

Secure modular reduction describes a class of algorithms which allow for secure evaluation of modular reduction between two parties, one of whom possesses the modulus while the other possesses an input value reduced. The security requirements vary based upon the application. In the proposed protocol, secure modular reduction is implemented with both a private modulus and a private input value.

Figure 6.3 illustrates the SMR protocol implemented within the proposed TPPS. Specifically, the Client provides a modulus M and the Server provides an input X. The Client receives $X \pmod{M}$ without learning the value of X or revealing the value of M to the Sender.

6.4 Third Party Private Search (TPPS) Protocol

The first version of the proposed protocol is implemented between the Client and the Database Owner directly. Let D be a database with entries of at maximum q_1 bits. Let q_2 be an integer such that $q_2 > q_1$. The specifics of parameter selection for q_1 and q_2 are discussed further in Section 6.9.

Algorithm 9 Third Party Private Search

Client Input: A q_1-bit integer x.

Database Owner Input: The database D, with entries of size at most q_1 bits each.

Client Output: A Boolean b, where $b = 1$ if $x \in D$.

```
1: The Client randomly selects a $q_2$-bit integer $r$ and sends $\overline{x} = x + r$ to the Database Owner.
2: The Database Owner computes
       $$y = \prod_{d \in D} (\overline{x} - d).$$
3: The Client and Database Owner perform Secure Modular Reduction to return
       $$\overline{y} = y \pmod{r}$$
   to the Client. If $\overline{y} = 0$, output 1; otherwise output 0.
```

Correct evaluation occurs with some probability depending on the parameters q_1 and q_2. The correctness of the above protocol is discussed in Section 6.6, and suggestions for parameter selection are presented in Section 6.9.

6.5 TPPS Over Encrypted Data

With the rise of cloud computing it is increasingly common for databases to be stored on a Server. With sensitive data such as medical data the values will first be encrypted, then stored. By using fully homomorphic encryption the Server can both store and perform operations over sensitive data. The next version of the proposed protocol is implemented between the Client, the Database Owner, and a Server who handles the bulk of the computational and storage tasks.

Algorithm 10 Third Party Private Search Over Encrypted Database

Client Input: A q_1-bit integer x.

Server Input: The encrypted database $[\![D]\!]$, encrypted under a fully homomorphic encryption scheme.

Database Owner Input: The private encryption/decryption key, k, for the database D.

Client Output: A Boolean b, where $b = 1$ if $x \in D$.

```
The Client randomly selects a $q_2$-bit integer $r$ and sends $\overline{x} = x + r$ to the Database Owner.
The Database Owner encrypts $\overline{x}$ under her private key and sends $[\![\overline{x}]\!]$ to the Server.
The Server computes
    $$[\![y]\!] = \prod_{d \in D} ([\![\overline{x}]\!] - [\![d]\!]) = \left[\!\!\left[ \prod_{d \in D} (\overline{x} - d) \right]\!\!\right]$$
    and sends $[\![y]\!]$ to the Database Owner.
The Database Owner decrypts $[\![y]\!]$ to obtain $y$.
The Client and Database Owner perform Secure Modular Reduction to return
    $$\overline{y} = y \pmod{r}$$
    to the Client. If $\overline{y} = 0$, output 1; otherwise output 0.
```

6.6 Correctness

In the above protocol, if a value is present in the database, then 1 is guaranteed to be returned. This is because the product

$$\begin{aligned} y &= \prod_{d \in D} (\overline{x} - d) \\ &= \prod_{d \in D} (x + r - d) \end{aligned}$$

contains the term $x + r - x$ for some $d = x \in D$, and yields rP for some integer value P. Because of this, $rP \pmod{r}$ will always yield 1, and the Client will receive a true positive.

On the other hand, it is quite possible for a false positive value to be returned. If $x \notin D$ the product y may *still* be divisible by r. While requiring r to be a prime could resolve this issue, it would introduce an unacceptable security flaw. The Database Owner could simply find the closest prime numbers to the value sent by the Client in order to determine a small set of candidates for the Client's original value. Therefore, r is allowed to be uniformly random, and in Section 6.8, we provide simulation results and suggestions for parameter sizes to minimize the likelihood of a false positive occurring.

6.7 Security

Security for the proposed protocol is examined in the honest-but-curious model. Because the goal of the protocol is data sharing, it is necessary that *some* information will be leaked during execution of the protocol. Therefore, we discuss security of the protocol as well as what restrictions must be put in place to avoid *too much* information being shared. How much information is "too much" will depend on the setting. In the proposed setting, the information which is shared with the Client and the Server is the *size* of the Database. The

Data Owner should place a limit on the number of queries a single Client is able to execute based on the maximum number of values contained in the database she wishes for the Client to be able to receive. On the other hand, the Data Owner does not learn what values the Client queries for within the database, but does learn the number of queries made by the Client.

The security of the Client's original value x is protected by the one-time pad r. Because r is a uniformly random q_2-bit value, the value $x + r$ will appear random as long as $x + r$ is still q_2-bits. As $q_2 \gg q_1$, this is a rare scenario that can easily be checked for and avoided during computation.

In Algorithm 9, the Server maintains access to the encrypted database. The security of the database therefore depends on the security of the encryption implemented. The suggested scheme, the GKS scheme [36], is secure against a ciphertext-only attack and hence satisfies the security requirements of the protocol. In both protocols, the Client never has direct access to the database in any form. All the Client receives is the value $\bar{y}$ at the end of the protocol. The privacy of this value depends on the security of the Secure Modular Reduction protocol. The SMR protocol suggested in Section 6.8 is secure in the honest-but-curious model.

6.8 Implementation

Implementation requires selection of SMR and FHE protocols that satisfy the necessary security requirements. Below we provide an overview of the selected protocols.

6.8.1 Secure Modular Reduction

The selected SMR protocol [65] implements the Paillier cryptosystem [48], a partially homomorphic encryption scheme. Let $[\![c]\!]_{\mathcal{P}}$ denote encryption of a value c under Paillier. This scheme satisfies the properties that

- $[\![c_1]\!]_{\mathcal{P}} \cdot [\![c_2]\!]_{\mathcal{P}} = [\![c_1 \otimes c_2]\!]_{\mathcal{P}}$
- $[\![c]\!]_{\mathcal{P}}^{a} = [\![ac]\!]_{\mathcal{P}}$

for all c_1, c_2 and a.

In this protocol the Client possesses the private modulus b while the Server has the private value a. The Server receives $a \pmod{b}$ without learning the value of b or revealing the value of a.

Algorithm 11 Secure Modular Reduction [65]

```
Client Input: An integer $b$.
Server Input: An integer $a$.
Client Output: $a \mod b$.

The Client generates the public key, secret key pair for Paillier encryption and shares the public key with the Server.
The Client sends $[\![b]\!]_{\mathcal{P}}$ to the Server.
The Server chooses $r_d \xleftarrow{\$} (\log_2 N - 1 - \log_2 a)$ and $r_m \xleftarrow{\$} \log_2 a$-bit integers, and computes $[\![r]\!]_{\mathcal{P}} = [\![b]\!]_{\mathcal{P}}^{r_d} \cdot [\![r_m]\!]_{\mathcal{P}} = [\![r_d b + r_m]\!]_{\mathcal{P}}$. It then sends
    $$[\![z]\!]_{\mathcal{P}} = [\![a]\!]_{\mathcal{P}} \cdot [\![r]\!]_{\mathcal{P}} = [\![a + r]\!]_{\mathcal{P}}$$
    to the Client
The Client computes $z \varnothing b = z \pmod d$ and sends $[\![z \varnothing b]\!]_{\mathcal{P}}$ to the Server.
The Server computes
    $$[\![a \varnothing b]\!]_{\mathcal{P}} = [\![z \varnothing b]\!]_{\mathcal{P}} \cdot [\![r_m]\!]_{\mathcal{P}}^{-1} = [\![z \varnothing b - r_m]\!]_{\mathcal{P}}$$
    and sends to the Client.
The Client decrypts to retrieve $a \varnothing b = a \mod b$.
```

6.8.2 Fully Homomorphic Encryption

The FHE scheme implemented within the protocol is the private-key GKS scheme. For more information on this scheme see Chapter 2.

6.9 Minimization of Prediction Error

In order to estimate the prediction error of the proposed protocol, a series of Monte Carlo method based experiments were implemented. The Monte Carlo method is a general term to describe the use of repeated random sampling in order to solve a problem that may or may not be deterministic. These experiments repeatedly computed the product

$$y = \prod_{d \in D} (x + r - d)$$

for a randomly generated databases D, random q_2-bit integers r, and random q_1-bit integers x. Observe that this corresponds to the product computed by the database owner in Algorithms 9 and 10. In these protocols, if $y \pmod{r} = 0$, then the protocol outputs 1, telling the Client that his value is contained in the database. The Monte Carlo experiments return 1 if $y \pmod{r} = 0$ and return 0 otherwise. These responses are counted as true positives, true negatives, false positives, or false negatives based on whether or not $x \in D$. For instance, if $x \notin D$ but $y \pmod{r} = 0$, the response is counted as a false positive.

Each experiment consisted of $200,000$ queries given a fixed q_1 and q_2 for a database containing 10 values. A method for extension to larger databases is presented in Section 6.10.

Let $\mathcal{Q}_1$ denote the set of all q_1-bit numbers. In each iteration of the experiment two values are randomly generated: a q_1-bit number, b, and a q_2-bit value, r. The first $100,000$ iterations randomly generate the values $a_1, \ldots, a_n$ from the set $\mathcal{Q}_1 \setminus \{b\}$, the set of all q_1 bit numbers not containing b. The second $100,000$ iterations assign $a_1 = b$ and randomly generate $a_2, a_3, \ldots, a_n$ from $\mathcal{Q}_1$. These restrictions are imposed in order to ensure that exactly half of the data should yield a positive classification and half a negative classification for an evenly distributed data set. The experiment was run for values $2 \leq q_1 \leq 17$, $3 < q_2 \leq 50$, and $n = 10$. This fixed size for n was chosen with the use of the large database extension

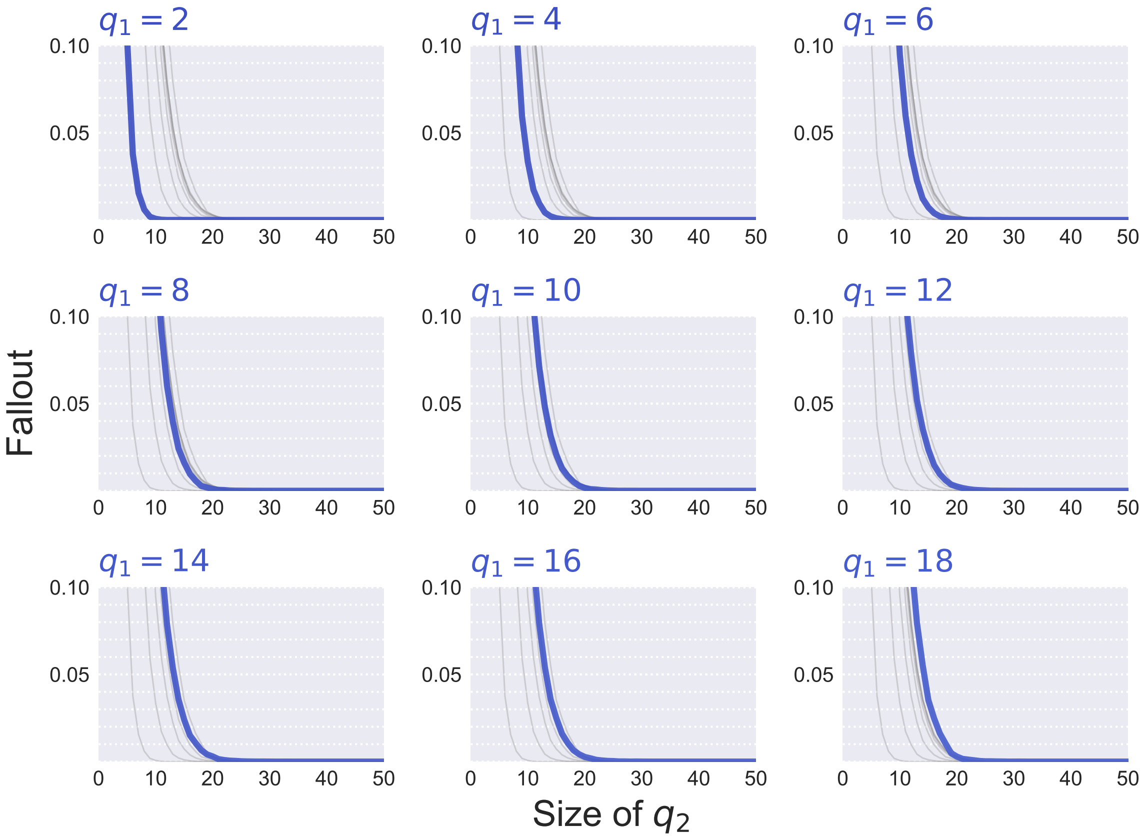


Figure 6.4: The observed fallout for $n = 10$.

method of Section 6.10 in mind. The full table of results from the Monte Carlo experiments are available in Appendix A.

Figure 6.4 shows a series of results of this experiment for a selection of values of q_1. The x-axis on each plot represents value of q_2, while the y-axis denotes the resulting fallout from the Monte Carlo experiments. Recall that fallout is defined as

$$\text{fallout} = \frac{\#\text{ false positives}}{\#\text{ false positives} + \#\text{ true negatives}}.$$

The proposed method has no false negatives, meaning its true positive rate is 100%. In order to avoid false positives, the value of the fallout must be minimized.

There are two methods of approaching selection of q_2 given q_1. The first method is to take a large value for q_2; however, depending on computational restraints, this could lead to expensive operations. Therefore, a smaller value may be taken for q_2 and the experiment repeated the requisite number of times in order to minimize the fallout to below the desired threshold.

6.10 Large Database Extension

When a database is large, the value of the products computed in Algorithms 9 and 10 could be come prohibitively large. The following extension protocol is proposed for databases with a large number of entries.

6.10.1 Unencrypted Model

In the unencrypted model, a large database with n elements is handled by splitting it into k distinct sub-databases, each containing up to m elements from the original database.

Algorithm 12 Third-Party Private Search on Large Database

Client Input: A value x.

Database Owner Input: A database D with n entries, an integer parameter m.

Client Output: A Boolean b, where $b = 1$ if $x \in D$.

```
1: The Database Owner randomly shuffles her database and splits it into $k$ distinct sub-databases $D_1$ through $D_k$, each containing (up to) $m$ entries.
2: The Client randomly selects a $q_2$-bit integer $r$ and sends $\overline{x} = x + r$ to the Database Owner.
3: for $i$ from 1 to $k$ do
4:     The Database Owner computes $y_i = \prod_{d \in D_k} (\overline{x} - d)$.
5: end for
6: for $i$ from 1 to $k$ do
7:     The Client and Database Owner perform Secure Modular Reduction to return $\overline{y_i} = y \pmod{r}$ to the Client. If $\overline{y_i} = 0$, set $b_i = 1$; otherwise set $b_i = 0$.
8: end for
9: Client outputs $b = \sum_{i=1}^{k} b_i$.
```

Note that the first `for` loop in lines 3–5 contains instructions carried out only by the Database Owner. The second `for` loop in lines 6–8 calls a sub-protocol which requires communication between the Database Owner and the Client. The computation carried out up to that second for loop is analogous to the computation carried out in the original protocol, albeit with smaller integer values due to the smaller number of multiplication operations performed. Extra time required by this protocol will occur during the required communication for Secure Modular Reduction. However, implementing the protocol in this model allows for implementation with large databases, a clear advantage.

6.10.2 Encrypted Model

In the encrypted setting, where private search is performed over an encrypted database, the method of model extension is almost the same as the unencrypted case. However, the structure of the encrypted data can be used advantageously via Single-Instruction Multiple Data (SIMD) instructions. Specifically, under the default parameters of the GKS encryption scheme with $n = 5$, a maximum of $2^n = 32$ values may be simultaneously encoded and encrypted in a single ciphertext.

Assume a Database Owner has a database containing n elements. During encryption of the database the Database Owner encrypts the maximum number of values within a single data point that SIMD allows. Say there are n' resulting ciphertexts. The Server stores these n' values in ciphertext form. During execution of the extended protocol, the Server randomly splits these n' values into subsets via the extension method above.

The number of calls to the Secure Modular Reduction protocol is not reduced, and must be performed for each sub-database as well as each SIMD slot. The described method provides a great boost in computation speed due to the SIMD slots as homomorphic multiplication is an expensive operation. With the default parameters of the GKS scheme containing 32 slots, the number of homomorphic multiplication operations required to be performed is divided by 32.

6.11 Evaluation

Tests were performed to determine the performance of the proposed protocol given varying sizes of databases in both the encrypted and unencrypted setting. Experiments were run on a MacBook Pro with a 2.3 GHz processor and 16 GB memory. Table 6.1 shows the execution time of the private search extension protocol. Results are given for a wide range of database sizes, with the time in seconds for execution in the encrypted and the unencrypted settings.

Dataset Size		**160**	**320**	**640**	**960**	**1280**	**1600**
Time (s)	**Unenc.**	0.016	0.031	0.063	0.098	0.128	0.170
	Enc.	0.290	0.454	0.821	1.203	1.645	1.958
Dataset Size		**2560**	**5120**	**7680**	**10240**	**12800**	**15360**
Time (s)	**Unenc.**	0.248	0.789	0.994	1.023	1.268	1.554
	Enc.	3.184	6.150	9.332	12.400	15.401	18.629

Table 6.1: Execution Time, Private Search Extension

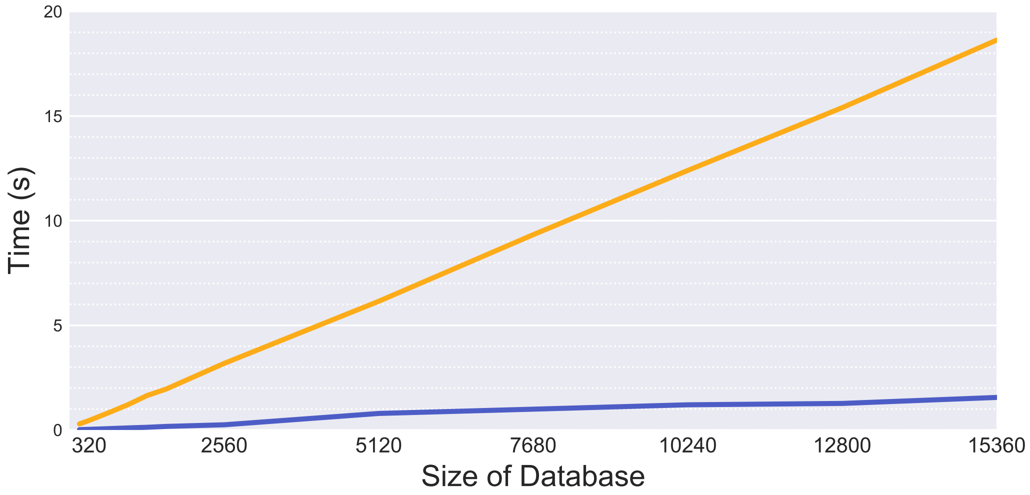


Figure 6.5: Execution Time, Private Search Extension

Figure 6.5 provides a visual comparison of encrypted versus unencrypted computation times. As expected, the time taken per protocol is linear with respect to the size of the database. Speeds in the unencrypted model took place in below half a second for small database sizes, and below two seconds for databases with $15,360$ elements. Computation in the encrypted model was more computationally intensive and performed in under 1 second for small database sizes, and under 20 seconds for large databases. All experiments resulted in a 100% true negative rate and false negative rate, meaning there were no false positives for false negatives. This is of great importance for medical applications, where a false positive can result in a misdiagnosis and a false negative could result in a crucially missed diagnosis.

6.12 Discussion

6.12.1 Computational Bottlenecks

Computation time of Algorithm 12 is determined by a number of factors. In particular, a major computational bottleneck will be homomorphic multiplication operations and communication costs.

Consider a database D with n entries in the unencrypted model. Say this database is split into a collection of smaller databases, each with m entries. Without loss of generality, consider databases where the size n is a multiple of m, as this will provide an upper bound on computation. This results in a total of $k = n/m$ sub-databases. In the first `for` loop in lines $3 - 5$, the Database Owner computes the product

$$y_i = \prod_{d \in D_k} (\overline{x} - d),$$

where each product y_i is computed via d multiplication operations. Therefore, the number of multiplications carried out is bounded by $k \cdot m = n/m \cdot m = n$.

The second `for` loop in lines $6-8$ carries out Secure Modular Reduction k times. During Secure Modular Reduction, seen in Algorithm 11, the Client communicates with the Server 2 times and the Server communicates with the Client 2 times. This would imply that a total of $4 \cdot k$ communications are required during this for loop. However, communication cost can be dramatically reduced by breaking up the steps along communication.

In particular, a series of k Secure Modular Reductions can be carried out in only 4 total communications between the Client and Server. The Client first generates a single public key, private key pair for Paillier, then encrypts all k values which are to be reduced under the public key. The Client sends all k of these encrypted values to the Server in one message. The Server then performs Step 3 of Algorithm 11 on all values received, and sends all k

resulting values to the Client in one message. Steps 4 and 5 proceed similarly, where the Client and Server perform all k computations locally and send k results simultaneously.

This method does not reduce the local computation required by the Client and the Server. It does, however, greatly reduce communication costs. All together, TPPS over a large database in the unencrypted model requires at most n multiplication operations and 5 communications between the Database Owner and the Client.

In the encrypted model, further optimization is possible. In particular, consider an implementation of SIMD where ℓ database elements may be encoded simultaneously. Then, a sub-database containing m ciphertexts will in fact contain as many as $m \cdot \ell$ database elements. Without loss of generality assume there is a database D of size n, where n is divisible by $m \cdot \ell$, in order to provide an upper bound on multiplication costs. Then, splitting D into sub-databases each containing m ciphertexts will result in

$$k = \frac{n}{m \cdot \ell}$$

sub-databases. Therefore, homomorphic multiplication is carried out only $k \cdot m = n/\ell$ times.

Communication costs can be minimized in the encrypted case in the same way there were minimized in the unencrypted case. This method requires one communication between the Database Owner and the Server, and one communication between the Server and Database Owner. TPPS over a large database in the encrypted model therefore requires n/ℓ homomorphic multiplication operations and 7 communications between Database Owner, Client, and Server.

6.12.2 Comparison

Other research that has focused on private document retrieval is not directly comparable. Reported times include costly preprocessing of text document databases as well as file transfer time costs. Raykova et al. provide results on their secure anonymous database search protocol, which returns a list of documents containing a queried keyword, by listing the number of document matches found for a variety of query and aggregate search function configurations [55]. This model presents a variable number of false negatives based upon the aggregate search function configuration implemented during construction of the documents' Bloom filters.

Private information retrieval carried out by Angel et al. is evaluated using Microsoft Azure's powerful data centers [3]. The PIR servers are equipped with 16-core 3.6 GHz processors with 112 GB of memory, and the Client's servers are equipped with 16-core 2.4 GHz processors and 32 GB of memory. The majority of computation occurred on the PIR server, and the goal of the protocol is to return to the Client the documents matching the Client's query.

Pappas et al. also perform experiments on a private database retrieval protocol, where the Client seeks to download files containing a keyword [49]. They perform queries on a data set containing $5,000$ keywords. They report the initial query response time, where the Client receives the set of document IDs containing a queried keyword, as well as the time for the entire document retrieval protocol. The initial query returning document IDs containing a keyword occurs as fast as under 50 milliseconds for a database containing $50,000$ keywords. The protocol achieves a hit ratio of approximately 90% for retrieval of document IDs containing the queried keyword. However, these results do not apply to the goal of database membership queries explored in this chapter, as a high occurrence of false positives or false negatives is unacceptable.

Khedr et al. perform what they call secure multiple keyword search [41]. This setting is similar to the setting explored in this chapter. Specifically, the authors explore the scenario where a Client wishes to query a text file for the presence of a keyword. The authors provide timing results for partially secure database search, where the Client's query is not hidden but the database itself is hidden, and later for a fully secure database search, where both the Client's query and the database values are hidden, using their proposed FHE scheme as well as IBM's HElib [59]. The authors run their experiments on a GPU with $2,048$ CUDA cores with 4 GB of memory. Files containing 140 words are queried in approximately 10 seconds using the authors' proposed scheme as a platform within their multiple keyword search protocol, while queries running via HElib as a platform take over $1,000$ seconds to query a file of the same size. The results given in Table 6.1 show significant improvement over this performance.

6.13 Conclusions

This chapter provides results on a third party private search protocol which performs private membership queries between a Client and Database Owner, with or without an intermediary Server storing the database in encrypted form. Results show that large databases can be queried quickly and accurately using the proposed method. Future work could focus on technical improvements leading to faster performance or extension to a SADS scheme with a high hit ratio.

This functionality has applications in a variety of fields. A potential application in the medical field is to allow researchers to privately query a database owned by another institution. In this scenario, the researchers could query the database to determine if it contains information of interest to them without the institution learning their query. Email could be monitored for spam keywords without compromising the privacy of a user's emails. Further applications lie in the field of law enforcement, where confidential data could be stored protected in encrypted format while maintaining the utility to allow investigators to query the data. For instance, a list of known offenders could be queried for the presence of a suspect by an investigator who cannot directly access the names of people on the list, all without the investigator compromising the privacy of the suspect.

In the next chapter, the presented third party private search protocol is used for classification of real medical data via decision tree models.

Chapter 7

Privacy-Preserving Decision Tree Classification

7.1 Introduction

Binary decision trees are a method of classification that can be represented in a simple diagram by interior decision nodes and terminal leaf nodes. The leaf nodes at the bottom of the tree provide the final classification for a data point. Due to the representation as a tree structure, decision trees are easily to interpret and understand. This ease of interpretability is one clear advantage of using decision trees. In fact, this method is favored among scientists in the medical community as it is easy to visualize and it "mimics the way a doctor thinks" by "stratify[ing] the population into strata of high and low outcome, on the basis of patient characteristics" [39]. Due to its medical utility, a privacy-preserving classification protocol using Classification and Regression Trees (CART) is presented in this chapter, and experiments on a real-world medical data set are performed efficiently.

The primary contribution in this chapter is the construction of a privacy-preserving decision tree classifier. This protocol uses a variety of cryptographic primitives in order to construct private decision tree classification. It utilizes third party private search protocol, fully homomorphic encryption, secure modular reduction, and a primitive called *oblivious transfer*. This chapter begins with discussion of the background and methods required for these primitives and related work in the field in Sections 7.2 and 7.3. Sections 7.4 and 7.5 present the proposed protocol and discusses its security. Sections 7.6, 7.8, and 7.9 discuss implementation of the protocol and classification results on a real-world medical data set.

7.2 Classification and Regression Tree (CART)

The Classification and Regression Tree method, or CART, is a tree-based implementation of supervised learning for classification and regression [12]. Like Naive Bayes, this method has the advantage of being both conceptually simple and robust.

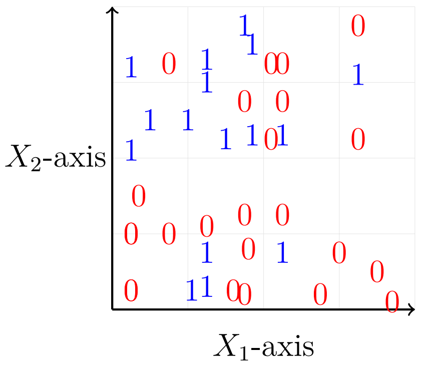


Figure 7.1: Binary data in $\mathbb{R}^2$.

A *tree-based* method operates by choosing a sequence of binary splits to apply to the data. Figure 7.1 shows a collection of toy data points with binary classifications in a two-dimensional subspace of $\mathbb{R}^2$ on axes X_1 and X_2. Each of these regions is then assigned a corresponding class based on a majority vote within the region. In Figure 7.6, the sub-figure 7.2 shows the region after a sequence of binary splits into 6 final regions. Sub-figure 7.3 gives the binary tree which represents these splits. Sub-figure 7.4 shows the sub-regions with the toy data points and sub-figure 7.5 shows the classification tree resulting from a majority vote of classes points within the sub-region.

While this example is for data points in $\mathbb{R}^2$, the concept applies to $\mathbb{Z}^2$ or categoricalvalued feature vectors with more than two classes. While it is difficult to visualize these binary partitions in M-dimensional space, the binary tree representation of the partitions remains straightforward to draw as a binary tree in any number of dimensions.

7.2.1 Growing Classification Trees

The exposition below follows that of Hastie [39] and uses data with d features from $\mathbb{R}^d$ which lies in a discrete set of c classes $G = \{G_1, \ldots, G_c\}$. The method can be easily extended to integer-valued and categorical data points. Consider p input data points, each of the form (X, G_i) where $X = (X_1, \ldots, X_d) \in \mathbb{R}^d$. Training determines a sequence of binary partitions that minimize the amount of error present in the final classification contained in each leaf.

A *greedy algorithm* is implemented in order to split the input space via binary partitions in an efficient manner. This greedy algorithm will minimize the classification error in each region at each step.

For the first split, a dimension, or *splitting variable*, j, and a *split point* s, are chosen in order to minimize the classification error in each of the two resulting regions based on a majority vote. Formally the two regions are defined by

$$R(j,s) = \{X : X_j \leq s\}$$
$$R'(j,s) = \{X : X_j > s\}$$

and the goal is to minimize the classification error over the variables j and s. Let $\mathrm{Err}(R(j,s))$ and $\mathrm{Err}(R'(j,s))$ represent the measure of *node impurity* (e.g. misclassification error, Gini index) in $R(j,s)$ and $R'(j,s)$, respectively. The equation

$$\min_{j,s} \left[\mathrm{Err}(R(j,s)) + \mathrm{Err}(R'(j,s))\right]$$

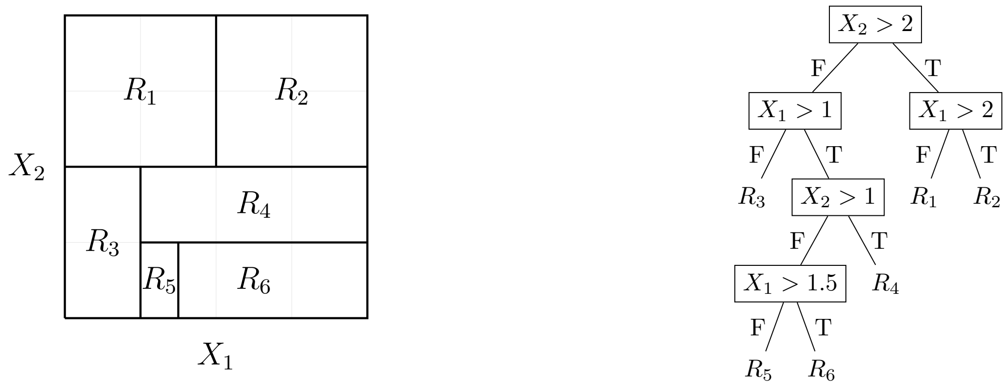


Figure 7.2: A two-dimensional region under a series of binary splits.

Figure 7.3: A binary tree corresponding to the binary splits in Figure 7.2

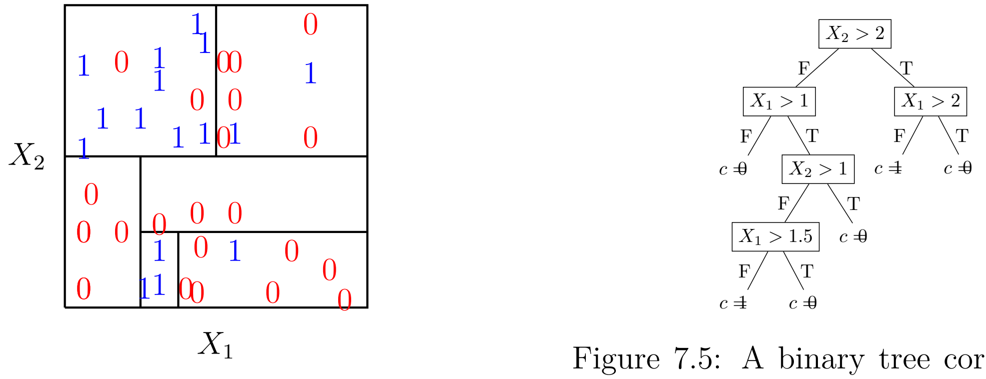


Figure 7.4: Data within a 2-dimensional region under a series of binary splits

Figure 7.5: A binary tree corresponding to a majority vote within the regions in Figure 7.4

Figure 7.6: The CART method

is minimized by testing all potential values for the splitting point s. This algorithm can be carried out efficiently [39].

This process is carried out in an iterative manner on each sub-region. Determining when to stop involves finding a balance between a tree that is too large and over fits to the data and a tree that is too small and under fits the data. One method is to grow the tree until each region contains only η data vectors then apply *pruning methods* such as cost-complexity pruning to shrink the tree. In this case, η is called the *minimum node size.*

Pruning is carried out follows. First an initial tree T_0 is grown until each region contains at most η data vectors. Say there are K leaf nodes determining regions $R_k \subset \mathbb{R}^d$ for $1 \leq k \leq K$.

Let T denote a subtree $T \subseteq T_0$ and define

$$N_k = \#\{X \in R_k\},$$

the number of data vectors in R_k, and

$$p_{k\ell} = \frac{1}{N_k} \sum_{X \in R_k} I(G_i = \ell),$$

the proportion of class ℓ vectors in region R_k. There are several potential measures of *node impurity*:

- The *classification error* given by
$$1 - p_{k\ell}.$$
- The *Gini impurity* given by
$$\sum_{\ell=1}^{c} (1 - p_{k\ell}).$$
- The *entropy* given by
$$-\sum_{\ell=1}^{c} p_{k\ell} \log p_{k\ell}.$$

Gini impurity and entropy are most often used as measures of node impurity during the tree growing stage while the misclassification error is most often used during the pruning phase [39]. Denote the chosen measure of node impurity as $Q_k(T)$. The cost complexity criterion

$$C_\alpha(T) = \sum_{k=1}^{|T|} N_k Q_k(T) + \alpha |T|$$

is minimized for a tuning parameter $\alpha \geq 0$. When $\alpha = 0$ the tree $T = T_0$, and T becomes smaller as α becomes larger. A value for α which determines the unique smallest subtree T minimizes the cost $C_\alpha(T)$. This optimization problem has a global solution [39], and the final model is given by the tree resulting from minimizing the cost.

7.2.2 Classification of New Data Points

Consider a data point X with an unknown class. Classifying this data point using a trained CART model consists of applying the condition in each node of the decision tree to X and following the branches sequentially until a leaf node is reached. The class contained in this leaf node is the class value assigned to X. The precise method of classification of new data points via a trained decision tree implemented in the presented protocols is given in Algorithm 13.

7.3 Methodologies

Moving from classification of new data points in the clear to *private* classification of new data points requires hiding both the tree structure from the Client and hiding the Client's data input from the Model Owner. One approach converts a decision tree into its *polynomial form* [6, 61]. Another approach converts a decision tree into a *complete binary tree* and performs a randomization procedure. These two approaches are discussed below. After this, *oblivious transfer* is introduced, as it will be a necessary step within the proposed private decision tree classification protocol.

7.3.1 Randomizing Trees

Methods which first convert a tree into a polynomial form run in the public-key setting between a Client, Server, and Model Owner. Consider a decision tree with n decision nodes. Each of these notes has a binary output. Denote the binary output of each node for some input as $b_1, b_2, \ldots, b_n$. Let $c_1, c_2, \ldots, c_n$ denote the corresponding leaf nodes, containing a binary classification value. A recursive procedure allows for efficient computation of a polynomial $P(b_1, b_2, \ldots, b_n, c_1, c_2, \ldots, c_n)$ such that the output of P corresponds to the class of the input value for which the decision nodes were evaluated [6]. The methods of Bost et al. [6] allow for private classification to occur under this model using two encryption schemes, the Quadratic Reciprocity (QR) scheme [34] and a public-key FHE scheme. For private classification, the Client holds the FHE private key and the Model Owner holds the QR key. The Client computes the values $[b_i]_{QR}$ for i from 1 to n using some privacy-preserving protocol, encrypts these (encrypted) values via FHE, and sends them to the Model Owner. The Model Owner then evaluates the polynomial P, encrypted under both QR then FHE. The authors then provide a method for the Client to receive the output of this function without revealing the output to the Model Owner.

These protocols all generally follow the broad steps, a modification on the work of Bost et al. [6].

1: The Client publishes a public key for some public-key encryption scheme.
2: The Model Owner encrypts the polynomial form P of a decision tree T and stores this encryption on the Server.
3: The Client and Model Owner perform a series of privacy-preserving protocols in order to determine the binary output of the tree on each node. In this setting, privacy-preserving means that the Client does not directly learn the evaluation of his data point on each node, and the Model Owner does not learn the Client's data point.
4: The Server evaluates the polynomial over the data point using fully homomorphic encryption [41, 61] or some other method [6].
5: The Client decrypts the output to receive his classification.

Khedr et al. perform private decision tree classification using this polynomial representation paradigm [41]. Their protocol allows for implementation of classification via the polynomial form of a binary decision tree and uses public-key fully homomorphic encryption for classification in place of the methods of Bost et al. [6]. Sun et al. also present a private

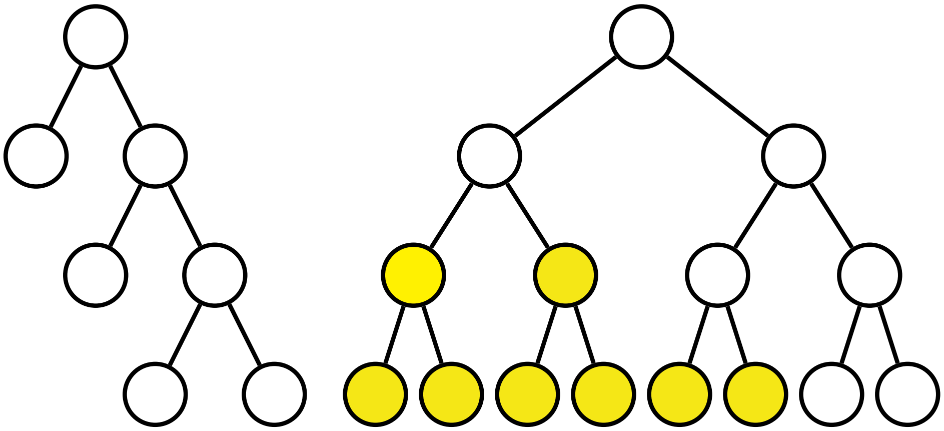


Figure 7.7: The completion (right) of a binary tree (left).

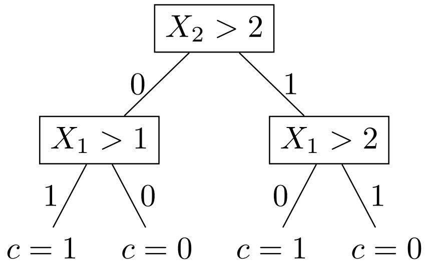
decision tree methodology also based on polynomial forms of trees using public-key fully homomorphic encryption [61].

In the private key setting, both parties are unable to encrypt their data under a public key, and another method is necessary

7.3.2 Complete Tree Randomization

The methodology presented by Wu et al. [68] of *complete binary tree randomization* does not require a polynomial representation of a tree. In this model, "dummy" nodes are introduced into a tree in order to impose a uniform depth upon its structure.

The dummy nodes contain random evaluations with binary output, where each response leads to the same outcome. When privatization of the tree structure is not a goal, this structure is redundant. However, with a complete binary tree, regardless of the outcome on individual nodes any data point will require the same number of evaluations to compute. This is an important aspect of a Model Owner ultimately hiding the tree structure from a Client. An example of a completion of a binary tree is shown in Figure 7.7. The yellow decision nodes contain random binary evaluation functions, and the leaf nodes contain the class corresponding to the assigned class in the original tree on the left.



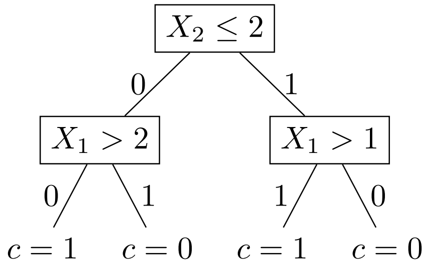


Figure 7.8: A binary tree (left) and the tree negated on the first node (right).

The authors describe a tree randomization procedure in which a complete tree is hidden via by randomly permuting the nodes of a tree T to obtain an equivalent tree T' [68]. An equivalent tree is a tree with the same depth and the same classification output for every data point, but with a different classification path within the tree. An equivalent tree is created by randomly flipping the outcome of the binary decision function at each node. Their algorithm proceeds as follows, with input of a tree T and output of a randomized tree T'. The tree T contains n leaf nodes t_i

```
1: Initialize $T' = T$.
2: Randomly choose $s = (s_1, \ldots, s_n) \in \{0, 1\}^n$.
3: For $i$ from 1 to $n$, if $s_i = 1$ then negate the decision function on the node $t'_i$ of the tree $T'$ and swap the subtrees originating at the left and right child nodes.
4: Re-order the node indexes and output $T'$.
```

And example of a binary tree which has been negated on the first node is available in Figure 7.8.

7.3.3 Oblivious Transfer

Originally introduced by Rabin in 1981, oblivious transfer was first described as an RSA-based cryptographic primitive which allowed a Receiver to obtain a message from a Sender with probability of 0.5 without the Sender knowing whether or not the value was received

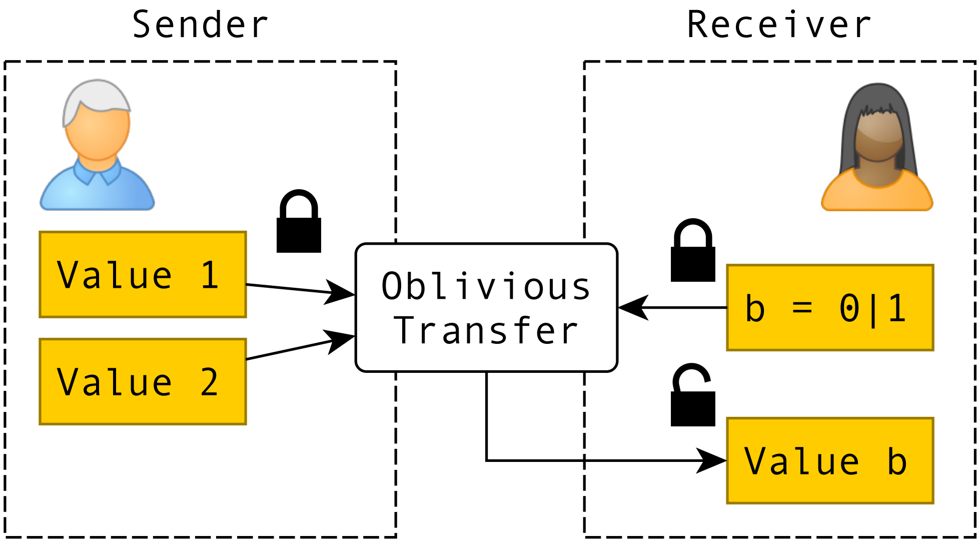


Figure 7.9: 1-of-2 Oblivious Transfer

[53]. In the years since, 1-*of*-2 *oblivious transfer* has developed into a two-party cryptographic primitive which allows a receiving party to obtain exactly one value out of two values sent by the sending party without revealing to the sender her choice. Figure 7.9 provides a visualization of the usual model. The Sender and Receiver perform a key generation algorithm which includes the Receiver's choice of index. The sender then encrypts the two values based on these keys and sends them to the Receiver. Decryption reveals the value associated with the index selected during the key generation algorithm while decrypting the other value yields an output which appears random to the Receiver, effectively masking its true value from her.

In a 1-of-n oblivious transfer protocol, the Sender has values $b_1, b_2, \ldots, b_n$ and the Receiver has chosen an index i. The Sender would like to share b_i with the Receiver without revealing any of the other values to her. The Receiver wishes to hide her index from the Sender. The protocol presented for third-party private search implements a 1-of-n oblivious transfer protocol.

7.4 Tree Representation and Decision Tree Classification

A complete binary decision tree can be represented in the following manner. A complete binary decision tree D is represented as a collection of k nodes, $D = \{d_1, d_2, \ldots, d_k\}$. A tree with depth m has $2^m - 1$ nodes. For a tree with depth m, nodes with indexes from 1 to $2^{m-1} - 1$ are decision nodes and nodes with indexes from 2^{m-1} to $2^m - 1$ are leaf nodes.

Determine child nodes based on the binary representation of the index at the node. The root node is assigned the index `1`, which corresponds to the integer value 1. Its child nodes are given by `10` and `11`, corresponding to nodes 2 and 3, respectively. Node `10` is the child node of `1` when the outcome on that node is negative, or 0; node `11` is the child node when the outcome is positive, or `1`. Figure 7.10 provides an example of a tree with depth $d = 4$. The value inside of the node represents the node index in binary, and the value on each arrow represents the binary decision outcome on that node.

To move from decision node to its child node, evaluate the node on the input data point and concatenate the decision to the binary representation of the node index. Convert back to decimal to retrieve the node index number. To move from a child node to its root node, dissociate the last value from the binary representation of the index. For example, the parent node of `1011` is `101`.

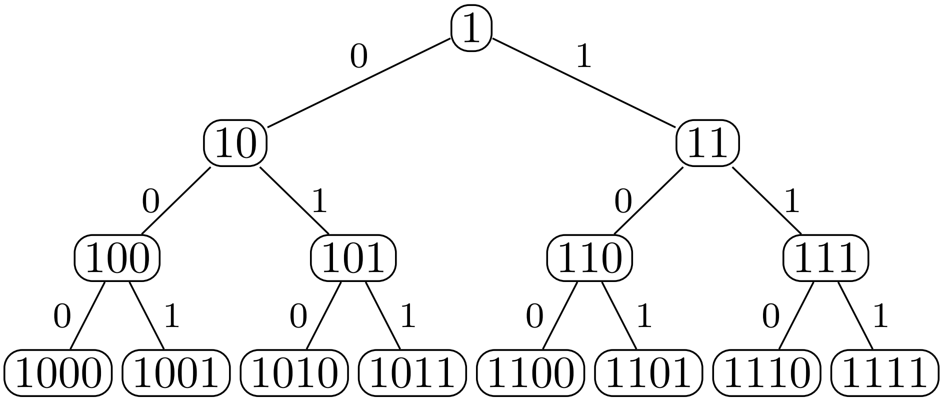


Figure 7.10: Binary Tree Node Assignment

These values can also be represented in base 10. Let I be the index of a node. Then, the index of its parent node is given by

$$\text{Parent}(I) = \lfloor I/2 \rfloor.$$

The indexes of its child nodes are given by

$$\begin{aligned}\text{LeftChild}(I) &= 2I\\ \text{RightChild}(I) &= 2I + 1.\end{aligned}$$

The leaf node associated with a data point may be reached using its binary representation via the following algorithm.

Algorithm 13 Decision Tree Classification

Input: A decision tree T in complete binary tree form with n nodes and depth m, and a data point X to be classified via T. Let b_I denote the binary evaluation output of X on the tree node with index I.

Output: The index of the leaf node containing the classification of X.

```
$I = 1$                          ▷ Initialize the node index to 1.
$J =$None                        ▷ Initialize an empty output $J$.
for $i$ from 1 to $m$ do
    $J = J \| b_I$.              ▷ Operator $\|$ denotes concatenation.
    $I = 2I + b_I$.
end for
Output $J$.
```

7.5 Private Decision Tree Classification Protocol

Consider a Client, $\mathcal{C}$, a Data Model Owner, $\mathcal{D}$, and a Server, $\mathcal{S}$. First, consider a general overview without implementing privacy-preserving measures. The Data Model Owner trains a binary decision tree, T, under the CART algorithm. This tree is presented as a complete tree of depth n contains of $2^n - 1$ nodes with index labeling as outlined above. Some of these nodes may be dummy nodes. Denote each decision node by t_i with corresponding decision function f_i for i from 1 to $2^{n-1} - 1$. The leaf nodes are denoted by t_i with class assignment c_i for i from 2^{n-1} to $2^n - 1$. The The Model Owner stores this tree on the Server. The Client has a data point with d features,

$$X = (X_1, X_2, \ldots, X_d).$$

The Client wishes to classify his data point using the Model Owner's tree. The Client computes the binary decision function f_i on each decision node t_i in order to determine his tree path, then retrieves the class assignment G_i which corresponds to that path.

Each step in this procedure must be randomized. First, the tree T must be replaced with an equivalent randomized tree T'. Furthermore, the Client needs a privacy-preserving method of determining his path in the randomized tree. Note that the Client must query *every* node in the tree – otherwise, the Server can determine the path he followed based on his queries, and a colluding Server and Model Owner could then determine his classification.

The proposed protocol implements third-party private search in order to perform classification. Recall that in medical applications it is common for features to take on only a discrete set of values. If the value is some continuous measurement, it can be quantized to take on only some discrete set of values.

This representation of a feature as a discrete set of possible values is used in order to implement third-party private search for node evaluation. Let y be some feature which can take on ℓ discrete values which are assigned numbers the numbers 1 to ℓ. Say a decision function splits this feature at k – all values greater than or equal to k result in `True`, while all values less than k result in `False`. Evaluation of this decision function may be reduced to set membership. The decision function can be represented by the set $\{k, k+1, \ldots, \ell\}$, and a `True` output occurs for all values which occur within the set.

Once the Client has determined the index of his classification on the randomized tree, he retrieves his final classification by implementing an oblivious transfer protocol with the Model Owner. Algorithm 14 carries out this private decision tree classification evaluation.

Algorithm 14 Private Decision Tree Classification

Client Input: An n-tuple of q_1-bit integers $x = (x_1, x_2, \ldots, x_n)$.

Model Owner Input: The trained decision tree D.

Client Output: The assigned class of x based on the decision tree D.

```
The Client performs $D' =$ RandomizeTree($D$).        ▷ See Algorithm 16.
Initialize variable $y =$ '1'.
for Node $i$ in $D'$ do
    Compute $b_i =$ Node($x, D, i$)                    ▷ See Algorithm 15.
end for
Perform Oblivious Transfer to reveal the classification value at the tree node determined in the loop above.
```

The method for performing private node evaluation using third-party private search is outlined in Algorithm 15.

7.5.1 Private Node Evaluation

The Client must compute the binary output of the decision function on each node for his input data point. In order to carry this out, the Client implements a version of the Third Party Private Search (TPPS) protocol described Algorithm 9 in Chapter 6.

While in the previous chapter the Client was able to perform TPPS within a data set for one value, the Client in this scenario has d values corresponding to the d features used within the model. An oblivious transfer protocol is implemented within the TPPS framework in order to mask which feature the Model Owner evaluates over during execution. This method of oblivious transfer with TPPS is described in Algorithm 15 below.

Algorithm 15 Private Node Evaluation Protocol

Client Input: An n-tuple of q_1-bit integers $x = (x_1, x_2, \ldots, x_n)$.

Model Owner Input: The index of the feature at the node, I, and the database for that node, D.

Client Output: `1` if $x_I \in D$, `0` if $x_I \notin D$.

```
The Client randomly selects a $\log_2(q_2)$-bit integer, $r$, according to the parameters of the TPPS protocol.
The Client computes
    $$\overline{x} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n}) = (x_1 + r, x_2 + r, \ldots, x_n + r).$$
The Client and the Model Owner perform an OT extension protocol in order to transfer the value $\overline{x_I}$ from the Model Owner to the Client, where $I \in \{1, 2, \ldots, n\}$ is the index of the decision variable on the node.
The Client and Model Owner perform a Third-Party Private Search protocol to determine whether $x_I \in D$. If $x_I \in D$, the client outputs 1; otherwise, the client outputs 0.
```

The private decision tree protocol in Algorithm 14 carries out Algorithm 15 to determine the binary output on each node. This binary output is used for the final classification of the Client's data point.

7.6 Security

The desired security is that the Client learns nothing about the Model Owner's learned model, the Model Owner learns no information about the Client's data point, and the Server learns no information about either party's private data.

7.7 Implementation

The following methods were used in order to implement the Private Decision Tree Evaluation protocol.

7.7.1 Tree Randomization

Tree randomization is implemented via the following, Algorithm 16. This algorithm proceeds by first generating an n-tuple of random bits, b_i, denoting whether a nodes decision will be reversed or not. If the node is reversed, the subtrees stemming from the children nodes are swapped. This differs from the protocol described in Section 7.3.2 in several small ways. Instead of copying the tree, randomizing, then re-indexing the output tree, the algorithm below computes a permutation π on the indexes of the nodes in T such that $T' = \pi(T)$ is a randomized version of T. Because of this difference in approach, the same n-tuple of random bits would result in different randomized tree outputs under the two algorithms. However, both ultimately result in an efficiently computed randomized tree.

Algorithm 16 Tree Randomization (adapted from [68])

Input: A decision tree T in complete binary tree form with n nodes and depth m.

Output: A randomized decision tree T' in complete binary tree form.

```
Generate a random permutation $\pi(n)$.
Generate an $n$-tuple of random bits, $b = \{b_i\}$.
for $i$ from 1 to $n$ do
    if $b_i = 1$ then
        $d = \log_2(i)$
        for $j$ from 1 to $m - d + 1$ do
            for $k$ from 0 to $2^{j-1}$ do
                Switch the value of $\pi(i \cdot 2^j + k - 1)$ with the value of $\pi(i \cdot 2^j - k + 2^{j-1} - 1)$.
            end for
        end for
    end if
end for
for $i$ from 1 to $n$ do                ▷ Create the randomized tree via the permutation $\pi$.
    $T'(i) = T(\pi(i))$.
end for
```

7.7.2 Oblivious Transfer

A protocol, simply called "The Simplest Protocol for Oblivious Transfer," is implemented in our experiments [16]. This is a group-based 1-of-n oblivious transfer protocol, the algorithm for which is provided below in Algorithm 17. In this setting, Alice is the Receiver in Figure 7.9 and Bob is the Sender.

Algorithm 17 1-of-n Oblivious Transfer [16]

```
Sender Input:
Receiver Input: The trained decision tree $D$.
Receiver Output: The assigned class of $x$ based on the decision tree $D$.

1: The Sender and Receiver share randomly generated public keys $p$ (prime) and $g \in \mathbb{Z}_p$ and agree upon a hash function $H$.
2: The Sender selects $b \xleftarrow{\$} \mathbb{Z}_p$ and sends $B = g^b$ to the Receiver.
3: The Receiver selects $a \xleftarrow{\$} \mathbb{Z}_p$ and sends $A = B^I g^a = g^{bI+a}$ to the Sender for her choice of $I \in \{1, 2, \ldots, n\}$.
4: The Sender computes the keys $k_i = (A/B^i)^b = g^{(I-i)b^2+ab}$ for $i = 1$ to $n$.
5: The Receiver computes the key $k = B^a = g^{ab}$.
6: The Sender sends $e_i = M_i \oplus H(k_i)$ to the Receiver for $i = 1$ to $n$.
7: The Receiver computes $M_I = e_I \oplus H(k)$.
```

7.8 Evaluation

Decision trees were trained using Python 3 with 10-fold cross validation. Random oversampling was implemented during training on the positive classification set, as these were less represented in the overall data set. Trees were limited to a maximum depth of 5, a minimum of 3 samples per split, and a minimum of 3 samples per leaf. Gini impurity was used to measure the quality of the split during training.

The protocol was implemented in C++ on a Windows 7 machine with a 3.40Ghz processor and 32.0GB memory using the GNU MP Bignum Library [62] for arbitrary precision arithmetic, Chou and Orlandi's "Simplest OT Extension" protocol [16], Veugen's secure modular reduction protocol [65], and the GKS private-key FHE scheme [36].

	Accuracy	**Sensitivity**	**Specificity**	**Precision**	**NPV**	**F1-score**
Mean	0.96628	0.96172	0.97500	0.98699	0.93487	0.97367
Stand. Dev.	0.02192	0.03010	0.05270	0.02700	0.04875	0.01695

Table 7.1: Decision Tree Classification Results

	Time (s)
Unencrypted, Not Private	0.00001
Encrypted	0.91251

Table 7.2: Decision Tree Classification Time

Training and testing took place under 10-fold cross validation. The results are available in Table 7.1 and Table 7.2. The results in Table 7.1 show that the decision tree classifier outperforms the Naive Bayes classifier of Chapter 5 in terms of performance over this data set. The results in Table 7.2 show that this classifier takes slightly longer than the Naive Bayes classifier, however. Despite this, classification of an individual data point still is carried out in less than one second on average on a tree with 32 decision nodes. These results show that the proposed protocol can be fast and accurate for private classification within medical applications.

7.9 Discussion

7.9.1 Computational Bottlenecks

The primary bottleneck during computation occurs due to the cost of homomorphic multiplication. During Algorithm 14, homomorphic multiplication is performed during node evaluation on each node. In particular, if k is the maximum number of values any feature of a data point X may take, then the number of homomorphic operations performed during each node evaluation is bounded above by k. A tree with N nodes therefore requires at most $n \cdot K$ homomorphic multiplication operations. This bottleneck could be avoided by implementing the protocol in the unencrypted third-party private search model discussed in Chapter 6.

A bottleneck also occurs due to communication costs. During each node evaluation a number of communications are performed. During Oblivious Transfer, the Client communicates with the Model Owner two times and the Model Owner communicates with the Client one time. In addition to the communication cost of Oblivious Transfer, private node evaluation in Algorithm 15 requires 6 additional communications between the Client, Server, and Model Owner in the encrypted model, and 4 communications in the unencrypted model. Therefore on a tree with N nodes, communication must be performed $9 \cdot N$ times in the encrypted model and $6 \cdot N$ times in the unencrypted model.

To speed up performance, it is important that the Model Owner perform pre-processing on the data. Properly pre-processed data could result in a smaller tree, and therefore in a faster classification time. In particular, feature selection should be implemented on the data in order to reduce the dimensionality before training a learned model. Feature selection is a powerful method to implement during model construction to improve the model's performance by identifying only the most relevant features within the data [39]. Feature selection can be carried out in a variety of ways, depending on the data set. Feature selection covers a wide variety of algorithms which include filters, wrapper methods, and embedded methods [37].

7.9.2 Comparison

Khedr et al. explain that their classifier should perform decision tree classification on a tree with four nodes in approximately 3.477 milliseconds [41]. This is an estimate they approximated based on the multiplication running time of their protocol, and no tests were implemented. A full implementation of their protocol on a tree of comparable size would be necessary in order to provide a true comparison of results.

Wu et al. perform private decision tree classification on a tree with 12 decision nodes in approximately 0.545 seconds [68]. Bost et al. perform decision tree classification using encryption methods which are not fully homomorphic. They report average running times on a 4 node tree of 1.579 seconds for the Client and 0.798 seconds for the Server, and on a 6 node tree they report running times of 2.297 seconds for the Client and 1.723 seconds for the Server. The results in Table 7.2 show speedup over these times.

7.10 Conclusion

Third-party private search can be implemented in conjunction with various cryptographic methods in order to perform efficient, private classification using classification trees. Future work could extend this method to implement the large database variant of TPPS given in Algorithm 12, as well as implementation of random forest.

Chapter 8

Conclusions

Machine learning and cryptography can be combined in order to implement a variety of multi-party computational tasks such as third-party private search and private classification. These methods can be efficient for computation over real-world medical data. In particular, private-key fully homomorphic encryption can be efficiently implemented for classification tasks.

The implementation of Gribov-Kahrobaei-Shpilrain (GKS) encryption for private-key fully homomorphic encryption was presented with multiple speed improvements. Single-Instruction Multiple Data (SIMD) parallelization was implemented in GKS in order to allow for computation of multiple values at one time via multiple encodings within a single ciphertext. In total, up to 2^n elements may be encoded in a single plaintext via a fully homomorphic embedding for a plaintext ring with n generators. Furthermore, an algorithm for generating the required parameters and change-of-basis transformations for implementation of the GKS cryptosystem was described. Experimental performance results show that the GKS cryptosystem can be efficiently implemented via C++ using an arbitrary precision arithmetic library.

Private Naive Bayes classification via private-key FHE was outlined and implemented over real-world medical data. Classification of real-world medical data via private-key FHE using Naive Bayes was carried out in less than half of a second. Furthermore, private Decision Tree classification was carried out over an encrypted model in under one second.

A number of other protocols with potential use in future applications were developed within this document. A privacy-preserving `argmax` protocol was outlined that enables classification using a Naive Bayes model via private-key FHE. A third-party private search algorithm was outlined that allows a Client to privately and efficiently perform a membership query of a database without being given direct access to the database.

8.1 Future Work

The techniques outlined in this paper could be utilized on their own or in conjunction with similar techniques in order to build tools that allow clinicians to privately classify their patients' data using models, which they cannot access in the clear. Proper pre-processing techniques could create stronger and more efficiently computed models. Furthermore, medical researchers may use these models in order to determine if their models are over-fit to their own data sets.

Further classification models of interest are Support Vector Machine (SVM) and deep learning methods. Privacy-preserving classification methods implementing private-key fully homomorphic encryption methods could be developed via these models and implemented in the clinical setting along with the models described in this work. Furthermore, various privacy-preserving bioinformatics techniques could be explored, and the privacy-preserving decision tree classification could be extended to random forests.

Applications of interest for further study outside of the medical setting include personal security as well as national security settings. Personal security protocols using fully homomorphic encryption could be spam filters for e-mail and private data mining of individual online behavior. Law enforcement could implement a version of third-party private search in order to allow law enforcement officials to query a database stored in encrypted format without access to that database or the decryption key.

Appendices

Appendix A

Third Party Private Search Parameters

The Third Party Private Search protocol proposed in Chapter 6 includes a prediction error in the form of false positives. Recall that the protocol implements three parameters: n, the number of elements in the input database; q_1, the maximum bit size of values in the database; and q_2, the bit size of the one-time pad implemented by the Client.

The values in the tables below display the fallout, given by

$$\text{fallout} = \frac{\#\text{false positives}}{\#\text{false positives} + \#\text{true negatives}}.$$

This measure is chosen because during the protocol, no false negatives will occur. It is necessary that parameters be chosen to avoid the case of false positives.

Experiments were run using the Monte Carlo method with various inputs for all three parameters. During each test, $100,000$ experiments were performed. Algorithm 18 shows the pseudocode for the experiments. In summary, for each experiment a database with n random elements of at most q_1 bits was generated for the Database Owner, and the Client's value was set to another q_1 bit integer. The Client's value was chosen randomly under the constraint that the Client's value does not appear in the database. Then, a random q_2-bit integer was chosen as the Client's one-time pad and the third-party private search protocol was performed. The experiment was implemented via Python 3.6.

Algorithm 18 Fallout Error Estimation

Input: Parameters n, q_1 and q_2. **Output:** A boolean value 1 denoting a false positive or 0 denoting a true negative.

```
Populate a random database $D = \{d_i\}_{i=1}^{n}$ with $n$ random integers between 1 and $2^{q_1}$.
Pick a random value $c$ between 1 and $2^{q_1}$ such that $c \notin D$.
Pick a random $q_2$-bit value $r$.
Compute the product
    $$p = \prod_{i=1}^{n} (c + r - d_i).$$
if $r|p$ then
    return 1.
else
    return 0
end if
```

Tables A.1 and A.2 show the fallout resulting for input parameters n, q_1, and q_2.

$\mathbf{q_2}$	$\mathbf{q_1 = 2}$	$\mathbf{q_1 = 3}$	$\mathbf{q_1 = 4}$	$\mathbf{q_1 = 5}$	$\mathbf{q_1 = 6}$	$\mathbf{q_1 = 7}$	$\mathbf{q_1 = 8}$	$\mathbf{q_1 = 9}$
3	0.41460	0.72118	0.86253	0.89877	0.91328	0.92093	0.92449	0.92553
4	0.22381	0.52138	0.70157	0.78353	0.81255	0.82663	0.83065	0.83624
5	0.10395	0.313	0.46225	0.56652	0.62425	0.65258	0.66602	0.67098
6	0.03768	0.172	0.30843	0.40693	0.47366	0.50564	0.52955	0.53678
7	0.01547	0.09118	0.18744	0.27385	0.3303	0.37508	0.39686	0.40954
8	0.00596	0.04517	0.11069	0.17915	0.22918	0.26606	0.28726	0.30213
9	0.00183	0.02085	0.05927	0.10907	0.14979	0.18012	0.20118	0.20953
10	0.00067	0.00968	0.03355	0.06601	0.0947	0.11892	0.13754	0.14909
11	0.00014	0.00405	0.01725	0.03872	0.05983	0.07888	0.09183	0.10067
12	0	0.00186	0.00957	0.02123	0.03717	0.04983	0.06008	0.06735
13	0.00001	0.0007	0.00426	0.01153	0.02259	0.03105	0.03971	0.04398
14	0	0.00025	0.00209	0.00603	0.01226	0.01853	0.02425	0.02889
15	0	0.0001	0.00102	0.00387	0.00722	0.01186	0.01613	0.01871
16	0	0.00005	0.0005	0.00157	0.00391	0.00711	0.00965	0.01156
17	0	0.00001	0.00015	0.00085	0.00212	0.00351	0.0058	0.0072
18	0	0.00001	0.0001	0.00054	0.00119	0.00223	0.00276	0.0044
19	0	0	0.00001	0.0002	0.00068	0.00115	0.00189	0.00268
20	0	0	0.00001	0.000011	0.00039	0.00075	0.001	0.00153
21	0	0	0	0.00004	0.0002	0.00026	0.00053	0.00091
22	0	0	0	0.00003	0.00006	0.00012	0.00043	0.00052
23	0	0	0	0.00001	0.00003	0.00013	0.00016	0.00025
24	0	0	0	0	0.00005	0.00008	0.00007	0.00019
25	0	0	0	0	0	0.00004	0.00005	0.00006
26	0	0	0	0	0	0	0.00003	0.00005
27	0	0	0	0	0.00001	0.00001	0	0.00004
28	0	0	0	0	0	0	0.00001	0.00003
29	0	0	0	0	0	0	0	0.00001
30	0	0	0	0	0	0	0	0.00001
31	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0
34	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0

Table A.1: Fallout for $n = 10$

$\mathbf{q_2}$	$\mathbf{q_1 = 10}$	$\mathbf{q_1 = 11}$	$\mathbf{q_1 = 12}$	$\mathbf{q_1 = 13}$	$\mathbf{q_1 = 14}$	$\mathbf{q_1 = 15}$	$\mathbf{q_1 = 16}$	$\mathbf{q_1 = 17}$
1	0.99951	0.99944	0.99954	0.9994	0.99951	0.99953	0.99954	0.99937
2	0.9917	0.99218	0.99232	0.99229	0.99204	0.99184	0.99193	0.99179
3	0.92762	0.92759	0.92747	0.9266	0.92638	0.92845	0.928	0.92831
4	0.83613	0.83924	0.8383	0.84121	0.84102	0.83971	0.83775	0.83847
5	0.67544	0.67629	0.68143	0.67705	0.67762	0.67795	0.67924	0.67743
6	0.54115	0.54548	0.54469	0.54435	0.54454	0.54413	0.54639	0.54424
7	0.41004	0.41331	0.41516	0.41628	0.41832	0.41945	0.41608	0.41557
8	0.30585	0.31239	0.31338	0.31307	0.31209	0.31094	0.31166	0.31321
9	0.22153	0.22481	0.2257	0.22458	0.22748	0.23065	0.22765	0.22604
10	0.15357	0.15952	0.15982	0.16164	0.16195	0.16376	0.16297	0.16129
11	0.10642	0.1118	0.11059	0.11336	0.11524	0.11457	0.11459	0.11353
12	0.07145	0.07646	0.07802	0.08019	0.07843	0.07957	0.07935	0.08035
13	0.04811	0.05156	0.05182	0.05294	0.0539	0.05299	0.05415	0.05276
14	0.03175	0.03241	0.03556	0.03594	0.03569	0.03709	0.0358	0.03773
15	0.0207	0.02135	0.02363	0.0235	0.02402	0.0238	0.02505	0.02495
16	0.01285	0.01476	0.0148	0.0154	0.01509	0.01694	0.01602	0.01656
17	0.00844	0.00913	0.00959	0.00961	0.01069	0.01059	0.01075	0.01047
18	0.00523	0.00526	0.00611	0.00564	0.0066	0.00681	0.00654	0.00719
19	0.00286	0.00339	0.00357	0.00408	0.00414	0.00444	0.00423	0.0041
20	0.00168	0.00214	0.00243	0.00235	0.00297	0.00287	0.00267	0.00268
21	0.00095	0.00105	0.00157	0.00183	0.00145	0.00151	0.0019	0.00175
22	0.00074	0.0008	0.00097	0.00092	0.0009	0.00094	0.00103	0.00119
23	0.00039	0.0003	0.00055	0.00059	0.00055	0.00065	0.00066	0.0006
24	0.00021	0.00026	0.00036	0.00032	0.0004	0.00039	0.0004	0.00052
25	0.00011	0.00017	0.00023	0.00016	0.0002	0.00022	0.0003	0.00021
26	0.00002	0.00009	0.00007	0.00012	0.0001	0.00015	0.0001	0.00015
27	0.00003	0.00002	0.00008	0.00003	0.00006	0.0001	0.0001	0.00011
28	0.00002	0.00004	0.00004	0.00001	0.00003	0.00005	0.00002	0.00004
29	0.00001	0.00002	0.00002	0.00001	0.00005	0.00003	0.00006	0.00007
30	0	0.00001	0.00002	0.00002	0.00002	0.00001	0.00003	0.00001
31	0	0.00001	0.00001	0	0	0.00001	0.00001	0.00002
32	0	0	0	0	0	0.00001	0	0.00001
33	0	0.00001	0	0	0.00001	0.00002	0	0
34	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0
36	0	0	0	0	0	0	0	0
37	0	0	0	0	0	0	0	0
38	0	0	0	0	0	0	0	0
39	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0
41	0	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0	0
43	0	0	0	0	0	0	0	0
44	0	0	0	0	0	0	0	0
45	0	0	0	0	0	0	0	0
46	0	0	0	0	0	0	0	0
47	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0
49	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0

Table A.2: Fallout for $n = 10$, continued

Bibliography

[1] Rakesh Agrawal and Ramakrishnan Srikant. ‘Privacy-preserving Data Mining’. In: *SIGMOD Rec.* 29.2 (May 2000), pp. 439–450.

[2] Jacob Alperin-Sheriff and Chris Peikert. ‘Faster Bootstrapping with Polynomial Error’. In: *Advances in Cryptology – CRYPTO 2014*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Aug. 2014, pp. 297–314.

[3] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. ‘PIR with Compressed Queries and Amortized Query Processing’. In: *2018 IEEE Symposium on Security and Privacy (SP)* (2018), pp. 962–979.

[4] Jean-Claude Bajard, Julien Eynard, M. Anwar Hasan, and Vincent Zucca. ‘A Full RNS Variant of FV Like Somewhat Homomorphic Encryption Schemes’. In: *Selected Areas in Cryptography – SAC 2016*. Ed. by Roberto Avanzi and Howard Heys. Cham: Springer International Publishing, 2017, pp. 423–442.

[5] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. ‘Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme’. In: *Cryptography and Coding: 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*. Ed. by Martijn Stam. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 45–64.

[6] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. ‘Machine Learning Classification over Encrypted Data’. In: Symposium on Network and Distributed System Security (NDSS). 2015.

[7] Zvika Brakerski. ‘Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP’. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 868–886. ISBN: 978-3-642-32009-5.

[8] Zvika Brakerski and Vinod Vaikuntanathan. ‘Efficient Fully Homomorphic Encryption from (Standard) LWE’. In: *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. FOCS ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 97–106.

[9] Zvika Brakerski and Vinod Vaikuntanathan. ‘Fully Homomorphic Encryption from ring-LWE and Security for Key Dependent Messages’. In: *Proceedings of the 31st Annual Conference on Advances in Cryptology.* CRYPTO’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 505–524.

[10] Zvika Brakerski and Vinod Vaikuntanathan. ‘Lattice-based FHE As Secure As PKE’. In: *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science.* ITCS ’14. Princeton, New Jersey, USA: ACM, 2014, pp. 1–12.

[11] Zvika Brakerski, Vinod Vaikuntanathan, and Craig Gentry. ‘Fully homomorphic encryption without bootstrapping’. In: *In Innovations in Theoretical Computer Science.* 2012.

[12] Leo Breiman, Jerome Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees.* Statistics/Probability Series. Belmont, California, U.S.A.: Wadsworth Publishing Company, 1984.

[13] Rahul C. Deo. ‘Machine Learning in Medicine’. In: 132 (20 Nov. 2015), pp. 1920–1930.

[14] Jan Camenisch, Markulf Kohlweiss, Alfredo Rial, and Caroline Sheedy. ‘Blind and Anonymous Identity-Based Encryption and Authorised Private Searches on Public Key Encrypted Data’. In: *Public Key Cryptography – PKC 2009.* Ed. by Stanisław Jarecki and Gene Tsudik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 196–214.

[15] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. ‘Private Information Retrieval’. In: *J. ACM* 45.6 (Nov. 1998), pp. 965–981.

[16] Tung Chou and Claudio Orlandi. ‘The Simplest Protocol for Oblivious Transfer’. In: *Progress in Cryptology – LATINCRYPT 2015.* Ed. by Kristin Lauter and Francisco Rodríguez-Henríquez. Cham: Springer International Publishing, 2015, pp. 40–58.

[17] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. ‘Fully Homomorphic Encryption over the Integers with Shorter Public Keys’. In: *Proceedings of the 31st Annual Conference on Advances in Cryptology.* CRYPTO’11. Santa Barbara, CA: Springer-Verlag, 2011, pp. 487–504.

[18] R. Cypher and J. L. C. Sanz. ‘SIMD architectures and algorithms for image processing and computer vision’. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.12 (1989), pp. 2158–2174.

[19] Emiliano De Cristofaro, Yanbin Lu, and Gene Tsudik. ‘Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information’. In: *Trust and Trustworthy Computing.* Ed. by Jonathan M. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, Angela Sasse, and Yolanta Beres. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 239–253.

[20] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. ‘Fully Homomorphic Encryption over the Integers’. en. In: *Advances in Cryptology – EUROCRYPT 2010.* Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, May 2010, pp. 24–43.

[21] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. *Manual for Using Homomorphic Encryption for Bioinformatics*. Microsoft Research, 2015. URL: `https://www.microsoft.com/en-us/research/publication/manual-for-using-homomorphic-encryption-for-bioinformatics/`.

[22] Cynthia Dwork. 'Differential Privacy'. In: *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–12.

[23] T. ElGamal. 'A public key cryptosystem and a signature scheme based on discrete logarithms'. In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 469–472.

[24] Daniel Faggella. *7 Applications of Machine Learning in Pharma and Medicine*. July 2018. URL: `https://www.techemergence.com/machine-learning-in-pharma-medicine/`.

[25] Jenfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. `http://eprint.iacr.org/2012/144`. 2012.

[26] Craig Gentry. 'Computing Arbitrary Functions of Encrypted Data'. In: *Commun. ACM* 53.3 (Mar. 2010), pp. 97–105.

[27] Craig Gentry. 'Fully Homomorphic Encryption Using Ideal Lattices'. In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC '09. ACM, 2009, pp. 169–178.

[28] Craig Gentry and Shai Halevi. *Fully Homomorphic Encryption without Squashing Using Depth-3 Arithmetic Circuits*. Cryptology ePrint Archive, Report 2011/279. `http://eprint.iacr.org/2011/279`. 2011.

[29] Craig Gentry, Shai Halevi, and Nigel P. Smart. 'Better Bootstrapping in Fully Homomorphic Encryption'. In: *Public Key Cryptography – PKC 2012: 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*. Ed. by Marc Fischlin, Johannes Buchmann, and Mark Manulis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–16.

[30] Craig Gentry, Shai Halevi, and Nigel P. Smart. 'Fully Homomorphic Encryption with Polylog Overhead'. In: *Advances in Cryptology – EUROCRYPT 2012*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Apr. 2012, pp. 465–482.

[31] Craig Gentry, Shai Halevi, and Nigel P. Smart. 'Homomorphic Evaluation of the AES Circuit'. In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Springer Berlin Heidelberg, 2012, pp. 850–867.

[32] Craig Gentry, Amit Sahai, and Brent Waters. 'Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based'. In: *CRYPTO*. Springer, 2013, pp. 75–92.

[33] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 'CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy'. In: *PMLR*. June 2016, pp. 201–210.

[34] Shafi Goldwasser and Silvio Micali. 'Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information'. In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. STOC '82. New York, NY, USA: ACM, 1982, pp. 365–377.

[35] Thore Graepel, Kristin Lauter, and Michael Naehrig. 'ML Confidential: Machine Learning on Encrypted Data'. In: *Information Security and Cryptology – ICISC 2012: 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*. Ed. by Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–21.

[36] Alexey Gribov, Delaram Kahrobaei, and Vladimir Shpilrain. 'Private-Key Fully Homomorphic Encryption in Rings'. In: *Groups, Complexity, Cryptology* 10.1 (2018), pp. 17–27.

[37] Isabelle Guyon and André Elisseeff. 'An Introduction to Variable and Feature Selection'. In: *The Journal of Machine Learning Research* 3 (Mar. 2003), pp. 1157–1182.

[38] Shai Halevi and Victor Shoup. *Algorithms in HElib*. Cryptology ePrint Archive, Report 2014/106. `http://eprint.iacr.org/2014/106`. 2014.

[39] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Second Edition. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2009.

[40] Soni Jyoti, Ansari Ujma, Sharma Dipesh, and Soni Sunita. 'Predictive Data Mining for Medical Diagnosis: An Overview of Heart Disease Prediction'. In: *International Journal of Computer Applications* 17.8 (Mar. 31, 2011), pp. 43–48.

[41] A. Khedr, G. Gulak, and V. Vaikuntanathan. 'SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers'. In: *IEEE Transactions on Computers* 65.9 (2016), pp. 2848–2858.

[42] Igor Kononenko. 'Machine learning for medical diagnosis: history, state of the art and perspective'. In: *Artificial Intelligence in Medicine* 23.1 (Aug. 1, 2001), pp. 89–109. ISSN: 0933-3657.

[43] Kim Laine. *Simple Encrypted Arithmetic Library*. Version 2.3.1. Microsoft. 2017.

[44] *LIBLINEAR – A Library for Large Linear Classification*. URL: `https://www.csie.ntu.edu.tw/~cjlin/liblinear`.

[45] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[46] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 'On Ideal Lattices and Learning with Errors over Rings'. In: *J. ACM* 60.6 (Nov. 2013), pp. 1–35.

[47] Centers for Medicare & Medicaid Services. *The Health Insurance Portability and Accountability Act of 1996 (HIPAA).* Online at `http://www.cms.hhs.gov/hipaa/`. 1996.

[48] Pascal Paillier. 'Public-Key Cryptosystems Based on Composite Degree Residuosity Classes'. In: *Advances in Cryptology — EUROCRYPT '99.* Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, May 1999, pp. 223–238.

[49] Vasilis Pappas, Mariana Raykova, Binh Vo, Steven M. Bellovin, and Tal Malkin. 'Private search in the real world'. In: *Proceedings of the 27th Annual Computer Security Applications Conference Pages.* ACSAC '11. ACM Press, 2011, pp. 83–92.

[50] Chris Peikert. 'A Decade of Lattice Cryptography'. In: *Foundations and Trends in Theoretical Computer Science* 10.4 (Mar. 2016), pp. 283–424.

[51] Huy Nguyen Anh Pham and Evangelos Triantaphyllou. 'The Impact of Overfitting and Overgeneralization on the Classification Accuracy in Data Mining'. In: *Soft Computing for Knowledge Discovery and Data Mining.* Ed. by Oded Maimon and Lior Rokach. Boston, MA: Springer US, 2008, pp. 391–431.

[52] D Powers. 'Evaluation: From precision, recall and fmeasure to roc, informedness, markedness and correlation'. In: *Journal of Machine Learning Technologies* 2 (2011), pp. 37–63.

[53] Michael O. Rabin. *How to exchange secrets with oblivious transfer.* Tech. rep. TR-81. Aiken Computation Lab, Harvard University, 1981.

[54] Y. Rahulamathavan, R. C. Phan, S. Veluru, K. Cumanan, and M. Rajarajan. 'Privacy-Preserving Multi-Class Support Vector Machine for Outsourcing the Data Classification in Cloud'. In: *IEEE Transactions on Dependable and Secure Computing* 11.5 (Sept. 2014), pp. 467–479.

[55] M. Raykova, A. Cui, B. Vo, B. Liu, T. Malkin, S. M. Bellovin, and S. J. Stolfo. 'Usable, Secure, Private Search'. In: *IEEE Security Privacy* 10.5 (2012), pp. 53–60.

[56] Oded Regev. 'On Lattices, Learning with Errors, Random Linear Codes, and Cryptography'. In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing.* STOC '05. Baltimore, MD, USA: ACM, 2005, pp. 84–93.

[57] Ronald Rivest, Len Adleman, and Michael Dertouzos. 'On Data Banks And Privacy Homomorphisms'. In: *Foundations of Secure Computation* 4.11 (1978), pp. 165–179.

[58] Ronald L. Rivest. 'Cryptography and machine learning'. en. In: *Advances in Cryptology — ASIACRYPT '91.* Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Nov. 1991, pp. 427–439.

[59] S. Halevi. *HElib: An Implementation of homomorphic encryption.* 2013. URL: `https://github.com/shaih/HElib`.

[60] Nigel P. Smart and Frederik Vercauteren. 'Fully Homomorphic SIMD Operations'. In: *Des. Codes Cryptography* 71.1 (Apr. 2014), pp. 57–81.

[61] X. Sun, P. Zhang, J. K. Liu, J. Yu, and W. Xie. ‘Private machine learning classification based on fully homomorphic encryption’. In: *IEEE Transactions on Emerging Topics in Computing* (2018). Early access.

[62] *The GNU MP Bignum Library*. URL: `https://gmplib.org/`.

[63] Jaideep Vaidya, Hwanjo Yu, and Xiaoqian Jiang. ‘Privacy-preserving SVM classification’. In: *Knowledge and Information Systems* 14.2 (2008), pp. 161–178.

[64] Thijs Veugen. *Comparing encrypted data*. 2011. URL: `http://msp.ewi.tudelft.nl/sites/default/files/Comparingencrypteddata.pdf`.

[65] Thijs Veugen. ‘Encrypted Integer Division and Secure Comparison’. In: *Int. J. Appl. Cryptol.* 3.2 (June 2014), pp. 166–180.

[66] Shuang Wang, Noman Mohammed, and Rui Chen. ‘Differentially private genome data dissemination through top-down specialization’. In: *BMC Medical Informatics and Decision Making* 14.1 (2014).

[67] W H Wolberg and O L Mangasarian. ‘Multisurface method of pattern separation for medical diagnosis applied to breast cytology.’ In: *Proceedings of the National Academy of Sciences of the United States of America* 87.23 (Dec. 1990), pp. 9193–9196.

[68] David J. Wu, Tony Feng, Michael Naehrig, and Kristin Lauter. ‘Privately Evaluating Decision Trees and Random Forests’. In: *Proceedings on Privacy Enhancing Technologies* 2016.4 (2016), pp. 335–355.

Autobiographical Statement

I was born in Dallas, Texas, and grew up in Dallas, Texas, and The Woodlands, Texas. After completing my primary and secondary education, I attended DePaul University in Chicago, Illinois. I received numerous awards and scholarships and served as Vice President of the Math Club. I graduated summa cum laude with a Bachelor of Arts in Pure Mathematics in 2012. My interest in research was sparked at this time during undergraduate mathematics research projects: One in combinatorics at DePaul University, and another in computational number theory at the Rose-Hulman Institute of Technology's Mathematics Research Experience for Undergraduates program.

I continued with graduate education in Pure Mathematics at DePaul University and at The Graduate Center, The City University of New York (CUNY). I passed graduate-level qualifying examinations in Analysis and in Algebra at DePaul University, as well as in Algebra and in Topology at The Graduate Center, CUNY. In 2015, I received a Master of Science with distinction in Pure Mathematics from DePaul University.

I matriculated into the Computer Science Ph.D. program at The Graduate Center, CUNY, in 2016, and received a Master of Philosophy in Computer Science in 2017. During this time I served as Chairperson of the Computer Science Students' Association and was the elected student representative on the Executive, Curriculum, and Elections Committees. I also served as the Computer Science Student Representative within the Doctoral Student Council and the Graduate Council, and co-organized the weekly Graduate Center

Crypto-Math Seminar.

My graduate research started in computational group theory and complexity theory within cryptography. To receive my Master of Philosophy in Computer Science, I presented a survey on the generic-case complexity of the conjugacy search problem in platform groups for non-commutative cryptosystems. I continued research in cryptography, where I researched fully homomorphic encryption. I was motivated then to apply my work to a problem with real-world implications, and my research turned to the intersections between cryptography, machine learning, and medical classification. I performed my doctoral research on utilizing fully homomorphic encryption for classification algorithms over medical data.

I will continue this research as a Postdoctoral Researcher at the Biomedical and Clinical Informatics Lab in the Department of Computational Medicine and Bioinformatics at the University of Michigan beginning October 2018. I will be involved in cryptographic research as well as the development of image processing techniques applied to segmentation, classification, and diagnosis using medical data.

Concurrent with my research, I pursued opportunities to teach mathematics and computer science at the undergraduate level. I have taught courses at New York University, New York City College of Technology, and John Jay College of Criminal Justice. Courses I taught include Python for Engineering, Cryptography and Cryptanalysis, and Calculus. Teaching allowed me to grow as a scholar and a mentor and provided an invaluable and rewarding experience.

In addition to my academic work, I co-organized the first LGBTQ panel at an official Star Trek convention during Star Trek Mission New York in 2016, titled “Queer Trekkers: Gene Roddenberry’s Galactic Utopias.”

I am inspired in my future work to continue to seek real-world applications which could make a positive impact on people’s lives. I look forward to the challenges and opportunities available in my future career.