이학박사 학위논문

Secure Computation via Homomorphic Encryption

(동형암호를 이용한 안전한 연산)

2017년 8월

서울대학교 대학원

수리과학부

정 희 원

Secure Computation via Homomorphic Encryption

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Heewon Chung

Dissertation Director : Professor Jung Hee Cheon

Department of Mathematical Sciences
Seoul National University

August 2017

Abstract

(Fully) Homomorphic encryption (FHE, HE) is one of the natural and powerful tools for ensuring privacy of sensitive data since it enables to handle ciphertexts without decryption and thus allow complicated computations on the encrypted data. Due to this property, homomorphic encryption can be applied to many scenarios in the real life, especially, databases. Until now, most of homomorphic encryption schemes restrict a plaintext space as an integer and thus numeric data should be represented by integers. However, there are many applications working in the real number system that operate on very sensitive information, for example, user's location information and patient's medical information. Usually, these information can be represented by the real numbers and thus it should be encoded into the integers. The general decimal representation requires *quite large* plaintext space and a polynomial representation also requires a *higher degree* of polynomials, which has a bad influence to the performance of FHE scheme.

In this thesis, we employ continued fraction to represent real numbers and to alleviate this inefficiency. With continued fraction, real numbers can be represented by a set of *quite small* integers and it makes performance improvement than other encoding techniques. Moreover, we can develop a set of algorithms and circuits using continued fraction for the following operations: homomorphic integer division, equality circuit and comparison circuits over the real numbers.

First, we suggest an algorithm for homomorphic integer division using continued fraction and restoring division algorithm. Since the integer is not closed under the division, the most of homomorphic encryption schemes cannot support the division, however, we suggest a transformation from rational numbers to continued fractions being encrypted and it allows to divide two encrypted integers. Further, we can evaluate a polynomial whose coefficients

are in the rational numbers.

Second, we describe comparison circuits over the encrypted real numbers including equality circuits. Since comparing two continued fraction is also easy as much as comparing two decimal numbers, we can build *more efficient* comparison circuits while maintaining the small message space utilizing the homomorphic comparison circuits over the integers. With our efficient comparison circuits, we can apply to the real-type database which indicates each numeric data is represented by the real numbers and our circuits enable to sorting and private database queries such as retrieval queries and aggregate queries, which makes database useful.

Finally, we present a proof of correct decryption in a single party homomorphic encryption. Although a server evaluates some polynomial being encrypted, the server cannot know any information about the result. Thus, if a server is interested in the result, a data owner returns the decryption result. The problem is that the server should believe the data owner at this time because the data owner can manipulate the decryption result and the server cannot recognize it. We prevent this situation by utilizing one-time message authentication code. Moreover, this technique can be applied to many scenarios, especially, a protocol for authentication of biometrics.

Keywords: homomorphic encryption, continued fraction, real number, database queries, authentication
Student Number: 2013-30900

Contents

Chapter 1

Introduction

If someone ask

> Given $\texttt{Enc}(x)$ and a polynomial $f(x)$, can you evaluate $\texttt{Enc}(f(x))$?

the answer is yes. A homomorphic encryption has a homomorphism property and thus given $\texttt{Enc}(x)$ and $\texttt{Enc}(y)$, one can easily compute $\texttt{Enc}(x+y)$ and $\texttt{Enc}(xy)$. Consequently, every polynomial and a circuit consisting of additions and multiplications can be evaluated being encrypted with homomorphic encryption. Thus, it is one of the natural and powerful tools for ensuring privacy of sensitive data since it enables to handle ciphertexts without decryption and thus allows complicated computations on the encrypted data. This concept is introduced by Rivest *et al.* [68] and in 2009, Gentry firstly construct a secure homomorphic encryption scheme [42] based on the ideal lattice. However, this scheme exhibits.a fairly poor performance. A later series of results were proposed to address this concern. In particular, Brakerski and Vaikuntanathan introduced the concept of leveled FHE which allows the evaluation of arbitrary circuits of polynomial depth [13]. Following this proposal, Brakerski, Gentry, and Vaikuntanathan (BGV) in [12] further presented a leveled FHE scheme with significantly improved performance.

1.1 Overview and Contributions

This thesis covers three different topics which are applications of homomorphic encryption. My proposal can be applied to the most existing homomorphic encryption scheme even if there exists more appropriate homomorphic encryption scheme for each applications. Hence, in this thesis, I do not target any specific scheme and I will not present a detail of the scheme. However, my implementation is build on the BGV scheme because it is not only supports SIMD operations but also stably supported by `HElib` [47], which is a widely used software library of BGV scheme.

We can summary contributions of this thesis in three-folds.

1.1.1 Homomorphic Integer Division

Even though homomorphic encryption can handle ciphertexts, it does not support dividing two encrypted integers because a set of integers is not closed under the division. Until now, the most existing homomorphic encryption scheme supports an integer plaintext space and thus we need to represent a real number as a set of integers (or an integer) to utilize the existing scheme. To do that, we use continued fraction which enables to represent a real number as a set of integers.

The next step is replacing a division algorithm to be executed with homomorphic encryption. Typically, the simplest way to replacing division algorithm is by subtracting repeatedly divisor from the dividend. However, a terminated condition is that subtracting result is less than divisor and since these are encrypted, we cannot distinguish some value satisfy the termination condition Hence, it cannot replace a division algorithm when employing homomorphic encryption. For this reason, we choose the restoring division algorithm which outputs a quotient and a remainder. The main idea of restoring division algorithm is

$$p_{i+1} = b \times p_i - q_{n-(j+1)} \times d$$

where p_i is i-th the partial remainder of the division, b is the radix, n is the number of digits in the quotient, q_i is the i-th digits of the quotient and d is the divisor. This algorithm is also an iterative algorithm, but the number of iterations are fixed as much as the size of a numerator. Moreover, it also contains a step for comparing an encrypted integer and zero, but it can be replaced by determining a sign of a integer when we use a binary field. For example, if a underlying message is positive, then the most significant bit of the integer should zero and if a underlying message is negative, then the most significant bit of the integer should be one. Therefore, we have to set $R = 2$ in order to replace comparing step.

The last step is expressing a real number to continued fraction being encrypted using restoring division algorithm. In fact, continued fraction and euclidean algorithm are based on the same principle. Euclidean algorithm is very efficient algorithm which enables to compute greatest common divisors of two integer. The fundamental is that the greatest common divisor of numbers does not change even if the larger number is subtracted from the other number. Since the subtraction reduce the large number, repeating this step outputs successively smaller pairs of numbers and finally the size of two numbers becomes equal and the number is the greatest common divisor of the original two integers. Since a remainder for two integers can be obtained by successively subtracting the smaller numbers to the larger number, computing a remainder is reduced the iteration number instead of a subtraction. In this sense, both euclidean algorithm and continued fraction require a step to compute a quotient and a remainder for two original integers. Since each partial quotient corresponds to a quotient for each iteration in euclidean algorithm, one can compute every partial quotient using euclidean algorithm. Moreover, restoring division algorithm outputs a quotient and a remainder for two integers and thus we can divide two integers and a output can be represented in continued fraction form by executing several restoring division algorithm.

Moreover, we can evaluate a polynomial over $\mathbb{Q}$ being encrypted. Our

encoding techniques enable to handle encrypted rational numbers as well as encrypted integers. Since the evaluation of any polynomial over $\mathbb{Q}$ is also in $\mathbb{Q}$, the polynomial over $\mathbb{Q}$ can be evaluated using our technique. Similarly, any arithmetic of two polynomials over $\mathbb{Q}$ is also a polynomial over $\mathbb{Q}$, we can evaluate it being encrypted.

1.1.2 Homomorphic Comparisons over the Real Numbers

We provide homomorphic comparison circuits including equality circuits over the real numbers. In the paper [25,26], the authors provide an equality circuit and a comparison circuit for fully homomorphic encryption using boolean circuits with data represented by bits. In order to tell which of two real numbers is larger with homomorphic encryption, these are should be encoded into integers because the most homomorphic encryption scheme only supports an integer message space. There are several encoding techniques: one is using the decimal representation and another method is an encoding to a polynomial. However, these techniques have a step of removing a precision by multiplying a prefix numerator such as a power of 10 and it makes large integer. For example, suppose a precision is 5 and any real number is approximated to a rational number of 5 places of decimals. One can make an integer by multiplying 10^5 to this rational number and the integer should be at least 16 bit integers because $10^5 \approx 2^{16}$. It means that the plaintext space should be large as much as 16 bits and if there are some multiplications, then the plaintext space should be much larger in order to prevent overflow the message. Since there are some parameters depending on the plaintext space, these parameters should be large, for example, the degree of cyclotomic polynomial and a modulus for a ciphertext space. Furthermore, a multiplicative depth consumes more than 1 even though the only one multiplication are performed, which implies that more multiplicative depth are required than theoretical multiplicative depth. The same problem occurs in the polynomial encoding.

In the same example, a real number is encoded into a polynomial of degree at least 16 and a degree of modulus polynomial should be enough large. Such large parameters leads to the poor performance.

For this reason, we employ continued fraction which enable to represent a real number as a set of integers. Let $X = [x_0; x_1, \ldots, x_{n-1}]$ be a continued fraction of $p/q \in Q$, where $X \in \mathbb{Q}$ and $x_i \in \mathbb{Z}$. In this case, we can easily derive that

$$\log x_0 + \log x_1 + \ldots + \log x_{n-1} < logp$$
$$\log x_0 + \log x_1 + \ldots + \log x_{n-1} < logq$$

and it implies that each partial quotient's size is relatively smaller than p and q. Because of the relatively small, this encoding strategy has advantages in terms of the size of parameter for homomorphic encryption. The performance of homomorphic encryption totally depends on the parameter, especially, the plaintext space. In addition, the previous encoding techniques requires to keep track of the precision in order to recover the real number. However, continued fraction is independent from the precision and thus we do not need to keep track of the precision. In this sense, continued fraction is one of the best encoding technique.

Beside the size of partial quotient, there are fascinating properties for continued fraction and what I am focusing is that it is easy to compare which of two continued fractions represent the larger number. We can easily compare two real numbers with decimal representation: just compare left-to-right since a digit dominates all other digits which is positioned on the right. Similarly, in the case of continued fraction, a partial quotient dominates all other partial quotients which is positioned on the right and thus the first different partial quotient determines the order. The only difference is that even-numbered partial quotient It could be summarized like this: Let $X = [x_0; x_1, \ldots, x_{n-1}]$ and $Y = [y_0; y_1, \ldots, y_{m-1}]$. Let k be the smallest index for which x_k is unequal to y_k. Then,

$$X < Y \text{ if } (-1)^k(x_k - y_k) < 0$$

and $X > Y$ otherwise. If there is no such k which implies $x_i = y_i$ for all $i < n$ and $n < m$, then $X < Y$ if n is odd and $X > Y$ if n is even. If there is no such k and $n = m$, then clearly $X = Y$. Now, we can translate it to the ciphertext domain using homomorphic encryption and give concrete constructions. For equality circuit, it should be the same for all partial quotients and thus it can be translated to

$$\mathtt{EQ}_{\mathbb{R}}(\bar{X}, \bar{Y}) = \prod_{i=0}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_i, \bar{y}_i)$$

For comparison circuits, at first we need to find the smallest index of different partial quotient. However, it cannot be done in the ciphertext domain and thus we should consider the worst case, which implies that we need to consider the case that the last partial quotient is different.

$$\begin{aligned}
\mathtt{GT}_{\mathbb{R}}(\bar{X}, \bar{Y}) =& \mathtt{GT}_{\mathbb{Z}}(\bar{x}_{n-1}, \bar{y}_{n-1}) + \sum_{i=0}^{\frac{n-2}{n}} \mathtt{GT}_{\mathbb{Z}}(\bar{x}_{2i}, \bar{y}_{2i}) \cdot \prod_{j \geq 2i}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j, \bar{y}_j) \\
&+ \sum_{i=0}^{\frac{n-2}{n}} \mathtt{LT}_{\mathbb{Z}}(\bar{x}_{2i+1}, \bar{y}_{2i+1}) \cdot \prod_{j \geq 2i+1}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j, \bar{y}_j) \\
\mathtt{LT}_{\mathbb{R}}(\bar{X}, \bar{Y}) =& \mathtt{LT}_{\mathbb{Z}}(\bar{x}_{n-1}, \bar{y}_{n-1}) + \sum_{i=0}^{\frac{n-2}{n}} \mathtt{LT}_{\mathbb{Z}}(\bar{x}_{2i}, \bar{y}_{2i}) \cdot \prod_{j \geq 2i}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j, \bar{y}_j) \\
&+ \sum_{i=0}^{\frac{n-2}{n}} \mathtt{GT}_{\mathbb{Z}}(\bar{x}_{2i+1}, \bar{y}_{2i+1}) \cdot \prod_{j \geq 2i+1}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j, \bar{y}_j)
\end{aligned}$$

Using our comparison circuits, we can apply to the database service, especially, sorting and private database queries. Since the most sorting algorithm consists of comparison and swap and swap is based on the comparison, we can easily build a circuit for swap. Similarly, private database queries are mainly based on the equality circuit and the comparison circuit and thus it means that one can execute private database queries being encrypted.

1.1.3 Integrity of Homomorphic Evaluations

Suppose a data owner has data and he gives encrypted data to a data consumer. Until now, we only consider the scenario such that the data owner wants to evaluate some functions without leaking information about his data. However, there is a scenario such that the data consumer wants to know the result. If the data consumer wants to know the decryption result, the data owner should decrypt a ciphertext and give back to the data consumer, however, the data owner cannot guarantee the correct decryption so he will need a proof of correct decryption.

In order to resolve this problem, at first, we employ a technique of a message authentication code (MAC). The MAC is a short piece of information, called a tag, used to authenticate a message. In other words, the MAC guarantees the message came from the sender and has not been changed. The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content. In my case, the sender is the data owner and a tag is generated by the data consumer and thus the data consumer has a secret key for the MAC. The data consumer evaluates some polynomial being encrypted and a tag for the evaluation is generated by the data consumer. Since homomorphic encryption is used to encrypt data, the data owner can get a tag of the result, but he cannot learn anything about the result because he does not have a secret for the MAC. After the data owner send the tag of the result, the data consumer can recover the result using the secret key for the MAC. Moreover, to improve the performance, we define a tag function $\mathtt{Tag}$ and a verification function $\mathtt{Vrf}$: for a message x,

$$\mathtt{Tag}(x) = r_0 x + r_1$$

where r_0 and r_1 are random numbers and $\mathtt{Vrf}(\tau) = (\tau - r_1)/r_0$. We can easily check that $\mathtt{Vrf}(\mathtt{Tag}(x)) = x$. Whenever the data consumer generates a tag, these random numbers r_0 and r_1 should be always newly generated in order to secure against adversaries. If someone use only a multiplication or

an addition to generate a tag, it can be easily broken and thus we insist that our tag function is the optimized.

However, this approach provides a type of decryption oracle, which could clearly result in a security weakness and allow a malicious data consumer to recover the original data. It should be noted that homomorphic encryption is only secure against chosen plaintext attacks (CPA) and insecure against chosen ciphertext attacks (CCA). In order to prevent an undesirable situation where the decryption oracle is obtained, we also introduce a method for enclosing the tag using a one-way function with discrete logarithm (DL) settings and thus the data consumer can only learn the enclosed value of the tag. This prevents misuse of the decryption key when receiving a decryption query for its ciphertext.

Moreover, our technique can be applied to the biometric authentication, especially, iris authentication. Recently, there are many use cases using biometric such as building entrance. In this case, a server store employee's biometric information and this information is used to determine the validity. However, biometric information is one of sensitive personal data and thus it should be protected whenever used since it cannot be changed and replaced. Hence, it should be stored in a ciphertext form. An iris can be converted into 2,048 bit and a matching algorithm for iris is based on the hamming distance. Since homomorphic encryption can compute a hamming distance being encrypted, the server can obtain a hamming distance result, however, it is still encrypted. In order to determine the validity, the server should know the result and thus our techniques can be applied to this situation in order to protect from the invalidity visitor. In this thesis, we suggest a secure biometric authentication protocol with our suggestions and give a security proof of our construction.

Chapter 2

Preliminaries

2.1 Notation

For readability, we define some notations and terminologies for the rest of the paper. A bar over some integer means that the integer is encrypted by an FHE encryption algorithm $\texttt{Enc}$; that is, $\bar{x} = \texttt{Enc}(x)$ for $x \in \mathbb{Z}$. A continued fraction is encrypted means that each partial quotient of the continued fraction is encrypted. For a continued fraction $X = [x_0; x_1, x_2, \ldots]$,

$$\texttt{Enc}(X) := [\texttt{Enc}(x_0); \texttt{Enc}(x_1), \texttt{Enc}(x_2), \ldots],$$

which can be abbreviated as $\bar{X} := [\bar{x}_0; \bar{x}_1, \bar{x}_2, \ldots]$. To distinguish an integer and a real number, a small letter indicates an integer (e.g. X) and a capital letter indicates a real number (e.g. x_i). We write $\alpha \leftarrow \texttt{A}(a, b)$ to denote an algorithm which receives two inputs a and b and outputs α. log denotes a logarithm with base 2.

2.2 Continued Fraction

A set of integers is closed under addition and multiplication, but not division. However, there are some methods representing rational numbers to integers,

e.g., continued fractions and decimal representations. Continued fractions are more *mathematically natural* representations of rational numbers than decimal representations. First of all, the continued fraction representation for a rational number is finite, but decimal representation for a rational number may be infinite. Moreover, every rational number has an unique continued fraction representation with some restrictions. The successive approximations generated in finding the continued fraction representation of a number, i.e., by truncating the continued fraction representation, are in a certain sense (described below) the best possible. Therefore, it has occasionally been considered to be a great tool in mathematics and it has been researched for a long time in a variety of topics. Here, we rephrase only those related to our works.

A continued fraction can be obtained through an iterative process of representing a number as the sum of its integer part and the reciprocal of remaining part, and then writing the remaining part as the sum of its integer part and remaining part, and so on. In other words, given a real number X and $r_i < 1$ for all i, we have

$$X = x_0 + \frac{1}{r_0} = x_0 + \frac{1}{x_1 + \frac{1}{r_1}} = x_0 + \frac{1}{x_1 + \frac{1}{x_2 + \frac{1}{r_2}}} = \cdots \tag{2.2.1}$$

and use $X = [x_0; x_1, x_2, \ldots]$ to denote this. Note that x_0 can be any integer, but for $i \in \mathbb{N}, x_i$ must be positive and with this restriction, the continued fraction of X is unique. We can define further terminologies related to continued fractions.

Definition 2.2.1 ([48]). For a continued fraction $x = [x_0; x_1, \ldots]$,

- x_i is called *a partial quotient* of x for all i.
- A continued fraction x is *finite* if the number of partial quotients of some x is finite.

Keeping this definition in mind, the following theorem states that the correspondence between rational numbers and a finite continued fraction

$[a_0; a_1, \ldots, a_n]$ with an integer a_0 and positive integer a_i for $i > 0$ and $a_n > 1$ is one-to-one. We consider the situation that an integer is divided by another integer, so only rational numbers are considered in this paper. [1]

Theorem 2.2.2 ([48]). *Any rational number can be represented as a finite continued fraction and the continued fraction representation is unique when the last partial quotient is larger than 1.*

Theorem 2.2.3 ([48]). *Let* $\alpha = [a_0; a_1, a_2, \cdots]$ *and* $p_i/q_i = [a_0; a_1, \cdots, a_i]$. *For any rational number* $\frac{a}{b}$ *with* $a \in \mathbb{Z}$ *and* $b \in \mathbb{N}$, *and* $1 \leq b \leq q_i$,

$$\left|\frac{p_i}{q_i} - \alpha\right| \leq \left|\frac{a}{b} - \alpha\right|,$$

with equality if and only if $\frac{a}{b} = \frac{p_i}{q_i}$.

Theorem 2.2.3 indicates that p_i/q_i is the best possible approximation to α among all rational numbers with the same or smaller denominator. In other words, a continued fraction is the best approximation tool of rational numbers.

Size of Partial Quotients. There are some useful theorems for the partial quotients. Each partial quotients are closely related to each other and thus a partial quotient can be represented as the previous partial quotients. Moreover, when a rational number is represented as a continued fraction, its denominator and numerator can also be written as a multivariate polynomial in terms of the partial quotients, respectively. It means that the size of each partial quotient is *much smaller* than the denominator and the numerator. Before confirming this statement, we give a definition for the partial quotients and then using this definition, we give some theorems.

Definition 2.2.4 (Convergent). Let $X = [a_0; a_1, \ldots, a_{n-1}]$, where $X \in \mathbb{Q}$ and $a_i \in \mathbb{Z}$. The *convergents* of X are defined as

$$C_0 := [a_0], C_1 := [a_0; a_1], \ldots, C_i := [a_0; a_1, \ldots, a_i]$$

[1] Of course, since real numbers can be approximate to rational numbers, our techniques also can be applied to real numbers.

and C_i represents rational numbers, denoted by $C_i = p_i/q_i$ where p_i/q_i is in reduced form.

Since any real number can be approximated to the rational number and any rational number can be represented as a finite continued fraction by Theorem 2.2.2, we only consider the finite continued fraction. But, this notation can be also applied to an infinite continued fraction and trivially, $X = \lim_{i\to\infty} C_i$, where $X \in \mathbb{R}$. In this paper, we only consider a finite continued fraction, not an infinite continued fraction. Using the property of convergence, we claim that the size of partial quotients is much smaller than denominator and nominator in terms of bit size.

Theorem 2.2.5. *Let $X = [x_0; x_1, \ldots, x_{n-1}]$ be a continued fraction of p/q, where $X \in \mathbb{Q}$ and $a_i, p, q \in \mathbb{Z}$. Then,*

$$\log x_0 + \log x_1 + \ldots + \log x_{n-1} < logp$$
$$\log x_0 + \log x_1 + \ldots + \log x_{n-1} < logq$$

Proof. Let $X = [x_0; x_1, \ldots, x_{n-1}]$, where $X \in \mathbb{Q}$ and $x_i \in \mathbb{Z}$ and its convergents $C_i = [x_0; x_1, \ldots, x_i] = p_i/q_i$. By the mathematical induction, we can easily derive that the number p_i and q_i are given by the recurrence

$$p_i = x_i p_{i-1} + p_{i-2} \tag{2.2.2}$$
$$q_i = x_i q_{i-1} + q_{i-2} \tag{2.2.3}$$

with initial condition $p_0 = x_0, p_{-1} = 1, q_0 = 1, q_{-1} = 0$. Eq.(2.2.2) and

Eq.(2.2.3) can be written as a matrix representation as follows.

$$\begin{aligned}
\begin{pmatrix} p_i & p_{i-1} \\ q_i & q_{i-1} \end{pmatrix} &= \begin{pmatrix} p_{i-1} & p_{i-2} \\ q_{i-1} & q_{i-2} \end{pmatrix} \begin{pmatrix} a_i & 1 \\ 1 & 0 \end{pmatrix} \\
&= \begin{pmatrix} p_{i-2} & p_{i-3} \\ q_{i-2} & q_{i-3} \end{pmatrix} \begin{pmatrix} x_{i-1} & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_i & 1 \\ 1 & 0 \end{pmatrix} \\
&= \cdots \\
&= \begin{pmatrix} x_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} x_i & 1 \\ 1 & 0 \end{pmatrix}
\end{aligned}$$

Therefore, for all i, p_i and q_i contain $x_0x_1 \ldots x_i$ and there is no negative term, which implies that

$$\begin{aligned}
p_i &> x_0x_1 \ldots x_i \\
q_i &> x_0x_1 \ldots x_i
\end{aligned}$$

Taking a logarithm with base 2 for both inequalities and one can obtain we desired. □

Compared to Euclidean Algorithm. Euclidean algorithm is an very efficient algorithm which enables to compute greatest common divisors of two integers. The fundamental of euclidean algorithm is that the greatest common divisor of numbers does not change even if the larger number is subtracted from the other number. Since the subtraction reduce the large number, repeating this step outputs successively smaller pairs of numbers and finally two numbers become equal. The number is the greatest common divisor of the original two integers. Since a remainder for two integers can be obtained by successively subtracting the smaller number to the larger number, computing a remainder is reduced the iteration number in the euclidean algorithm instead of a subtraction. In this sense, both euclidean algorithm and continued fraction require a step to compute a quotient and a remainder for two

integers and actually, these algorithms are based on the same primitive. For two integer p and q, let r_i be a remainder and q_i a quotient for all i in the euclidean algorithm. Let r'_i be a remainder in continued fraction for all i. Note that a partial quotient is the same as q_i, so we also use q_i for the partial quotient.

Euclidean Algorithm	Continued Fraction	Relation
$p = q \cdot q_0 + r_0$	$\frac{p}{q} = q_0 + r'_0$	$r_0 = r'_0 \cdot q$
$q = r_0 \cdot q_1 + r_1$	$\frac{1}{r'_0} = q_1 + r'_1$	$r_1 = r'_1 \cdot r_0$
$r_0 = r_1 \cdot q_2 + r_2$	$\frac{1}{r'_1} = q_2 + r'_2$	$r_2 = r'_2 \cdot r_1$
$r_1 = r_2 \cdot q_3 + r_3$	$\frac{1}{r'_2} = q_3 + r'_3$	$r_3 = r'_3 \cdot r_2$
$\vdots$	$\vdots$	$\vdots$

where $0 \leq r_i < r_{i-1}$ and $0 \leq r'_i < 1$. Likewise, every partial quotient is the same as a quotient in the euclidean algorithm and every remainder for continued fraction can be expressed by a remainder of the euclidean algorithm, both algorithms are primitively the same.

2.3 Homomorphic Encryption

An FHE scheme, denoted by $\mathsf{FHE} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Ev})$, is a quadruple of probabilistic polynomial-time algorithms, as follows.

Key generation. The algorithm takes the security parameter λ and outputs a public encryption key pk, a public evaluation key ek and a secret decryption key sk. We write the algorithm as $(pk, ek, sk) \leftarrow \mathtt{KeyGen}(1^\lambda)$ and assume that the public key specifies the plaintext space P and the ciphertext space C.

Encryption. The algorithm $\bar{x} \leftarrow \mathtt{Enc}_{pk}(x)$ takes the public key pk and a message $x \in \mathsf{P}$ and outputs a ciphertext $\bar{x} \in \mathsf{C}$.

Decryption. The algorithm $x^* \leftarrow \mathtt{Dec}_{sk}(\bar{x})$ takes the secret key sk and a ciphertext c and outputs a message $x^* \in \mathsf{P}$.

Homomorphic evaluation. The algorithm takes the evaluation key ek, a function $f : (\{0,1\}^*)^n \rightarrow \{0,1\}^*$, and a set of n ciphertexts $\bar{\alpha}_1, \ldots, \bar{\alpha}_n$ and outputs a ciphertext $\bar{\alpha}_f$, denoted by $\bar{\alpha}_f \leftarrow \mathtt{Ev}_{ek}(f, \bar{\alpha}_1, \ldots, \bar{\alpha}_n)$.

In 2009, since Gentry's first secure FHE scheme from ideal lattices [42], various studies [29, 37, 77] have been presented on constructing efficient FHE schemes. However they have fairly poor performance. As a solution of efficient FHE, Brakerski and Vaikuntanathan [14] introduced the concept of leveled FHE schemes which allows the evaluation of functions of at most a pre-determined multiplicative depth, instead of arbitrary functions. Shortly after, Brakerski, Gentry, and Vaikuntanathan [12] proposed a leveled FHE scheme over polynomial rings, which has significantly improved performance over the previous schemes. Therefore, there are several good candidates for instantiating FHE; examples include Brakerski et al.'s scheme [12] and Bos et al.'s scheme [11].

An FHE scheme is said to be semantically secure if it achieves indistinguishability against chosen plaintext adversaries. We use a widely known formulation of semantic security [44], defined as follows.

Definition 2.3.1 (Semantic Security)**.** An FHE scheme is semantically secure if for any polynomial-time adversary $\mathcal{A}$, it holds that

$$|\Pr[\mathcal{A}(pk, \mathtt{Enc}(pk, m_0)) = 1] - \Pr[\mathcal{A}(pk, \mathtt{Enc}(pk, m_1)) = 1]|$$

is negligible in security parameter λ where $(pk, ek, sk) \leftarrow \mathtt{KeyGen}(1^\lambda)$ and $m_0, m_1 \in \mathsf{P}$ are chosen by the adversary $\mathcal{A}$.

2.4 Homomorphic Comparisons over the Integers

In the paper [25,26], the authors provide an equality circuit and a comparison circuit for fully homomorphic encryption using boolean circuits with data represented by bits.

2.4.1 Equality Circuit over the Integers

The equality circuit for two integers x and y outputs an encryption of 1 if two integers are the same, and otherwise, outputs an encryption of 0. To compare two integers, every bit of the same position should be the same and adding two bits and plus 1 enables to check the coincidence two bits because for $a, b \in \{0, 1\}$, $a + b + 1$ outputs 1 in $\mathbb{Z}_2$ if $a = b$, otherwise, outputs 0 in $\mathbb{Z}_2$.

x_i	y_i	$x_i = y_i$	$x_i + y_i + 1$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	1

Hence, using this equation, the authors in [25,26] can formalize the equality test $\mathtt{EQ}_{\mathbb{Z}}$ over the integers as follows; for $\bar{x} = \bar{x}_{\ell-1} \cdots \bar{x}_0$ and $\bar{y} = \bar{y}_{\ell-1} \cdots \bar{y}_0$ where $\bar{x}_i, \bar{y}_i \in \mathbb{Z}_2$,

$$\mathtt{EQ}_{\mathbb{Z}}(\bar{x}, \bar{y}) = \prod_{i=0}^{\ell-1}(1 + \bar{x}_i + \bar{y}_i) = \begin{cases} \bar{1} & \text{if } x = y \\ \bar{0} & \text{if } x \neq y \end{cases}$$

2.4.2 Greater-Than and Less-Than Circuits over the Integers

There are two types for the comparison circuits; one is Less-Than circuit and the other is Greater-Than circuit. Less-Than circuit compares two integers and outputs an encryption of 1 if x is less than y and outputs an encryption of 0 if x is greater than y and similarly, Greater-Than circuit compares two integers and outputs an encryption of 0 if x is greater than y and outputs an encryption of 1 if x is less than y. Actually, the principle of these two circuits is the same and then Greater-Than circuit can be easily obtained from Less-Than circuit by adding 1 to the output, and vice versa. If a integer $x = x_{n-1} \cdots x_0$ is less-than (or greater-than) other integer $y = y_{n-1} \cdots y_0$, there exists $0 \leq i < n$ such that $x_i < y_i$ (or $x_i > y_i$) and $x_j = y_j$ for all $i < j$. In similar as the equality circuit, $y_i(x_i + 1)$ (or $x_i(y_i + 1)$) outputs $1 \in \mathbb{Z}_2$ if $x_i = 0 < y_i = 1$ (or $y_i = 0 < x_i = 1$) and outputs 0.

x	y	$x > y$	$x(y+1)$	$x < y$	$(x+1)y$
0	0	0	0	0	0
0	1	0	0	1	1
1	0	1	1	0	0
1	1	0	0	0	0

Hence, using this relation, the authors can also formalize less-than circuit $\mathtt{LT}_{\mathbb{Z}}$ (or greater-than circuit $\mathtt{GT}_{\mathbb{Z}}$) over the integers combining the above equality circuit.

$$\mathtt{LT}_{\mathbb{Z}}(\bar{x}, \bar{y}) = \bar{y}_{\ell-1}(\bar{x}_{\ell-1} + 1) + \sum_{i=0}^{\ell-2} \left(\bar{y}_i(\bar{x}_i + 1) \cdot \prod_{j>i}^{\ell-1} (\bar{x}_i + \bar{y}_i + 1) \right) = \begin{cases} \bar{1} & \text{if } x < y \\ \bar{0} & \text{if } x \geq y \end{cases}$$

$$\mathtt{GT}_{\mathbb{Z}}(\bar{x}, \bar{y}) = \bar{x}_{\ell-1}(\bar{y}_{\ell-1} + 1) + \sum_{i=0}^{\ell-2} \left(\bar{x}_i(\bar{y}_i + 1) \cdot \prod_{j>i}^{\ell-1} (\bar{x}_i + \bar{y}_i + 1) \right) = \begin{cases} \bar{1} & \text{if } x > y \\ \bar{0} & \text{if } x \leq y \end{cases}$$

2.5 Fuzzy Extractor

Fuzzy extractors consist of two algorithms: one is generating a random number (`Gen`) and the other is reproducing a random number (`Rep`). `Gen` takes an input w such as biometric data and outputs a random number R and a helper value $P \in \{0,1\}^*$. `Rep` takes an input w' which may be different from w and P and it outputs the same random R whenever w' is close to w. For more detail, we give a formal definition of fuzzy extractors and its security.

Definition 2.5.1 (Fuzzy Extractors [35])**.** An $(\mathcal{M}, m, \ell, t, \epsilon)$-fuzzy extractor is a pair of randomized procedures `Gen` and `Rep` with the following properties:

1. The generation procedure `Gen` on input $w \in \mathcal{M}$ outputs an extracted string $R \in \{0,1\}^\ell$ and a helper string $P \in \{0,1\}^*$.

2. The reproduction procedure `Rep` takes an element $w' \in \mathcal{M}$ and a helper string $P \in \{0,1\}^*$ as inputs. The correctness property of fuzzy extractors guarantees that $\texttt{Rep}(w', P) = R$ whenever $\mathsf{HD}(w, w') \leq t$ and $(R, P) \leftarrow \texttt{Gen}(w)$.

3. The security property guarantees that for any distribution W on $\mathcal{M}$, the string R is nearly uniform even for those who observe P: if $(R, P) \leftarrow \texttt{Gen}(W)$, then $\mathsf{SD}((R, P), (U_\ell, P)) \leq \epsilon$ where U_ℓ is a random ℓ-bit string.

A fuzzy extractor is *efficient* if `Gen` and `Rep` run in expected polynomial time.

Digital Lockers. A concept of a digital locker is first proposed by Canetti and Dakdouk [19]. Conceptually, digital locker is almost same as symmetric encryption, however, the digital locker can maintain a security even when a key with low-entropy is used multiple times. Like other cryptosystems, obtaining any information of plaintext from the ciphertext is as hard as guessing the secret key.

A digital locker is a pair of randomized procedures `lock` and `unlock`: `lock` plays an encryption role and `unlock` a decryption role. `lock` performs locking a message m using a secret key k and outputs a ciphertext c and

`unlock` executes the unlocking using the secret key. Digital lockers have an additional feature in `unlock`. If `unlock` receives a correct secret key, it outputs a message, however, when it receives a wrong secret key, it outputs empty set $\perp$ with high probability. In this reason, anyone can recognize that the current key is the wrong. We use notation $c = \texttt{lock}(k, m)$ and $m = \texttt{unlock}(k, c)$ when k is the correct key.

The security of digital lockers is

Definition 2.5.2 (Composable Secure Digital Locker [20])**.** The pair of algorithm $(\texttt{lock}, \texttt{unlock})$ with security parameter λ is an ℓ-composable sucure digital locker with error γ if the following hold:

Correctness. For all k and m, $\Pr[\texttt{unlock}(k, \texttt{lock}(k, m)) = m] \geq 1-\gamma$. Moreover, for any $k' \neq k$, $\Pr[\texttt{unlock}(k', \texttt{lock}(k, m)) = \perp] \geq 1 - \gamma$.

Security. For every PPT adversary A and every positive polynomial p, there exists a simulator S and a polynomial $q(\lambda)$ such that for any sufficiently large s, any polynomially long sequence of values (m_i, k_i) for $i = 1, \ldots, \ell$ and any auxiliary input $z \in \{0,1\}^*$,

2.5.1 Reusable Fuzzy Extractor

In the recent Eurocrypt, Canetti *et al.* construct reusable fuzzy extractors that is a fuzzy extractor with reusability property. It supports a couple of times enrollment phases with the same input value, so one can register many unrelated service providers. Specifically, the algorithm `Gen` can execute multiple times on correlated $w_1, \ldots, w_\rho$ of a given input. For each input, `Gen` produce independent pairs of values $(r_1, p_1), \ldots, (r_\rho, p_\rho)$. The security of reusable fuzzy extractors is that an adversary

Main idea of Canetti *et al.* scheme [20] is that a random r is locked multiple times by some random subsets $v_1, \ldots, v_\ell$ of an input and thus each locked value can be unlocked only with $v_1, \ldots, v_\ell$, respectively. When one wants to regenerate the same r with one's input, one should make subsets

$v'_1, \ldots, v'_\ell$ by the same process constructing v_i's and then try to unlock with these subsets. If one could succeed in making the same subset as v_i, one can obtain the wanted r. Since the composable digital lockers enables to sample multiple times, they can be used to construct reusable fuzzy extractors.

Concrete Construction. Let $\mathcal{M} = \{0,1\}^n$ be a input space and $W = w_1 \ldots w_n \in \mathcal{M}$, where $w_i \in \{0,1\}$. Let ℓ be Let $\texttt{lock}, \texttt{unlock}$ be an ℓ-composable secure digital locker with error γ.

`Gen`
Input: $W = w_1 \ldots w_n$
1. Sample $r \xleftarrow{\$} \{0,1\}^\kappa$
2. **for** $i = 1, \ldots, \ell$
(i) choose uniformly $j_{i,m} \xleftarrow{\$} \{1, \ldots, n\}$ for each $1 \le m \le k$
(ii) $v_i \leftarrow w_{j_{i,1}} \ldots w_{j_{i,k}}$
(iii) $c_i \leftarrow \texttt{lock}(v_i, r)$
(iv) $p_i \leftarrow c_i, (j_{i,1}, \ldots, j_{i,k})$
3. **return** (r, p) where $p = p_1 \ldots p_\ell$

Algorithm 2.1: `Gen` of Canetti's Reusable Fuzzy Extractors

Each subset is made by randomly chosen k positions of the input and thus position information is required when recovring r. In this sense, p indicates that each subset is come from and it would be a helper value.

`Rep`
Input: $W' = w'_1 \dots w'_n, p = p_1 \dots p_\ell$
1. **for** $i = 1$ **to** ℓ
(i) Parse p_i as $c_i, (j_{i,1}, \dots, j_{i,k})$
(ii) $v'_i \leftarrow w'_{j_{i,1}} \dots w'_{j_{i,k}}$
(iii) $r_i \leftarrow \texttt{unlock}(v'_i, c)$
(iv) **if** $r_i \neq \perp$
(v) **return** r_i.

Algorithm 2.2: `Rep` of Canetti's Reusable Fuzzy Extractors

Parameters Setting. To consider the FRR(false reject ratio), it needs to satisfy the following inequality:

$$\left(1 - \left(1 - \frac{t}{n}\right)^k\right)^\ell + \ell \cdot \gamma \leq \delta$$

Using the approximation $e^x \approx 1 + x$, they suggested a parameter setting such that $\ell \cdot \gamma \leq \delta/2$, $tk = cn \log n$ and $\ell \approx n^c \log \frac{2}{\delta}$ for some constant c.

Chapter 3

Algorithms for Homomorphic Integer Division

3.1 Overview and Related Works

Homomorphic encryption enables to handle ciphertexts. It can add, subtract and multiply two ciphertext without decryption, however, it does not support dividing two ciphertexts. In order to solve this problem, there are technical challenges.

Until now, the most existing homomorphic encryption scheme only supports an integer plaintext, but a set of integers is not closed under the division, so the output cannot be put in the message of homomorphic encryption. Thus, in order to divide two ciphertexts, someone should construct a new homomorphic encryption scheme which supports an arithmetic of real numbers or otherwise, a real number (or, a rational number) should be represented to an integer (or a set of integers). In this sense, we employ continued fraction which has interesting properties. For example, every real number has a *essentially unique* continued fraction representation and most of all, simple rational numbers have representations with *few* terms and *small* integers. Hence, continued fraction could be a good candidate for representing real numbers.

A division algorithm should be replaced by alternative algorithm which only uses a couple of additions and multiplications. And we suggest an alternative algorithm as restoring division algorithm which outputs a quotient and a remainder from a divisor and a dividend. Generally, it can be replaced by subtracting divisor repeatedly from the dividend, however, this algorithm is terminated when the output is less than the dividend. Even though one can utilize the comparison circuit [25, 26] on the ciphertexts domain, one cannot check whether the output satisfies the terminated condition since the output is also encrypted. On the other hand, restoring division algorithm is also an iterative algorithm, but the number of iterations depends on the input size and it can be easily known. When a plaintext space is $\mathbb{Z}_2$, then it is the same as the number of ciphertexts. Furthermore, this algorithm also has a step to check whether negative or positive, but it can be easily known from the most significant bit.

Continued fraction and euclidean algorithm are based on the same principle. Thus, a partial quotient can be obtained through euclidean algorithm and each iteration of euclidean algorithm also outputs a quotient and a remainder, which can be executed by restoring division algorithm. The remaining problem is the number of partial quotients. But, each iteration of euclidean algorithm outputs a partial quotient and thus the number of partial quotients is the same as the number of iterations of euclidean algorithm which depends on the input size and it is easily known. In addition, we can also evaluate a polynomial whose coefficients are in the rational numbers since our technique can represent an encrypted rational number.

Related Works. As related works, I first investigate Graepel et al.'s result [46] and Bos et al.'s work [10]. In both works, the authors first fix a desired precision, multiply through by a fixed denominator, and round to the nearest integer because any real number can be approximated by rational numbers to arbitrary numbers and subsequently encoded to ring elements. However, this approach to represent real numbers has some drawbacks. When two encoded rational numbers are multiplied, it should be performed with-

out any modular reduction and thus the plaintext space of FHE must be sufficiently large. And these encoding techniques always keep track of the precision in order to reconstruct a real number, but we do not need it because continued fraction is regardless of the precision.

The most closely related work to the present one is Jäschke et al.'s scheme and Costache et al.'s scheme [31]. In [52], Jäschke and Armknecht dealt with a particular encoding for rational numbers in the FHE context. However, due to their specialty of encoding, Boolean comparison on encrypted data of n-bit length should be inefficiently implemented since $O(n)$ multiplication depth is needed rather than $O(\log n)$. More recently, the work of [31] investigated fixed-point arithmetic in ring-based somewhat homomorphic encryption (SHE). However, this approach does not allow to compute a reciprocal of encrypted data and support integer division.

3.2 Restoring Division Algorithm

As you can see in Eq (2.2.1), the main obstacle representing continued fraction when the inputs are encrypted is computing partial quotients x_i for all i To formalize this problem, suppose there are two encrypted integers using homomorphic encryption, called $\bar{N}$ and $\bar{D}$ and compute $\bar{N}/\bar{D}$ in a continued fraction form. Since computing a partial quotient is equivalent to computing a quotient, our goal is computing a quotient when given two encrypted integers. Typically, the simplest way to computing a quotient is by repeated subtraction as shown in Algorithm 3.1.

Division Algorithm
Input: numerator n and denominator d
Output: quotient q and remainder r
1. $r \leftarrow n$ and $q \leftarrow 0$
2. **while** $r \geq d$
(i) $r \leftarrow r - d$
(ii) $q \leftarrow q + 1$
3. **return** q and r

Algorithm 3.1: Division Algorithm

Algorithm 3.1 could be executed even if n and d are encrypted because every step except line 2 consists of an addition, however, the problem is the termination condition of this algorithm requires **while**$(r \geq d)$. Even though we can compare two ciphertexts through a comparison circuit [25], the output of this circuit is still encrypted and thus we cannot know the output of the termination condition. For this reason, we choose the restoring division algorithm and in this algorithm, comparing two integers is replaced by determining a sign of some integer. [1] Restoring division algorithm is based on a standard recurrence equation.

$$p_{i+1} = b \times p_i - q_{n-(j+1)} \times d$$

where p_i is i-th the partial remainder of the division, b is the base (radix), n is the number of digits in the quotient, q_i is the i-th digits of the quotient and d is the divisor. In this paper, we restrict the message space of FHE is $\mathbb{Z}_2$ because it occurs the most efficient performance and it implies the radix $b = 2$. At first, we give the restoring division algorithm on plaintext domain and then we translate it to the ciphertext domain. Restoring division algorithm on plaintext domain can be seen in Algorithm 3.2.

[1] There are also non-restoring division algorithm, however, it requires more computational complexity than the restoring division algorithm.

Restoring Division on Plaintext

Input: numerator n and divisor d

Output: quotient q and remainder r

1. $r \leftarrow n$ and $k \leftarrow \log n$
2. $d \leftarrow d << k$
3. **for** $i = k - 1$ **to** 0
   - (i) $r \leftarrow 2r - d$
   - (ii) **if** $r < 0$
   - (iii) **then** $q_i = 0$
   - (iv) $r \leftarrow r + d$
   - (v) **else** $q_i = 1$
4. **return** q and r

Algorithm 3.2: Restoring Division Algorithm on Plaintext Domain

Since **if-else** statements can be replaced by using a couple of multiplications and Algorithm 3.2 always terminate after n iterations, we can handle it even if every input is encrypted. For the simplicity, let $r = r_{\ell-1} \dots r_0$ and $q = q_{\ell-1} \dots q_0$ be the binary representation. Note that ℓ is the same as the number of ciphertexts for n when we use a plaintext space $\mathbb{Z}_2$ and a quotient and a remainder should not exceed the numerator n, so allocating ℓ bits is enough for q and r. Since we set a plaintext space $\mathbb{Z}_2$, two additional circuits `FullAdder` and `BinarySubtract` are required in order to calculate bit addition and bit subtraction, respectively. We do not give a detail about `FullAdder` and `BinarySubtract` in this section and see the appendix for more details.

To determine the sign of r, we only look into the most significant bit (MSB) of r. If MSB of r is 1, then, R should be negative and otherwise, R is positive. Thus, we can now handle **if-else** statements totally even if inputs are encrypted and restoring division algorithm on ciphertext domain can be seen in Algorithm 3.3.

Restoring Division on Ciphertext
Input: encrypted numerator $\bar{n}$ and encrypted denominator $\bar{d}$
Output: encrypted quotient $\bar{q}$ and encrypted remainder $\bar{r}$
1. $\bar{r} \leftarrow \bar{n}$ and $k \leftarrow$ number of ciphertexts for $\bar{n}$
2. **for** $i = k - 1$ **to** 0
(i) $\bar{r} \leftarrow 2\bar{r} - \bar{d}$
(ii) $\bar{q}_i \leftarrow 1 - \bar{r}_{n-1}$
(iii) $\bar{r} \leftarrow \bar{r}_{n-1} \cdot (\bar{r} + \bar{d}) + (1 - \bar{r}_{n-1}) \cdot \bar{r}$
3. **return** $\bar{q}$ and $\bar{r}$

Table 3.3: Restoring Division Algorithm on Ciphertext Domain

3.3 Homomorphic Integer Division

There are a couple of methods to represent the rational numbers such as decimal representation, polynomial embedding [10] or continued fraction. Each encoding techniques have their own strong aspect compared to other encoding techniques. For example, when the natural decimal representation is used, anyone can easily compare two rational numbers and when the technique for polynomial embedding is used, the basic arithmetics except the division can be executed easily because these are exactly the same as the arithmetics on the polynomials. However, these encoding techniques require large plaintext space to present the real numbers and the division is not supported with these encoding techniques.

To encrypt the rational numbers through homomorphic encryption, the rational number should be represented into (a set of) integers because homomorphic encryption only supports an integer message space. But, since a message space is directly related to the performance matter, the choice of message space is the most crucial part in terms of efficiency. In this sense, continued fraction encoding is the best choice when applying homomorphic encryption because it can be represented to a set of *relatively small* integers.

3.3.1 Algorithm

Now, we can present a concrete algorithm for dividing two encrypted integers using Algorithm 3.3. Suppose there are two encrypted integers $\bar{N}$ and $\bar{D}$. Since a partial quotient is the same as a quotient, we can get one partial quotient after executing Algorithm 3.3. It means we execute the restoring division algorithm several times, we can get all partial quotients of $\bar{N}/\bar{D}$.

The only remaining problem is how many repeating restoring division algorithms are required to represent all partial quotients. Fortunately, we can solve this problem by combining euclidean algorithm. Euclidean algorithm is an very efficient algorithm which enables to compute greatest common divisors of two integers. Since every quotient is actually one of the partial quotients in continued fractions, euclidean algorithm is very close to continued fractions and thus the number of partial quotients is the same as the number of repeating euclidean algorithm and it totally depends on the input bit size. Euclidean algorithm can compute greatest common divisor of numerator and denominator with number of iterations $2 \log d$, where d is the bit size of denominator. Therefore, we can also obtain all partial quotients after executing restoring division algorithm $2 \log d$ times. A concrete construction for dividing two encrypted integers can be seen in Algorithm 3.4.

Homomorphic Integer Division

Input: Encrypted Numerator $\bar{N}$ and Encrypted Denominator $\bar{D}$

Output: Encrypted Continued Fraction $[\bar{q}_0; \bar{q}_1, \ldots, \bar{q}_{2d-1}] = \bar{N}/\bar{D}$

1. $d \leftarrow$ number of ciphertexts for $\bar{D}$
2. **for** $i = 0$ **to** $2d - 1$
   (i) $(\bar{q}_i, \bar{r}_i) \leftarrow$ `RestoringDivision`$(\bar{r}_{i-2}, \bar{r}_{i-1})$
3. **return** $[\bar{q}_0; \bar{q}_1, \ldots, \bar{q}_{2d-1}]$

Algorithm 3.4: Division Algorithm

3.3.2 Efficiency

We describe how to divide two encrypted integers and represent through continued fractions in Section 3.3.1. There are many types of necessary operations in order to execute our suggestion such as homomorphic additions HA and homomorphic multiplications HM. In this section, we analyze the efficiency of the proposed algorithms. Since `FullAdder` and `BinarySubtract` also can be represented by using HA and HM, we can evaluate every step of our proposal in terms of the number of HA and HM. In Algorithm 3.4, every step but line 4 is just allocated to the variables and so there is no cost. Therefore, the computational complexity of Algorithm 3.4 is the same as multiple times of the computational complexity of Algorithm 3.3.

Restoring Division Algorithm. At first, we investigate a computational complexity of Algorithm 3.3 in terms of HA and HM. Since `BinarySubtract` is required to execute Algorithm 3.3, we need to set the plaintext space $\mathbb{Z}_2$. It means every bit of a message is encrypted respectively and ciphertexts are generated as many as the bit-size of the message. For this reason, line 3 does not require automomorphism to shift bit left and just reallocates the memory. Line 4 requires `BinarySubtract` which will be covered in the below paragraph and line 5 needs a homomorphic subtraction, not `BinarySubtract`. In line 5, it seems `FullAdder` is necessary, however, actually this `FullAdder` is just restoring R before executing `BinarySubtract`. For this reason, $r_{n-1} \cdot \texttt{FullAdder}(\bar{R}, \bar{D})$ requires just one homomorphic multiplication and line 6 requires only two homomorphic multiplications and one homomorphic subtraction. In total, in order to execute the restoring division algorithm, it costs just two HA, two HM and one `BinarySubtract`. Note that the homomorphic subtraction is the same as the homomorphic addition.

Bit Subtraction and Full Adder. Now, we give a computational complexity of `BinarySubtract` and `FullAdder`. Even though `FullAdder` is not necessary in line 5 of Algorithm 3.3, `FullAdder` is still necessary because of `BinarySubtract`. Since we need to evaluate a complexity of $\texttt{BinarySubtract}(\bar{R}, \bar{D})$, we may assume the input of `BinarySubtract` is n bits. The first step of

`BinarySubtract` is making 2's complement $\bar{R}$ and it requires n HA and one `FullAdder`. And the next step is adding $\bar{D}$ through `FullAdder`, so it also cost one `FullAdder`. Note that `FullAdder` for two n bits costs $2n-1$ HA and $n-1$ HM due to carry. Therefore, in order to execute `BinarySubtract` for two inputs of size n bits, it costs $5n-2$ HA and $2(n-1)$ HM.

Homomorphic Division Algorithm. By the previous paragraph, we can know that restoring division algorithm for two integers of size n bits requires $(3n+2)$ HA and $2n$ HM. According to Algorithm 3.4, the number of $2n$ iterations are required and each iteration consists of the restoring division algorithm. Hence, we can conclude that Algorithm 3.4 requires $2n(3n+2)$ HA and $4n^2$ HM, where n is the input size.

	# HA	# HM
`FullAdder`	$2n-1$	$n-1$
`BinarySubtract`	$5n-2$	$2n-2$
Restoring Division Algorithm	$5n$	$2n$
Homomorphic Division Algorithm	$10n^2$	$4n^2$

Table 3.5: Necessary number of arithmetics for each algorithm

3.4 Homomorphic Arithmetics over the Polynomials

The cloud computing enables to evaluate polynomial and provide useful service to a user. For example, a Cox model is a statistical technique for exploring the relationship between the survival of a patient and several explanatory variables, and it provides an estimate of the effect of treatment on survival after adjusting for other explanatory variables. In addition, this technique allows us to estimate the hazard (or risk) of death for an individual given their prognostic variables. To protect the sensitive medical data, the data should

be encrypted, however, these data are not form of integers but homomorphic encryption supports only integer message space.

In the previous section, we have described an algorithm for homomorphic integer division in Algorithm 3.4. Actually, this algorithm is the same as transforming encrypted rational numbers to encrypted continued fraction and then we can also handle encrypted rational numbers as well as encrypted integers. In this section, we present the method for evaluating polynomials homomorphically being encrypted. Since the evaluation of any polynomial over $\mathbb{Q}$ is also in $\mathbb{Q}$, the polynomial over the rational numbers can be computed using the above technique. Moreover, any arithmetic of two polynomials over $\mathbb{Q}$ is also a polynomial over $\mathbb{Q}$ and thus we can also evaluate it being encrypted.

3.4.1 Description

Suppose there are two polynomials whose coefficients are in the rational numbers, say $a(x) = \sum_{i=0}^{n-1} a_i x^i \in \mathbb{Q}[x]$ and $b(x) = \sum_{i=0}^{n-1} b_i x^i \in \mathbb{Q}[x]$. Then, $a(x) \pm b(x) = \sum_{i=0}^{n-1} (a_i \pm b_i) x^i$ and since the rational numbers are closed under the addition and subtractions, $a(x) \pm b(x) \in \mathbb{Q}[x]$. No matter whether $x = p$ is in the integers or the rational numbers, $a(p) \pm b(p)$ are in the rational numbers with high probability. Similarly, $a(x) \cdot b(x)$ is also a polynomial of degree $2n - 2$ whose coefficients are in the rational numbers. It implies that we can evaluate an addition, a subtraction and a multiplication of two polynomials over $\mathbb{Q}$ being encrypted and using Algorithm 3.4, we can express the encrypted rational numbers to the encrypted continued fraction.

Consider dividing two polynomials in $\mathbb{Q}[x]$. Even though the rational numbers are closed under the divison, $a(x)/b(x)$ is not a polynomial over $\mathbb{Q}$. However, when we evaluate $a(x)/b(x)$ at $x = p \in \mathbb{Q}$, $a(p)/b(p)$ becomes the rational numbers and then we can also utilize Algorithm 3.4. Therefore, we can conclude that we can evaluate every arithmetics for polynomials over $\mathbb{Q}$ using the above techniques.

Chapter 4

Algorithms for Homomorphic Comparisons over the Real Numbers

4.1 Overview and Related Works

Cloud computing enables general users to outsource their complex data manipulation systems to the commercial public cloud while enjoying high availability and economic savings. For ensuring data privacy, sensitive data should be encrypted before outsourcing to a remote server. Thus, enabling the cloud server to manipulate all types of data without decryption is of paramount importance. However, the great majority of existing solutions for secure data manipulation focus on computations on the integers; thus there are unavoidably restrictions to integer arithmetic or comparison.

In this work, we provide an efficient solution for private comparison between non-integer data types. Different from existing secure comparison protocols with real numbers that have been studied in the realm of multiparty computation, our work allows the cloud server to compare encrypted real numbers without any interaction with users. For the purpose of non-interactiveness, we base our comparison protocol on fully homomorphic en-

cryption (FHE); moreover, by encoding real numbers with continued fraction, we relieve the computational overhead caused by FHE. Such a non-interactive approach as ours fits well into the cloud environment. Furthermore, we present interesting applications built on top of our comparison solution: private sorting and private database query. Our micro-benchmark results demonstrate that our solution is a viable approach for cloud computing.

In this chapter, we make the following specific contributions. At first, we design efficient homomorphic comparison algorithms, called equality, less-than, greater-than, for the real numbers in a homomorphic encryption framework. Our suggested algorithms are based on the comparison circuits over the integers since a real number is represented by a set of integers. In this thesis, we only consider a plaintext space is $\mathbb{Z}_2$, however, it can be changed depending on the situations and our circuits support an arbitrary plaintext space. If two continued fractions are the same, then every partial quotients of the same position should be the same. If two continued fractions are different, then a partial quotient of some position should be different and since this partial quotient dominates all right-side partial quotients, the comparison result depends on that partial quotient. With these properties, we can translate it to the ciphertexts domain and present concrete algorithms on the ciphertext domain. After then, we evaluate the developed techniques to demonstrate our techniques are feasible and are capable of efficient homomorphic comparisons over FHE-encrypted real numbers. Even though we have to execute comparison circuits for all partial quotients, each partial quotient is independent and thus it can be performed at the same time. On the other hands, in order to obtain a result, we have to some more multiplications between the result for each partial quotients, but due to small plaintext space, these operations are also on the small ciphertext domain compared to the other encoding techniques.

At last, We list specific applications on top of our FHE-based algorithms whose expected services rely on computations over the real numbers in a

privacy-preserving manner. Especially, we have focused on the database service such as sorting and private database queries. The most sorting algorithms consist of the comparison and swap algorithm. Moreover, swap algorithm can be easily constructed using the comparison algorithm since the position of two data is changed (swap) when the order is wrong (comparison). Hence, our suggestions could be applied to the almost generic sorting algorithm. Similarly, private database queries are mainly based on the quality circuit and the comparison circuits. For example, in the case of retrieval queries, equality circuit is the main building block and in the case of aggregate queries, equality circuit and comparison circuits are the main building block. Thus, with our techniques, one can execute such private database queries being encrypted.

Related Works. As mentioned above, secure comparison is a key primitive and thus there have been lots of solutions for the secure comparison problem. However most solutions have focused on integer comparisons regardless of which areas their underlying tools come from (e.g., MPC or FHE). We recommend reference [32] as a good survey for these existing techniques.

Catrina and Saxena represent rational numbers as fixed-point representation [21]. More specifically, they write a rational number q by $q = n_0.n_1 \cdots n_\ell$ with precision ℓ, compute $q' = \sum_{i=0}^{\ell} n_i \beta^i \in \mathbb{Z}$ for a prefixed radix β, and take q' as a fixed-point encoding of q by observing that $q = q' \cdot \beta^{-\ell}$ as well as by publicly opening ℓ. Then the authors provide an interactive comparison protocol that tests if a secret integer q' is equal or less than zero [21, §3.2]. This protocol can be easily extended to greater-than comparison between two secret rational numbers. However this protocol requires $O(\ell)$ rounds. Furthermore, the length of the fractional part, ℓ, should be revealed.

Franz et al. in [40] use a logarithmic representation proposed in [55]. For some $b \in \mathbb{N}$, $[-b, b]$ denotes a closed interval $\{r \in \mathbb{R} | -b \leq r \leq b\}$. Given a non-zero real number $r \in [-b, b]$ along with a scaling factor ℓ, a base for logarithm β, and a constant $c \geq b$, compute $\tau = \left\lceil -\ell \cdot \log_\beta \left(\frac{|r|}{c} \right) \right\rfloor$ and take (s, z, τ) as an encoding of r where $s \in \{0, 1\}$ is the sign of r and $z \in \{0, 1\}$

is a flag indicating whether $r = 0$ or not. One can recover r by setting $r = s \cdot c \cdot \beta^{-\tau/\ell}$. The authors note a protocol to compare two real numbers encrypted by a special-purpose additive homomorphic encryption by Damgård et al. [33]. However, because Damgård et al.'s encryption scheme requires bit decomposition, their protocol consisting of 3 rounds requires to encrypt encodings of real numbers in a bit-by-bit manner and their underlying encryption scheme strictly limits a general use of encrypted data.

Recently, Liu et al. [58] propose a floating point arithmetic protocol while representing real number as the IEEE 754 standard format. Given a real number r, the authors write it by $r = (s, z, e, \ell)$ where s, z are the same as above, e is the exponent, and ℓ is the mantissa defined in the IEEE 754 standard [50]. Their basic observations are tow-folds. The first is that every component in the representation consists of integers, and the second is that repeatedly applying bit manipulations bit shift and alignment to component-wise representations of floating point numbers preserves their arithmetics. As a result, if additive shares of each component are distributed and bit manipulation protocols over the shares are repeatedly invoked, then additive shares of floating point arithmetics are obtained. However, no floating-point number comparison protocol is mentioned. Using the same representation and the same cryptographic tools, Aliasgari et al. provide a series of protocols for complex floating point arithmetics such as square root, exponentiation, and logarithm. Thus their protocols do not give a floating point comparison solution.

More recently, Pullonen and Siim [66] and Dimitrov et al. [34] also study secure arithmetics over real numbers. Pullonen and Siim improve Aliasgari et al.'s protocols by fusing garbled circuit with secret sharing schemes. However, no protocol for comparison is provided. Different from other existing work, Dimitrov et al. represent real numbers as their golden representation. Precisely, given a real number r they write it by $r = a - \varphi \cdot b$ where φ is the golden ratio. Then they construct protocols for computing square root, logarithm, and exponentiation over pairs of secretly shared integers (a, b) representing

real numbers. Likewise, no explicit protocol for real number comparison is provided.

A common characteristic of existing solutions in the MPC framework is that higher-layer protocols built on top of them have logarithmic round complexity; for example, secure sorting for N secret shared N real numbers requires $O(\log N)$ rounds.

4.2 Comparing Two Continued Fractions

For better understanding, we first design algorithms (precisely, boolean circuits) for comparing two continued fractions in the clear, and then extend them such that they do the same on FHE-encrypted CFs. We construct the algorithms by plugging in the integer comparison algorithms discussed in Section 2.4.

4.2.1 Our Idea: Comparing Two CFs in the Clear

Given two real numbers represented by the decimal expansion, it is straightforward to compare the two numbers. Concretely, it can be done by comparing two integers of the same position with a left-to-right direction. When considering the use of FHE to encrypt numbers in decimal form, one needs to instantiate an FHE scheme with a relatively large plaintext space (e.g., $\mathbb{Z}_{2^{20}}$), for keeping a reasonable precision as mentioned before. This leads to performance degradation, which, in turn, may lead to a lower performance throughput of upper-layer applications, compared to the same circuits on encryptions under an FHE scheme of a smaller plaintext space.

As observed in Section 2.2, we introduce continued fraction to fix this performance problem. A way to compare real numbers in CF are quite similar to that for numbers in decimal form, but due to the smaller plaintext space, it is possible to achieve a more efficient comparison circuit between FHE-encrypted real numbers.

Containing an equal number of partial quotients. Denoting by n the number of partial quotients, let $X = [x_0; x_1, \ldots, x_{n-1}]$ and $Y = [y_0; y_1, \ldots, y_{n-1}]$. For demonstrating our idea, we look into a few cases where their partial quotients at a specific position are different from each other.

Case 1. If $x_0 \geq y_0$, then $X \geq Y$.

By the definition of CF (see Eq. (2.2.1)), we see that the partial quotients x_0 and y_0 dominate all other partial quotients x_i and y_i for $i > 0$. Therefore, X is greater (resp., less) than Y if x_0 is greater (resp., less) than y_0.

Case 2. If $x_0 = y_0$ and $x_1 \geq y_1$, then $X \leq Y$.

Consider the case where $x_0 = y_0$. Then, Eq. (2.2.1) can be written as follows.

$$\frac{1}{X - x_0} = x_1 + \frac{1}{x_2 + \frac{1}{x_3 + \ldots}} \quad \text{and} \quad \frac{1}{Y - y_0} = y_1 + \frac{1}{y_2 + \frac{1}{y_3 + \ldots}},$$

which can be re-written as $\frac{1}{X-x_0} = [x_1; x_2, \ldots, x_{n-1}]$ and $\frac{1}{Y-y_0} = [y_1; y_2, \ldots, y_{n-1}]$. Therefore, the result of the comparison between $\frac{1}{X-x_0}$ and $\frac{1}{Y-y_0}$ completely depends on x_1 and y_1. The only difference from Case 1 is that if x_1 is greater (resp., less) than y_1, then X is less (resp., greater) than Y because $\frac{1}{X-x_0} > \frac{1}{Y-y_0}$ (resp., $\frac{1}{X-x_0} < \frac{1}{Y-y_0}$) where $x_0 = y_0$.

Case 3. If $x_0 = y_0$, $x_1 = y_1$ and $x_2 \geq y_2$, then $X \geq Y$.

Consider the case where $x_0 = y_0$ and $x_1 = y_1$. Then, $\frac{1}{X-x_0} - x_1 = [0; x_2, \ldots, x_{n-1}]$ and $\frac{1}{Y-y_0} - y_1 = [0; y_2, \ldots, y_{n-1}]$. Since the first two partial quotients are the same, x_2 and y_2 determine the comparison between $\frac{1}{X-x_0} - x_1$ and $\frac{1}{Y-y_0} - y_1$. If x_2 is greater (resp., less) than y_2, $\frac{1}{X-x_0} - x_1$ is less (resp., greater) than $\frac{1}{Y-y_0} - y_1$, which implies that X is greater (resp., less) than Y.

Containing an unequal number of partial quotients. Let $X = [x_0; x_1, \ldots, x_{n-1}]$ and $Y = [y_0; y_1, \ldots, y_{m-1}]$ for distinct positive integers n and m. Without loss of generality, we assume $n \leq m$. For $0 \leq i \leq n-1$, it is easy to compare two CFs by applying the same rule as above. However, it is not clear to compare

two CFs because it may happen that $x_i = y_i$ for all $i \in \{0, 1, \ldots, n-1\}$, but these is no partial quotient of X corresponding to y_i for some $i \geq n$. To fix this problem, we get two real numbers in CF to have the same number of partial quotients, using the following simple trick.

A key observation behind our idea is that since $0 = \frac{1}{\infty}$, adding ∞ to the end of partial quotients at the X side causes no side effects. Namely, since

$$\frac{1}{x_{n-1}} = \frac{1}{x_{n-1} + \frac{1}{\infty}},$$

we have that $X = [x_0; x_1, \ldots, x_{n-1}] = [x_0; x_1, \ldots, x_{n-1}, \infty]$

$= [x_0; x_1, \ldots, x_{n-1}, \overbrace{\infty, \infty, \ldots, \infty}^{m-n}]$. In conclusion, we can always write two real numbers as two continued fractions with the same number of partial quotients. Thus, the idea as sketched above still makes sense in comparing two real numbers in CF that have a different number of partial quotients.

Our argument can be formally stated by the following theorem. The proof of the theorem is clear by mathematical induction on k.

Theorem 4.2.1. *Let $X = [x_0; x_1, \ldots, x_{n-1}]$ and $Y = [y_0; y_1, \ldots, y_{m-1}]$ and $n \leq m$. Let k be the smallest index for which $x_k \neq y_k$. Then,*

$$X < Y \text{ if } (-1)^k (x_k - y_k) < 0$$

and $X > Y$ otherwise. If there is no such k which implies $x_i = y_i$ for all $i < n$ and $n < m$, then $X < Y$ if n is odd and $X > Y$ if n is even. If there is no such k and $n = m$, then clearly $X = Y$.

In addition, Alg. 4.1 provides a concrete algorithm to compare two continued fractions in the clear.

4.3 Equality Circuit

In this section, we translate the equality condition to the ciphertext domain. Note that if two continued fraction are the same, then every partial quotient for the same position should be the same.

4.3.1 Construction

Equality circuit, denoted by $\mathtt{EQ}_{\mathbb{R}}$, which takes as input two encrypted CFs can be defined as follows.

$$\mathtt{EQ}_{\mathbb{R}}(\bar{X}, \bar{Y}) := \prod_{i=0}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_i, \bar{y}_i).$$

Let $\bar{X} = [\bar{x}_0; \bar{x}_1, \ldots, \bar{x}_{n-1}]$ and $\bar{Y} = [\bar{y}_0; \bar{y}_1, \ldots, \bar{y}_{m-1}]$ be two encrypted CFs. Only if $n = m$ and $x_i = y_i$ for all $0 \leq i \leq n-1$, equality of two CFs holds true. The first condition is easily checked by the number of their ciphertexts, and the second condition is examined by comparing two encrypted partial quotients at the same position, i.e., by performing $\mathtt{EQ}_{\mathbb{Z}}(\bar{x}_i, \bar{y}_i)$. If two partial quotients at a specific position i are the same, then $\mathtt{EQ}_{\mathbb{Z}}(\bar{x}_i, \bar{y}_i)$ outputs $\bar{1}$ and outputs $\bar{0}$, otherwise. Hence, if $x_i = y_i$ for all $i \in \{0, \ldots, n-1\}$ then every output of $\mathtt{EQ}_{\mathbb{Z}}$ is $\bar{1}$, and the product of all of these outputs becomes $\bar{1}$. On the other hand, if there is at least a pair of partial quotients different from each other, then because the output of corresponding $\mathtt{EQ}_{\mathbb{Z}}$ is $\bar{0}$, the product of the outputs is $\bar{0}$.

4.3.2 Complexity

We analyze the complexity of our equality circuit in terms of the computational complexity and the multiplicative depth. Let n be the size of the partial quotients and k be the number of partial quotients. For the readability, we write HA and HM to denote the number of homomorphic additions and multiplicatives, respectively. Note that for equality circuit for two n-bit integers, the necessary multiplicative depth is $\log n$ and the computational complexity is that 2 HA and $\log n\mathsf{HM}$.

Multiplicative Depth. Our equality circuit can be divided to two parts; one is executing equality test for the partial quotients of the same position and the other is multiplying all results of the equality test. Then, the multiplicative depth for the first part requires $\log n$ and the second part requires $\log k$ multiplicative depth. Therefore, in order to execute our equality circuit, the necessary multiplicative depth is $\log n + \log k = \log nk$.

Computational Complexity. Even though we have to execute equality tests for all partial quotients, it can be done at the same time since each test is independent which implies that the complexity is the same complexity of one equality test, 2 HA and nHM. Since the second part requires k homomorphic multiplications, the necessary complexity for the second part is kHM. In total, the computational complexity of our equality construction is that 2 HA and $(n+k)$HM.

4.4 Greater-Than and Less-Than

In this section, we translate the equality condition to the ciphertext domain. If two continued fractions are different, then two partial quotients of the same position should be different. However, in the ciphertext domain, we cannot find such a position and thus we should consider the worst case.

4.4.1 Construction

Two comparison circuits, denoted by $\mathtt{GT}_{\mathbb{R}}$ and $\mathtt{LT}_{\mathbb{R}}$, respectively, taking as input two encrypted CFs can be defined using integer-based circuits as follows.

$$\begin{aligned}
\mathtt{GT}_{\mathbb{R}}(\bar{X},\bar{Y}) =& \mathtt{GT}_{\mathbb{Z}}(\bar{x}_{n-1},\bar{y}_{n-1}) + \sum_{i=0}^{\frac{n-2}{n}} \mathtt{GT}_{\mathbb{Z}}(\bar{x}_{2i},\bar{y}_{2i}) \cdot \prod_{j\geq 2i}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j,\bar{y}_j) \\
&+ \sum_{i=0}^{\frac{n-2}{n}} \mathtt{LT}_{\mathbb{Z}}(\bar{x}_{2i+1},\bar{y}_{2i+1}) \cdot \prod_{j\geq 2i+1}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j,\bar{y}_j) \\
\mathtt{LT}_{\mathbb{R}}(\bar{X},\bar{Y}) =& \mathtt{LT}_{\mathbb{Z}}(\bar{x}_{n-1},\bar{y}_{n-1}) + \sum_{i=0}^{\frac{n-2}{n}} \mathtt{LT}_{\mathbb{Z}}(\bar{x}_{2i},\bar{y}_{2i}) \cdot \prod_{j\geq 2i}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j,\bar{y}_j) \\
&+ \sum_{i=0}^{\frac{n-2}{n}} \mathtt{GT}_{\mathbb{Z}}(\bar{x}_{2i+1},\bar{y}_{2i+1}) \cdot \prod_{j\geq 2i+1}^{n-1} \mathtt{EQ}_{\mathbb{Z}}(\bar{x}_j,\bar{y}_j)
\end{aligned}$$

Let $\bar{X}$ and $\bar{Y}$ be two encrypted CFs as above. Comparison between CFs moves in a left-to-right direction as comparing two integers. For two integers

written in binary, when the first difference appears at an index i in two binary expansions, an integer of the i-th bit being 1 is greater than the other integer. On the other hand, as shown in Theorem 4.2.1, when the first difference appears at the index i in two CF expansions, comparison between two CFs has to take into account the index i itself along with the i-th partial quotients; that is, if the index i is odd, the inequality of the two i-th partial quotients is the inequality of the two real numbers, but if the index i is even, the reversed inequality of the two i-th partial quotients becomes their inequality.

In the case where $n \neq m$, we need to take special care when comparing two encrypted CFs. As many special partial quotients, ∞, as the difference between n and m are added so that both CFs have the same number of partial quotients. For performance reason, we do not encrypt the special partial quotient. We then merely have to check if the index at which the first partial quotient appears is even or odd since ∞ is greater than integers.

4.4.2 Complexity

We analyze the complexity of our comparison circuits in terms of the computational complexity and the multiplicative depth. Since greater-than test and less-than test are based on the same principle and greater-than test result can be obtained from the less-than test result easily, the multiplicative depth and the computational complexity are the same. Similar to Chapter 4.3.2, let n be the size of the partial quotients and k be the number of partial quotients and HA and HM to denote the number of homomorphic additions and multiplicatives, respectively. Note that for comparison circuit for two n-bit integers, the necessary multiplicative depth is $\log n+1$ and the computational complexity is that $(n + 1 + \log n)\mathsf{HA} + (2n - 2)\mathsf{HM}$.

Multiplicative Depth. Our comparison circuits can also be divided to two parts; one is executing comparison tests for the partial quotients of the same position and the other part is arithmetics on the results of the comparison

tests. Then, the first part requires $\log n + 1$ multiplicative depth. For the second part, we have to multiply all equality tests which requires $\log k$ multiplicative depth and multiply it to every comparison test which requires $\log k$, too. In total, the multiplicative depth for our comparison circuit is $\log n + 1 + \log k + \log k = \log nk^2 + 1$.

Computational Complexity. Similarly in equality test, comparing two partial quotients of the same position can be done at the same time which implies the complexity of the first part is $(n + 1 + \log n)\mathsf{HA} + (2n - 2)\mathsf{HM}$. For the second part, we have to multiply all k equality tests and multiply it to k comparison tests. Moreover, there are 2 more homomorphic additions required. Therefore, the computational complexity of our circuits is $(n + 3 + \log n)\mathsf{HA} + (2n - 2 + 2k)\mathsf{HM}$.

Comparison. For the simplicity, let $\ell = \lceil \log x \rceil$ where $x = X \cdot 10^k$ and n is the number of partial quotients.

Our circuits are also designed on top of the integer-based circuits, but a main difference is that the baseline integer circuits are evaluated at encryptions under an FHE scheme instantiated with a small plaintext space, compared to the use of other representations. More precisely, we use a plaintext space of $\lceil \ell/n \rceil$ bits on average; however, as a trade-off, the number of ciphertexts amounts to the number of partial quotients, i.e., n ciphertexts of $\lceil \ell/n \rceil$-bit messages. Even though we have to perform equality test (or comparison test) per partial quotient, because each test can be carried out in parallel and with a smaller plaintext space, our suggestion is still more beneficial than other choices in the sense of running time.

Concretely, our equality (resp., greater-than) circuit is about 3 times (resp. 1.5 times) faster than an equality (resp., greater-than) circuit with decimal expansion. We defer the detailed results of experiment for each of input encoding to Section 4.5.3.

As for equality test, multiplying each result of comparison between encrypted partial quotients requires a multiplicative depth of $\lceil \log n \rceil$, in addition. Similarly, comparison test requires additional additional $2\lceil \log n \rceil$ ho-

momorphic multiplications and $\lceil \log n \rceil$ homomorphic additions. In Table 4.1, we report measurements with respect to the plaintext space, the number of ciphertexts, and multiplicative depth.

Table 4.1: Measurements between encodings

Algorithms	Parameters	CF (Ours)	Decimal
$\texttt{EQ}_{\mathbb{R}}$	Ptxt Space	$\mathbb{Z}_2^{\ell/n}$	$\mathbb{Z}_2^{\ell}$
	# Ctxt	n	1
	Depth	$\lceil \log \ell/n \rceil + \lceil \log n \rceil$	$\lceil \log \ell \rceil$
$\texttt{GT}_{\mathbb{R}}/\texttt{LT}_{\mathbb{R}}$	Ptxt Space	$\mathbb{Z}_2^{\ell/n}$	$\mathbb{Z}_2^{\ell}$
	# Ctxt	n	1
	Depth	$\lceil \log \ell/n \rceil + \lceil \log n \rceil + 2$	$\lceil \log \ell \rceil + 1$

4.5 Implementation

In this section, we present a list of experiments to evaluate the performance of homomorphic evaluation of comparison circuits on real numbers. We start by describing the test environment for reproducibility. Next, the approach used to select the scheme parameters is presented. Lastly, experiment settings alongside with results and analysis are provided for each experiment. The code is uploaded on my GitHub [27].

4.5.1 Environment

We use the BGV [12] FHE scheme to implement the equality and comparison circuits on integers and reals. A complete C++ implementation of the BGV is provided by the open source library `HElib` [47]. Both `HElib` and the number theoretic library `NTL` [70] (version 10.3.0) are used to facilitate the implementation. We also utilize the Open Multi-processing (OpenMP) [64] to boost the performance by issuing several threads to run concurrently on the available computing resources. Although, `HElib` supports Bootstrapping, we use a leveled variant of the BGV scheme that supports homomorphic evaluations

up to a predefined level. This is a common approach used when the circuit multiplication depth is known in prior to avoid the expensive Bootstrapping procedure.

We developed our implementation via C++ on a lightly loaded 64-bit machine equipped with 2 CPUs and 64 GB RAM, each CPU has 6 cores with hyper-threading enabled. The system can be though of as if there are available 24 processing units. The operating system is ArchLinux (4.8.13-1-ARCH) and the compiler used is GCC (6.3.1 20170109).

4.5.2 Scheme Parameters

In order to guarantee security, one needs to be vigilant while setting the cryptosystem parameters to insure that the fastest known attack requires unrealistic computational resources or time that is far beyond the lifetime of private data. We set the security parameter in HElib to 80-bit security level for all the experiments. The plaintext modulus is set to 2. The m-th cyclotomic polynomial is initiated automatically using "`FindM`" function in HElib. As will be shown in section 4.5.3, we vary the number and size of partial quotients. Afterwards, we choose the minimum circuit multiplication depth that guarantees correct results. A complete listing of the chosen parameters is shown in Table 4.2.

Table 4.2: Parameters used

Parameter	Value	Meaning
λ	80	security level
p	2	plaintext modulus
r	1	lifting power
c	3	number of columns in key switching matrix
d	0	embedding degree
L	5-11	multiplicative depth
n	4-8	number of partial quotients
$\lceil \frac{\ell}{n} \rceil$	3-5	bit length of a partial quotient

4.5.3 Experimental Results and Comparisons

In this section, we present a set of timing experiments for the implemented circuits. The testing methodology can be summarized as follows: real numbers are generated randomly and converted to continued fractions representation. Each partial quotient in $\mathbb{Z}$ is converted to binary form. Each bit is encoded as a polynomial, i.e. polynomials at this stage are simply constants in $\{0, 1\}$. We pack these polynomials and encrypt them in a single ciphertext in a SIMD fashion. Thus, a separate ciphertext is required for each partial quotient. After generating all the ciphertexts, the desired circuit can be evaluated homomorphically. We only measure the time required to evaluate the circuit. Key generation, encryption and decryption execution times are not counted for in all experiments. Chrono system functions are used to measure the elapsed time with microseconds precision. We run each experiment 10 times and the average execution time is always reported. To guarantee accurate timing figures and to avoid any caching effect, we develop a python script that executes the C++ binary and collects the evaluation time generated in each run. Thus, each run is completely independent from the rest.

In Table 4.3, the third column shows the average evaluation time in (sec) for the equality circuit. While varying the number and size of partial quotients, we prepared our experiment with the circuit's estimated multiplicative depth of $\lceil \log \ell/n \rceil + \lceil \log n \rceil$. However, for some cases, we used different multiplicative depth (L) from the estimated depth when the ciphertext includes extra multiplicative level. We think that one reason for this difference in the depths is that multiplicative depth is not always consumed in a single multiplication, but only consumed when there is enough large noise in the ciphertext. The fourth column of the table shows the average evaluation time in (sec) for the comparison circuit. Similarly, we also the number and precision of partial quotients and use a suitable multiplicative depth.

Lastly, we perform experiments for comparing the running time of our CF based circuits with other well-known circuits that employ decimal expansion for the same real numbers. To do this, we choose three random CFs by

Table 4.3: Average execution time for basic circuits

k	n	Equality test		Geater-than test	
		L	time (in sec)	L	time (in sec)
3	6	7	1.743	9	8.831
	7	7	2.075	9	9.141
	8	7	2.433	9	11.787
4	4	5	0.685	8	4.776
	5	7	1.483	9	6.424
	6	7	1.750	9	8.731
5	4	7	1.600	9	6.678
	5	7	2.072	9	6.954
	6	7	2.433	11	19.539

varying the number of partial quotients and express these numbers in decimal representation. These numbers are shown in the first column in Table 4.4. We encode each real number in continued fractions and decimal expansion. Then, we encrypt each encoding to generate two ciphertexts encrypting the same real number. The smallest FHE parameters are chosen to guarantee correct evaluations. The average running time of equality and greater-than circuits for the two ciphertexts is reported. It can be clearly seen that CF circuits outperform the decimal ones by at least a factor 3.36×.

Table 4.4: Comparison of running times between encodings

Input	in CF form		in Decimal form	
	Equality	Greater-than	Equality	Greater-than
$7.194444 = [7; 5, 6, 1]$	0.49 sec	2.77 sec	1.65 sec	3.76 sec
$6.313559 = [6; 3, 5, 3, 2]$	0.54 sec	2.96 sec	1.69 sec	4.52 sec
$15.322749 = [15; 3, 10, 6, 1, 2]$	0.59 sec	2.93 sec	1.71 sec	4.43 sec

4.6 Applications to Database Service

We can construct a wide range of applications by utilizing our FHE-based real number comparison technique. In this section, we list two promising applications among them: sorting a list of FHE-encrypted real numbers and processing database queries on FHE-encrypted databases.

4.6.1 Sorting

Sorting is one of the most fundamental topics in computer science which has been researched for very long time. Moreover, with the development of cloud service and cloud computing, sorting on encrypted database has also emerged in these days. For this, order-preserving encryption (OPE) [3] and order-revealing encryption (ORE) [9,24,57] are two appealing solutions for sorting on encrypted data in terms of efficiency. However, OPE and ORE would leak some information about the relative distance between the underlying messages. In particular, they could leak more information in specific setting, for example, a set of non-uniform data [38].

Comparison. On the contrary, FHE with semantic security does not leak any information about the underlying messages and thus FHE-based sorting leaks no information to a semi-honest server beyond that revealed by its output (e.g., the size of list).

There are some solutions for sorting on FHE-encrypted datasets, for example, see [22, 23]. Their protocols develop fully homomorphic circuits for comparison and swap operations and utilize them in conventional sorting algorithms. However, to our knowledge, they only support integer-type databases. Therefore, there are great restrictions to their use for the practical applications. For these reasons, we build a sorting algorithm by adopting our encrypted real number comparison algorithm designed in Section 4.2.1.

Swap. Given comparison circuits $\mathtt{GT}_{\mathbb{R}}$ and $\mathtt{LT}_{\mathbb{R}}$, it is easy to build a swapping algorithm of two encrypted values. This means that we can homomorphically sort FHE-encrypted real numbers only using our basic circuits. A swap circuit

receives a sequence of two numbers (in our cases, two real numbers) and outputs an in order sequence of the same size. Without loss of generality, we only focus on ascending order, but descending order can be obtained with minor changes. For two encrypted real numbers $\bar{X}$ and $\bar{Y}$, denoting by $[\bar{X}, \bar{Y}]$ that $\bar{X}$ and $\bar{Y}$ is stored in order, swap in ascending order is defined by:

$$\begin{aligned}\mathtt{Swap}(\bar{X}, \bar{Y}) := [\ &\mathtt{LT}_{\mathbb{R}}(\bar{X}, \bar{Y}) \cdot \bar{X} + \mathtt{LT}_{\mathbb{R}}(\bar{Y}, \bar{X}) \cdot \bar{Y}, \\ &\mathtt{GT}_{\mathbb{R}}(\bar{X}, \bar{Y}) \cdot \bar{X} + \mathtt{GT}_{\mathbb{R}}(\bar{Y}, \bar{X}) \cdot \bar{Y}\ ].\end{aligned}$$

Efficiency. Most of conventional sorting algorithms consist of comparison and swap. Theoretically, thus we can execute any sorting algorithm with our technique. However, since the performance of most FHE-based schemes is closely related to the multiplicative depth, much work has been put to reduce it.

In [22], Gizem et al. propose a new sorting algorithm, called direct sort and greedy sort, whose multiplicative depth is lower than existing sorting algorithms from $O(n \log^2 n)$ to $O(\log n + \log \ell)$ where n is the size of input list and ℓ is the bit size of elements. In order to sort n values, say $a_0, \ldots, a_{n-1}$, their protocol first constructs an $n \times n$ matrix whose component (i, j) is $\mathtt{GT}_{\mathbb{Z}}(a_i, a_j)$ for $0 \leq i, j < n$. Since the sum of i-th row of the matrix indicates the number of elements greater than the element a_i, it can sort all values in according to the sum. In particular, combining their techniques with merger sort, one can obtain a more efficient sorting algorithm.

Using our comparison circuits of real numbers, we can apply their technique to our setting. Similarly, we can build a comparison matrix whose component (i, j) is $\mathtt{GT}_{\mathbb{R}}(a_i, a_j)$ for $0 \leq i, j < n$ instead of $\mathtt{GT}_{\mathbb{Z}}(a_i, a_j)$ and we can sort an encrypted list of real numbers.

4.6.2 Private Database Queries

A database query refers to the process of retrieving data that satisfy a set of constraints from a database. Database queries are one of the core operations that make databases so useful. However, if users' data are very sensitive

such as medical or economical information, part (sometimes the whole) of the database holding such sensitive attributes needs to be protected from unauthorized access by for example encryption. For this reason, both the database and cryptography communities have shown great interest in providing privacy-preserving and secure query services over outsourced data, for short, private database query (PDQ).

PDQ techniques allow users to obtain desired information from encrypted databases without revealing information to the server. Searchable encryption (SE) [74] can be used to retrieve information as a result of queries; however, it can leak information about the plaintext messages like ORE and OPE. Furthermore, SE does not allow to evaluate encrypted polynomials.

In this work, we consider two simple types of PDQ, called *retrieval queries* and *aggregate queries*. Consider two queries over a medical database:

- What is the first name of all patients whose random plasma glucose is above 11.345009 mmol/l?
- What is the average age of patients whose random plasma glucose is above 13.10134 mmol/l?

To answer these queries, we have to identify records (or tuples) satisfying a given search condition and, if necessary, to compute over the result set. We notice that functional encryption such as SE, OPE and ORE cannot evaluate aggregate functions like `avg` and `min` over ciphertexts. This motivates the need to develop FHE-based PDQ protocols (e.g., [7, 25]) to support from simple to aggregate queries at a time.

Retrieval Queries. Retrieval queries consist of a list of target attributes, their owner database names, and a search condition. A way to offer reasonable performance is to encrypt only the private constants in the search condition. Then on receiving such a retrieval query, the cloud server can see which attributes in a given database are compared to the encrypted constants. In general retrieval queries can be again classified into conjunctive query and disjunctive query. Conjunctive queries require that all predicates in a

search condition hold true while disjunctive queries require that at least one predicate in a search condition hold true.

Assume that a database D has a schema $D(A_1, A_2, \ldots, A_\tau)$ where D is a database name and each A_i is the attribute name, and it has N encrypted tuples. For simplicity, we denote by $\texttt{Search}(A_i[j], \bar{X}_i)$ the i-th predicate to compare an encrypted constant $\bar{X}_i$ to an encrypted attribute value $\bar{Y}_{ij}$ of A_i for some $j \in \{1, \ldots, N\}$. Here we mean by "compare" either equality or greater-than test. Then we can describe retrieval queries over the encrypted database D as

$$\texttt{Conj}(\bar{X}_{i_1}, \ldots, \bar{X}_{i_\eta})[j] = \texttt{Search}(A_{i_1}[j], \bar{X}_{i_1}) \wedge \cdots \wedge \texttt{Search}(A_{i_\eta}[j], \bar{X}_{i_\eta})$$
$$\texttt{Disj}(\bar{X}_{i_1}, \ldots, \bar{X}_{i_\eta})[j] = \texttt{Search}(A_{i_1}[j], \bar{X}_{i_1}) \vee \cdots \vee \texttt{Search}(A_{i_\eta}[j], \bar{X}_{i_\eta}).$$

If the target attribute is A_k, then the output of conjunctive queries is given by $\{\bar{Y}_{k,j} \cdot \texttt{Conj}(\bar{X}_{i_1}, \ldots, \bar{X}_{i_\eta})[j]\}_{1 \le j \le N}$. In the same way, we can describe the output of disjunctive queries.

Aggregate Queries. Aggregate functions return a single result based on a group of data is formed based on retrieval queries. In general, aggregate functions include `sum`, `avg`, and `count`. Since we employ continued fraction to represent the real numbers, one may not be familiar to arithmetics with continued fraction. In 1972, Bill Gosper proposed the general arithmetic algorithm [45] between continued fractions. This algorithm allows arithmetics between two continued fractions as well as a continued fraction and a rational number.

The algorithm receives two real numbers X and Y represented by continued fraction and outputs $F(X, Y) := (a + bX + cY + dXY)/(e + fX + gY + hXY)$, where $a, b, c, d, e, f, g, h \in \mathbb{Z}$. For example, setting $b = 1, c = 1, e = 1$ indicates $F(X, Y) = X + Y$, setting $d = 1, e = 1$ indicates $F(X, Y) = X \cdot Y$, and setting $a = 2, b = 3, g = 4$ indicates $F(X, Y) = (2 + 3X)/4Y$. Since $F(X, Y)$ includes every primitive arithmetic, we can evaluate every primitive arithmetics as well as a two-variable linear fractional transformation.

In [28], the authors explore the possibility of employing FHE with Gosper

algorithm, which implies that we can evaluate two encrypted continued fraction. Theoretically, with Gosper algorithm, we can evaluate arbitrary polynomials using the same technique to the arbitrary polynomial instead of $F(X, Y)$. However, because it becomes quite complex when the degree of polynomial is more than 2, the algorithm is not suitable for aggregate queries in terms of efficiency. More specifically, Gosper algorithm contains a step for representing rational numbers $F(X, Y)$ using the partial quotients of X and Y which incurs high computational costs. For each partial quotient, one multiplication is required to reconstruct numerator and denominator, respectively. For this reason, we separately store both numerators and denominators together with their continued fractions to reduce computational overhead. Then arithmetic with numerators and denominators becomes much easier than Gosper algorithm which can considerably reduce the cost for constructing rational numbers.

Algorithm *Comparing Two Continued Fractions*

Input: $X = [x_0; x_1, \ldots, x_{n-1}]$ and $Y = [y_0; y_1, \ldots, y_{m-1}]$

Output: The result of the comparison of X and Y

```
1.  k ← the smallest index such that x_k ≠ y_k
2.  if ∃k then
3.     c ← (−1)^k (x_k − y_k)
4.     if c < 0 then
5.        return X < Y
6.     else
7.        return X > Y
8.  else
9.     if n = m then
10.       return X = Y
11.    else if n < m then
12.       if n is odd then
13.             return X < Y
14.       else
15.             return X > Y
16.    else
17.       if m is odd then
18.             return X < Y
19.       else
20.             return X > Y
```

Alg. 4.1: Comparison of two CFs in the clear.

Chapter 5

Algorithms for Integrity-based Homomorphic Evaluations

5.1 Overview and Related Works

Biometric authentications are seeing greater industrial deployment, including mobile payment systems such as Apple Pay and Alipay. Compared to other types of authentication (e.g., passwords and secure tokens), biometrics cannot be lost or forgotten and, in particular, users being authenticated should be present at the time and place of authentication. On the other hand, privacy loss in biometric authentication systems is substantially more serious than in other authentication systems because biometrics are difficult to be replaced once stolen. Most recently, hackers have stolen a total of 5.6 million fingerprint records from the U.S. government [2]. The stolen biometric databases could be used to fool certain systems. Thus, it is imperative to develop a solution with a far stronger protection of such data.

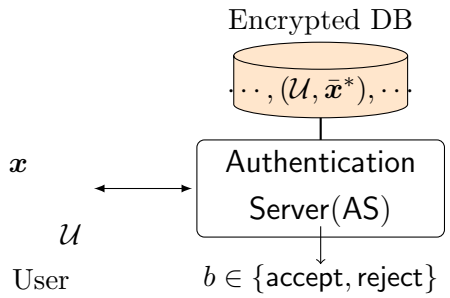

Figure 5.1: Our authentication framework.

Cryptographic protection. We investigate a secure biometric authentication method without relying on trusted hardware. Our approach is to encrypt and store a userâs biometric template in a server as in Figure 5.1. During authentication, the user sends an encrypted biometric attempt to the server, which authenticates by comparing two ciphertexts without decryption. Comparing ciphertexts without decryption is the main security property that our proposed scheme provides.

One may consider a trivial approach to store only hashed templates in the server through a one-way hash function such as SHA3 [62]. However, biometric inputs are not exactly the same every time they are captured due to scanning noise and so cannot have the same hashed values. Indeed, we have no hash function to map two slightly different inputs to the same value.

Related Works. One solution for protecting biometric data during authentication is to only store them in a user's device and to use hardware-based security mechanisms, as employed by the Fast Identity Online [1] method. However, this leads to another authentication problem between the user's device and a server, so this approach is not appropriate for some applications. Furthermore, trusted execution environments, such as ARM TrustZone [4], may not always be available and theoretical security guarantees cannot be provided, although they are not easy to compromise [60].

Alternatively, we could consider the method based on searching encrypted data by Song et al. [74]. However, this method is not satisfactory because

it needs to match two ciphertexts where the plaintexts differ slightly from each other. Other studies have recommended using functional encryption [41] based on cryptographic multilinear maps, but we are currently unaware of any secure cryptographic multilinear maps and it is not clear whether we will be able to obtain secure and efficient cryptographic multilinear maps in the near future.

Comparison with Previous Schemes. Recently, some studies have proposed homomorphic encryption techniques for securely computing matching algorithms as well as secure multiparty computation (SMC) techniques to guarantee the full privacy of biometric data. SMC provides the same functionality as our approach based on SHE, but it requires *interaction-intensive* computations. Kerschbaum et al. [54] suggested an SMC-based protocol, but it is only secure when all of the participants are honest. Erkin et al. [39] proposed a protocol using Paillier's cryptosystem, but it also requires that the participants are honest.

Several related studies have considered the secure computation of the HD using oblivious transfers (OTs) and garbled circuits, e.g., see [15, 17, 49, 51]. But these methods were restricted only to secure HD computation and storing biometrics openly.

Similar methods have been proposed in privacy-preserving research based on additive homomorphic encryption. Osadchy et al. [65] proposed a face identification protocol, but it is only secure when the participants are honest. They reported that online computation by a server required about 0.3 seconds for 900-bit values, but 213 seconds for offline computation. Blanton and Gasti [6] suggested an iris identification protocol based on a semi-honest model, but their OT-based protocol required $O(n)$ interactions between the user and the server. Blanton and Aliasgari [5] proposed solutions for iris identification based on predicate encryption, but their methods were only efficient with very small biometric templates.

Kulkarni and Namboodiri [56] presented an iris authentication scheme based on the SHE scheme described by Boneh et al. [8], but the online ex-

ecution time on the server was 58 seconds for 2048-bit iris data. Recently, Bringer et al. [16] showed that oblivious RAM techniques can be used in an iris identification protocol, but the service provider is openly provided with the user's biometrics. Karabat et al. [53] proposed an authentication protocol that only uses threshold homomorphic encryption, but their scheme required 6.1 seconds on the server side and 2.1 seconds on the user side.

5.2 Models and Settings

5.2.1 System Model and Participants

Our system is designed for an authentication server (AS, or server for short) to authenticate each user using the 1:1 method; therefore, our system comprises a user and an AS.

The user denoted by $\mathcal{U}$ has a binary feature $\boldsymbol{x}$ extracted from a biometric source. The server denoted by $\mathcal{S}$ has rich computational resources and storage, which allow the evaluation of arbitrary functions based on SHE ciphertexts, but it *cannot decrypt* the ciphertexts evaluated by itself as well as the ciphertexts given by the user.

5.2.2 Threat Model

We address two security-related goals. The first goal is that no AS should be able to learn anything about the biometric data contributed by users except for that revealed by the final result obtained after execution. Even if the AS and some users collude, they should not learn anything about the biometrics from other honest users excluding the final result and its implications. The second goal is that an impostor should not be able to fool the AS into believing that he is authentic (see [67] for details of the possible threats).

In this study, we focus on the following attacks to ensure secure authentication and privacy of the user's biometrics.

- Threat #1. Participants on the user side of our system are highly motivated to be malicious, and thus they make an honest AS output an acceptance at the conclusion of user authentication. Furthermore, they may collude with other arbitrary users to attempt to break through the systemÂs security.

- Threat #2. Some participants on the server side of our system are also motivated to behave maliciously and learn information about the private biometrics of honest users because these data could potentially be sold to an attacker, e.g., passport forgers. A typical example is a biometric database administrator corrupted by an attacker.

Assumptions. It is possible that a third entity (e.g., service providers such as Amazon) could develop a rich service using our authentication platform, which might allow collusion with the AS. However, we do not consider additional tools to defend against collusion, but we suggest that there is a high risk of a penalty when compromising a user's biometrics in this manner.

5.2.3 Security Model

A formal security definition is given in Section **??**, but we now provide an outline of the security and privacy properties exhibited by our authentication system.

Correctness. If all of the participants execute a given protocol faithfully, the verification result produced after executing the protocol is equal to the result obtained after submitting each honest participant to a biometric database in the open.

Privacy. The servers can only learn the verification result and its implications after executing a protocol.

Security. A dishonest user cannot persuade an honest AS into accepting him as an authentic user.

5.3 Integrity of Homomorphic Evaluations

5.3.1 Message Authentication Code

MAC is a cryptographic tool used to ensure data integrity. A MAC scheme denoted by $\mathsf{MAC} = (\mathtt{MKg}, \mathtt{Tag}, \mathtt{Vrf})$ is a triple of PPT algorithms as follows.

- $mk \leftarrow \mathtt{MKg}(1^\lambda)$. The key generation algorithm takes the security parameter λ and outputs a secret key $mk \in \{0,1\}^\lambda$ to generate a tag.
- $(x, \tau) \leftarrow \mathtt{Tag}(mk, x)$. As inputs, the algorithm takes a secret key mk and a message $x \in \mathsf{P}$, and outputs a tag τ along with the message x, where P is a message space.
- $b \leftarrow \mathtt{Vrf}(mk, x, \tau)$. The verification algorithm takes the secret key mk, a message x, and a tag τ as inputs, and outputs $b \in \{0,1\}$. If τ is a valid tag for x, then $b = 1$; otherwise, $b = 0$.

If a secret key from the MAC key generation function can be used only once (the MAC scheme can be forged otherwise), we call it an OTM scheme. An OTM is secure if an OTM withstands a chosen-message attack. A formal definition of security for the OTM is given as follows:

Definition 1. Consider the following experiment denoted by $\mathbf{Ex}_{\mathcal{A}}^{\mathsf{OTM}}(\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

1. The challenger $\mathcal{C}$ runs $\mathtt{MKg}(1^\lambda)$ and obtains a secret key mk for a MAC.
2. The adversary $\mathcal{A}$ adaptively determines a message x. In $\mathtt{Tag}$ queries, the adversary $\mathcal{A}$ sends x and receives a MAC tag $\tau \leftarrow \mathtt{Tag}(x)$.
3. After the adversary $\mathcal{A}$ decides that the query is over, $\mathcal{A}$ outputs (x^*, τ^*).
4. The game outputs 1 iff $\mathtt{Vrf}(mk, x^*, \tau^*) = 1$ and $x \neq x^*$.

Definition 2. An OTM scheme $\mathsf{OTM} = (\mathtt{MKg}, \mathtt{Tag}, \mathtt{Vrf})$ is *secure* if for all PPT adversaries $\mathcal{A}$,

$$\Pr\left[\mathbf{Ex}_{\mathcal{A}}^{\mathsf{OTM}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

5.3.2 Protocol Constructions

In this section, we provide two protocols, called a woodenman protocol and an ironman protocol. A woodenman protocol is secure under the Threat #1 as I defined before and an ironman protocol is secure under the Threat #1 and Threat #2.

A Woodenman Protocol

Our starting point is MAC. Our main observation is that the homomorphic property of SHE allows to run a MAC generation algorithm over ciphertexts. Specifically, given a SHE scheme $\mathsf{FHE} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Ev})$, we consider a function $\mathtt{Tag}^*$ such that for a MAC generation function $\mathtt{Tag}$,

$$\mathtt{Tag}^* \circ \mathtt{Enc}(d) = \mathtt{Enc} \circ \mathtt{Tag}(d).$$

Indeed, we take $\mathtt{Tag}^*$ as $\mathtt{Ev}_{ek}(\mathtt{Tag}, mk, \bar{d})$, which leads us to the description shown in Figure 5.2. This solution is attributable to a property of our settings where the server can be viewed simultaneously as the originator and the recipient.

User $\mathcal{U}(sk)$		AS $\mathcal{S}(mk, \bar{d})$
	$\xleftarrow{\bar{d}, \bar{\tau}}$	$\bar{\tau} = \mathtt{Ev}_{ek}(\mathtt{Tag}, mk, \bar{d})$
$d = \mathtt{Dec}_{sk}(\bar{d})$		
$\tau = \mathtt{Dec}_{sk}(\bar{\tau})$	$\xrightarrow{d, \tau}$	$b = \mathtt{Vrf}(mk, d, \tau)$

Figure 5.2: Protocol level properties of our basic idea.

However, this design strategy implies that our choice of the underlying MAC scheme determines the overall performance of a higher layer protocol that uses the scheme. Therefore, we need to use a highly efficient MAC scheme. In addition, the server will communicate with a number of users so it is necessary to provide a large amount of storage for all of the secret

keys. Thus, we apply an OTM variant in our protocol, i.e., a simple variant of the Simmons OTM [71], which requires only one modular multiplication. However, our variant does not include the modular reduction process because modular arithmetic over encryptions is highly expensive.

DESCRIPTION. We assume that the server has computed the distance $\bar{d}$ between an encrypted template and an encrypted attempt, where $\bar{d}$ denotes the encrypted result of the computation. We focus on the HD between two biometrics, but the details of HD computation are given in our discussion of the full authentication protocol.

Let R_ℓ be a set of $\ell = \ell(\lambda)$-bit integers. Given an encrypted HD value $\bar{d}$, the server chooses r_0 and r_1 uniformly at random in R_ℓ, before computing $\bar{\tau} = r_0 \cdot \bar{d} + r_1$. The server sets $mk = (r_0, r_1)$. The user can obtain $\tau = r_0 \cdot d + r_1$ by decrypting $\bar{\tau}$, but cannot recover d because it does not know the secret key mk. The server recovers d using its secret key mk and outputs $b = \mathtt{Vrf}(mk, \tau, d)$ by checking if $d = \frac{(\tau - r_1)}{r_0}$.

protocol *Ensuring integrity of a matching result*	
syntax: $\langle \mathcal{U}(sk), \mathcal{S}(\bar{d}) \rangle \rightarrow (\perp, (d, b))$ where $b \in \{0, 1\}$	
1. $[\mathcal{S}]$	$(r_0, r_1) \xleftarrow{\$} (R_\ell)^2$,
	$\bar{\tau} \leftarrow r_0 \cdot \bar{d} + r_1$
2. $[\mathcal{U} \leftarrow \mathcal{S}]$	$(\bar{d}, \bar{\tau})$
3. $[\mathcal{U}]$	$(d, \tau) \leftarrow \big(\mathtt{Dec}(\bar{d}), \mathtt{Dec}(\bar{\tau})\big)$
4. $[\mathcal{U} \rightarrow \mathcal{S}]$	(d, τ)
5. $[\mathcal{S}]$	Check if $d \stackrel{?}{=} \frac{\tau - r_1}{r_0}$

Protocol 5.1: Our Woodenman Scheme Π_{T1}

PERFORMANCE.

After the server obtains $\bar{d}$, it only needs to perform one addition and multiplication by constants. These operations are cheaper than other homomorphic operations (see Table 5.4). On the server's side, the most intense computation involves computing $\bar{d}$ by evaluating a matching function at the

encrypted biometrics. As described in Section 5.4.3, SIMD and ciphertext-packing techniques improve the performance of this evaluation significantly. We discuss the process optimization details in a later section.

An Ironman Protocol

Allowing decryption queries leads to attacks that can compromise biometric privacy. We discussed how to mount such an attack in Section **??**. In our solution to circumvent this attack, a user does not send d directly but a hint enclosing d instead, which allows the server to verify: (1) whether the d in the hint is authentic and (2) whether the d hidden in the hint is less than a matching threshold T.

The concept and description. To implement our solution, we require that all of the values decrypted by the user are raised to the power of a generator in the DL setting.

Let $\mathbb{G}$ be a cyclic group of a large prime order p where the DL assumption holds, and let h be a generator of $\mathbb{G}$. Let $H_1 : \{0,1\}^* \to R_\ell$ and $H_2 : \mathbb{G} \to \{0,1\}^{\mathsf{poly}(\lambda)}$ be random oracles. Then, a user chooses a random generator $h \in \mathbb{G}$ and computes $v = h^\tau$ with $\tau = \mathtt{Dec}(sk, \bar{\tau})$. According to the DL assumption, the server cannot construct an efficient algorithm to compute τ from v. The only remaining problem is how to allow the server to run the verification using the value v. Our solution is as follows: given a threshold T, we require that the user builds a set G of hashed values by $G = \{H_2(h^j) | j \in [T]\}$, and sends the value v along with the set G to the server.

The MAC verification $\mathtt{Vrf}(r_0, r_1, u, v)$ is straightforward: compute $h^* = (vu^{-r_1})^{\frac{1}{r_0}}$ and check if $H_2(h^*) \in G$, where $u = h$. Note that d or h^d are not given to the server. The full description of the protocol is shown in Protocol 5.2.

protocol *Ensuring privacy of biometrics*	
syntax: $\langle \mathcal{U}(sk), \mathcal{S}(\bar{d}) \rangle \rightarrow (\perp, b)$	
1. $[\mathcal{S}]$	$(r_0^*, r_1^*) \xleftarrow{\$} (\{0,1\}^*)^2$,
	$(r_0, r_1) \leftarrow (H_1(r_0^*), H_1(r_1^*))$,
	$\bar{\tau} \leftarrow r_0 \cdot \bar{d} + r_1$
2. $[\mathcal{U} \leftarrow \mathcal{S}]$	$(\bar{d}, \bar{\tau})$
3. $[\mathcal{U}]$	$(d, \tau) \leftarrow \left(\mathtt{Dec}(\bar{d}), \mathtt{Dec}(\bar{\tau})\right)$,
	$h \xleftarrow{\$} \mathbb{G}$ such that $\langle h \rangle = \mathbb{G}$
4. $[\mathcal{U} \rightarrow \mathcal{S}]$	$(u, v, G) \leftarrow (h, h^{\tau}, \{H_2(h^j) \mid j \in [T]\})$
5. $[\mathcal{S}]$	$h^* \leftarrow (vu^{-r_1})^{\frac{1}{r_0}}$,
	check if $H_2(h^*) \stackrel{?}{\in} G$

Protocol 5.2: Our Ironman Scheme Π_{T2}

PERFORMANCE. Additional computation overheads are incurred on the user side. The user performs T exponentiations in modulo p as well as T hash operations. A biometric is encrypted into κ ciphertexts where $\kappa = \lceil n/N \rceil$, so the total cost comprises κT exponentiations and κT hashings. However, this cost is significantly cheaper than that of homomorphic computations on SHE ciphertexts. The user's decryption costs are the same as before, i.e., 2κ times `Dec` operations.

5.3.3 Security Proof

In this section, I provide a proof of security of my proposal. At first, I prove a woodenman protocol is secure, that is, secure against Threat # 1. After then, I give a security proof for an ironman protocol, that is secure against Threat # 2.

Security Against Threat #1.

As described above, the server generates an encrypted tag $\bar{\tau}$ and receives the decryption of $\bar{\tau}$ with the help of the user. From a MAC security perspective, the user (i.e., decryptor) is modeled as an attacker. Thus, we propose a new security model where the user is allowed to produce any output that it wants to return. Next, we define security against one-time decryptor forgery attacks. This concept requires that even if the adversary possesses the decryption key for the challenge input, the adversary should not be able to return an incorrect output that passes verification.

Definition 3 (Decryptor Forgery Experiment). Let $\mathsf{FHE} = (\mathtt{KeyGen}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Ev})$ be a SHE scheme and $\mathsf{OTM} = (\mathtt{MKg}, \mathtt{Tag}, \mathtt{Vrf})$ be an OTM scheme, and let Π be a protocol associated with FHE and OTM. We consider the following experiment between a PPT adversary $\mathcal{A}$ and the server $\mathcal{S}$.

$$
\begin{aligned}
&\textit{Experiment } \mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi}(\mathsf{FHE}, \mathsf{OTM}, \lambda)\\
&\quad (pk, sk) \leftarrow \mathcal{A}^{\mathtt{KeyGen}(\cdot)}\\
&\quad \text{Run the protocol } \Pi\langle \mathcal{A}(sk), \mathcal{S}(\bar{d})\rangle\\
&\quad \mathcal{A} \text{ outputs } (d^*, \tau^*) \text{ such that } d^* \neq \mathtt{Dec}(sk, \bar{d})\\
&\quad \text{If } \mathtt{Vrf}(mk, d^*, \tau^*) = 1, \text{ output 1; else, output 0}
\end{aligned}
$$

For correctness, we require that given the private key sk, for every mk and every d such that $\mathtt{Vrf}(mk, d, \mathtt{Tag}(mk, d)) = b$, we have $\Pi\langle \mathcal{U}(sk), \mathcal{S}(\bar{d})\rangle \rightarrow (\perp, (d, b))$. Next, we require that an adversary controlling a user succeeds in the experiment $\mathbf{Ex}$ with a negligible probability.

Definition 4. A protocol Π is *secure* against decryptor-forgery attacks if it is correct and if a negligible function $\mathsf{negl}(\cdot)$ exists for every PPT algorithm $\mathcal{A}$ such that

$$\Pr\left[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi}(\mathsf{FHE}, \mathsf{OTM}, \lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

We argue that the Woodenman protocol Π_{T1} is secure against a corrupted user. In Lemma 1, we first show that our OTM variant is secure against one-time chosen-message forgery. Then, we prove Theorem 1 using the lemma.

Lemma 1. Let R_ℓ be a set of ℓ-bit integers. Given d and τ such that $\tau = r_0 \cdot d + r_1$ for two integers r_0, r_1 chosen uniformly from R_ℓ,

$$\Pr_{r_0,r_1}[\tau = r_0 \cdot d + r_1] < 2^{2-\ell},$$

where the probability is taken over the random choice of r_0 and r_1 from R_ℓ.

Proof. d and $\tau = r_0 \cdot d + r_1$ are given, so we can obtain a set of candidate points r_0 and r_1. In particular, by the extended Euclidean algorithm for d and τ, unique $\tilde{r}_0$ and $\tilde{r}_1$ exist such that $\tau = \tilde{r}_0 d + \tilde{r}_1$ with $\tilde{r}_1 < d$ and $\tilde{r}_0 \geq 2^{\ell-1}$, so this can be transformed into

$$\begin{aligned}\tau &= \tilde{r}_0 d + \tilde{r}_1 \\ &= (\tilde{r}_0 - 1)d + (d + \tilde{r}_1) = (\tilde{r}_0 - 2)d + (2d + \tilde{r}_1) = \cdots \\ &= 2^{\ell-1}d + (\tilde{r}_0 - 2^\ell)d + \tilde{r}_1.\end{aligned}$$

We can see that these are all candidates for r_0 and r_1, and thus the number of candidates is $\tilde{r}_0 - 2^\ell + 1$. Furthermore, $\tilde{r}_0$ is a quotient of τ and d, $\tilde{r}_0 = \lfloor \tau/d \rfloor$. Therefore, the probability of choosing the correct pair of r_0 and r_1 is $\frac{1}{\tilde{r}_0 - 2^{\ell-1}+1} = \frac{1}{\lfloor \tau/d \rfloor - 2^{\ell-1}+1}$. Moreover, because r_0 is chosen randomly from R_ℓ, we can assume that $r_0 \approx 3/2 \cdot 2^{\ell-1} = 3 \cdot 2^{\ell-2}$, and thus the probability of this assumption becomes

$$\Pr_{r_0,r_1}[\tau = r_0 \cdot d + r_1] \approx \frac{1}{2^{\ell-2}+1} < \frac{1}{2^{\ell-2}}.$$

Thus, we may conclude the lemma. □

Before proceeding to the next step, we need to define two random variables, dirty and verify. Intuitively, $\mathsf{dirty} = 1$ if a user modifies the decryptions of a pair $(\bar{d}, \bar{\tau})$ and $\mathsf{verify} = 1$ if the protocol outputs an accept.

Definition 5. We define a random variable dirty such that $\mathsf{dirty} = 1$ if and only if $d^* \neq \mathtt{Dec}(\bar{d})$ or $\tau^* \neq \mathtt{Dec}(\bar{\tau})$. In addition, we define verify such that $\mathsf{verify} = 1$ if and only if the honest server outputs an accept (i.e., $b = 1$) at the end of the protocol Π_{T1}.

Theorem 1 states that our basic protocol Π_{T1} is secure against Threat #1 except for a negligible probability of the security parameter λ.

Theorem 1. Let R_ℓ be a set of ℓ-bit integers. Assume that the underlying OTM scheme is secure against one-time existential forgery attacks. Then, for every PPT adversary $\mathcal{A}$, a negligible function $\mathsf{negl}(\cdot)$ exists such that

$$\Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}}(\mathsf{FHE},\mathsf{OTM},\lambda)=1] \leq 2^{3-\ell}+\mathsf{negl}(\lambda).$$

Proof. To prove the theorem, it is sufficient to show that

$$\Pr\left[\mathsf{dirty}=1 \wedge \mathsf{verify}=1\right] \leq \mathsf{negl}(\lambda).$$

Thus, we prove that we can construct an efficient algorithm to forge our OTM scheme with nonnegligible probability, where we assume that an adversary can succeed in passing the protocol Π_{T1} with a nonnegligible probability by using (d^*,τ^*) such that $d^* \neq \mathtt{Dec}(\bar{d})$ or $\tau^* \neq \mathtt{Dec}(\bar{\tau})$. The remainder of the proof follows directly by a standard reduction argument; however, the calculation of the success probabilities can be quite tedious.

We proceed to construct an OTM adversary $\mathcal{A}_{\mathrm{otm}}$, which works as follows. Let $\mathcal{A}$ be an adversary for the protocol Π_{T1} such that $\Pr\left[\mathsf{dirty}=\mathsf{verify}=1\right] = \epsilon(\lambda)$.

The adversary $\mathcal{A}_{\mathrm{otm}}$. Based on the input 1^λ and the values (d,τ) from the challenger,

1. $\mathcal{A}_{\mathrm{otm}}$ invokes $\mathcal{A}$ based on input 1^λ and outputs (pk,ek).
2. $\mathcal{A}_{\mathrm{otm}}$ interacts with $\mathcal{A}$ and acts as the honest server in the protocol, as follows:
   (a) $\mathcal{A}_{\mathrm{otm}}$ sets up the OTM key mk of the honest server in the protocol and sets $\bar{d} = \mathtt{Enc}_{pk}(d)$, but computes $\bar{\tau} = \mathtt{Enc}_{pk}(\tau)$ without the key mk, where (d,τ) is given by the challenger.
   (b) $\mathcal{A}_{\mathrm{otm}}$ sends the pair $(\bar{d},\bar{\tau})$ to the adversary $\mathcal{A}$.

3. After receiving a pair of decryptions (d^*, τ^*), $\mathcal{A}_{\mathrm{otm}}$ checks if $d = d^*$. If $d = d^*$, then $\mathcal{A}_{\mathrm{otm}}$ outputs a pair of random values $(r_0, r_1) \in (R_\ell)^2$. Otherwise, if $d \neq d^*$ and $\tau \neq \tau^*$, it outputs $\left(r_0 = \frac{\tau - \tau^*}{d - d^*}, \tau - d \cdot r_0\right)$; however, if $d \neq d^*$ and $\tau = \tau^*$, it outputs $(0, \tau)$.

The intuitive explanation of the attack strategy by $\mathcal{A}_{\mathrm{otm}}$ is straightforward. Next, we prove that $\mathcal{A}_{\mathrm{otm}}$ outputs the correct pair of values (r_0, r_1) with a probability $\epsilon(\lambda)(1 - \mathsf{negl}(\lambda))$, which is nonnegligible if $\epsilon(\lambda)$ is nonnegligible. Thus, we define Fail as the event, where $\mathcal{A}_{\mathrm{otm}}$ outputs a pair of random values during this attack. We have

$$\begin{aligned} rCl\, \Pr[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda) = 1] &= \Pr[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda) = 1 | \neg\mathsf{Fail}] \cdot \Pr[\neg\mathsf{Fail}] + \\ &\quad \Pr[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda) = 1 | \mathsf{Fail}] \cdot \Pr[\mathsf{Fail}]. \quad (5.3.1) \end{aligned}$$

We can see that $\Pr[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}},\Pi_{T1}}(1^\lambda) = 1 | \mathsf{Fail}] = 2^{2-\ell}$ by the definition of Fail, and the probability that $\mathcal{A}_{\mathrm{otm}}$ outputs an incorrect pair of values on the condition that Fail does not occur is negligible at most. Thus, we have

$$\Pr[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda) = 1 | \neg\mathsf{Fail}] \geq 1 - \mathsf{negl}(\lambda)$$

for a negligible function $\mathsf{negl}(\cdot)$.

Now, we only have to compute $\Pr[\mathsf{Fail}]$ and $\Pr[\neg\mathsf{Fail}]$ to evaluate Eq. (5.3.1). We compute $\Pr[\neg\mathsf{Fail}]$ by

$$\begin{aligned} &\Pr[\neg\mathsf{Fail}] = \\ &\quad \Pr[\neg\mathsf{Fail} | \mathsf{dirty} = \mathsf{verify} = 1] \cdot \Pr[\mathsf{dirty} = \mathsf{verify} = 1] + \\ &\quad \Pr[\neg\mathsf{Fail} | \mathsf{dirty} = 0 \vee \mathsf{verify} = 0] \cdot \Pr[\mathsf{dirty} = 0 \vee \mathsf{verify} = 0]. \end{aligned}$$

According to our assumption regarding $\mathcal{A}$, we have $\Pr[\mathsf{dirty} = \mathsf{verify} = 1] = \epsilon(\lambda)$. Hence, it follows that $\Pr[\mathsf{dirty} = 0 \vee \mathsf{verify} = 0] = 1 - \epsilon(\lambda)$. Next, if $\mathsf{dirty} = \mathsf{verify} = 1$, then $\mathcal{A}_{\mathrm{otm}}$ outputs a pair of random values only when $d = d^* \wedge \tau \neq \tau^*$. Thus,

$$\Pr[\neg\mathsf{Fail} | \mathsf{dirty} = \mathsf{verify} = 1] = 1 - 2^{2-\ell}.$$

By contrast, if $\mathsf{dirty} = 0$ or $\mathsf{verify} = 0$, then $\mathcal{A}_{\mathrm{otm}}$ always outputs a pair of random values. Thus, $\Pr[\neg\mathsf{Fail}|\mathsf{dirty} = 0 \vee \mathsf{verify} = 0] = 0$. By combining these, we have

$$\Pr[\neg\mathsf{Fail}] = (1 - 2^{2-\ell})\epsilon(\lambda) \text{ and } \Pr[\mathsf{Fail}] = 2^{2-\ell}\epsilon(\lambda).$$

By entering this into Eq (5.3.1), we have

$$\begin{aligned}
&\Pr[\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda) = 1] \\
&\qquad = (1 - \mathsf{negl}(\lambda))(1 - 2^{2-\ell})\epsilon(\lambda) + 2^{4-2\ell}\epsilon(\lambda) \\
&\qquad = \epsilon(\lambda)\left(1 - \mathsf{negl}(\lambda) - 2^{2-\ell}(1 - \mathsf{negl}(\lambda) - 2^{2-\ell})\right) \\
&\qquad = \epsilon(\lambda)(1 - \mathsf{negl}(\lambda)) - \mathsf{negl}^\star(\lambda)
\end{aligned}$$

for a negligible function $\mathsf{negl}^\star(\cdot)$. Thus, if $\epsilon(\lambda)$ is nonnegligible, then $\mathcal{A}_{\mathrm{otm}}$ succeeds in $\mathbf{Ex}^{\mathrm{otm}}_{\mathcal{A}_{\mathrm{otm}}}(1^\lambda)$ with a probability of $\epsilon(\lambda)(1 - \mathsf{negl}(\lambda))$, which is also nonnegligible. However, this contradicts Lemma 1.

We have proved that $\Pr[\mathsf{dirty} = \mathsf{verify} = 1]$ is negligible. For notational convenience, we assume that the experiment $\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}}$ takes $(\mathsf{FHE}, \mathsf{OTM}, \lambda)$ as inputs. To show the remainder of the proof, we observe that

$$\begin{aligned}
&\Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1] \\
&\qquad = \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 1] + \\
&\qquad\quad \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] \\
&\qquad = \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 1 \wedge \mathsf{verify} = 1] + \\
&\qquad\quad \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 1 \wedge \mathsf{verify} = 0] + \\
&\qquad\quad \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] \\
&\qquad \leq \Pr[\mathsf{dirty} = \mathsf{verify} = 1] + \\
&\qquad\quad \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{verify} = 0] + \\
&\qquad\quad \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] \\
&\qquad \leq \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{verify} = 0] + \\
&\qquad\quad \Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] + \mathsf{negl}(\lambda).
\end{aligned}$$

We can see that $\Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{verify} = 0] = 2^{2-\ell}$ and $\Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1 \wedge \mathsf{dirty} = 0] = 2^{2-\ell}$, so we conclude that

$$\Pr[\mathbf{Ex}^{\mathrm{dfa}}_{\mathcal{A},\Pi_{T1}} = 1] \leq 2^{3-\ell} + \mathsf{negl}(\lambda).$$

This completes the proof of the theorem. □

Security Against Threat #2.

We define a precise security model for analyzing the security of Protocol 5.2. We should note that it is not necessary to allow the adversary full access to a decryption oracle, as observed above. Nevertheless, the protocol seems to be useful in allowing the attacker to extract some biometric information because it reveals details in Step 5 of the protocol Π_{T2}.

To characterize privacy, we need to modify protocol Π given in Section 5.3.2, and we define a new ideal functionality $\mathcal{F}$ as

$$\widetilde{\Pi}\langle \mathcal{U}(sk), \mathcal{S}(\bar{d})\rangle \rightarrow (\perp, b),$$

where $b \in \{0, 1\}$. We note that d does not appear in the output from the server.

Our privacy requirement captures the notion of protecting the biometric templates of users during the execution of protocols. In the secure computation model, participants have their own private inputs and they want to evaluate a desired functionality $\mathcal{F}$ based on their inputs, but without revealing any information except the outputs and their unavoidable implications [43].

Intuitively, the following two scenarios should be indistinguishable in a computational sense: (1) securely computing $\mathcal{F}$ by executing a protocol; and (2) privately sending their private inputs to a trusted party, who then computes $\mathcal{F}$ and privately returns the outputs to each participant. This formalization of secure computing is referred to as the simulation-based approach. In the standard simulation-based privacy proof technique, given a well-defined privacy-leakage, a simulator running in polynomial time can generate a transcript that is indistinguishable from the output of the real protocol. If an

efficient simulator exists, then an adversary cannot learn any additional information other than the defined leakage. The simulator must perform its task without knowing the private information of the participant who wants to proves his authenticity.

We now argue that our ironman protocol Π_{T2} is secure against Threat #2. In this proof, we show that when a participant is corrupted (either user or server), a simulator exists denoted by $\mathcal{A}$, which can produce a view that is statistically indistinguishable from the view of the participant who interacts with the honest counterpart. Assuming that one participant is corrupted, we build an efficient simulator with access to the public input and private materials (e.g., secret key and biometrics) of the corrupted participant. In addition, the simulator knows the public output.

It should be noted that the proposed protocol ensures computational privacy for both the user and the server because the underlying SHE scheme provides IND-CPA security.

Theorem 2. Assuming that the SHE scheme provides IND-CPA security, protocol Π_{T2} in Protocol 5.2 is secure in the presence of semi-honest adversaries.

First, we outline the solution by showing how we can construct an efficient simulator $\mathcal{A}$. The first case (say, Case I) allows the attacker (i.e., the simulator) to learn the decryption key of the SHE scheme and the corrupted user's biometric. Hence, the simulator can output a view which is indistinguishable from that in the real protocol executions by using the ciphertext indistinguishability of SHE. In the second case (say, Case II), the simulator knows the secret key of a MAC scheme and all the decryptions of the matching results enclosed in the DL setting. Thus, the adversary cannot distinguish the output of the simulator from that of the real protocol.

Proof. We show that when a participant is corrupted, a simulator can produce a view for the adversary, which is computationally indistinguishable from the view obtained after executing the real protocol based on its secret inputs as well as the public information.

Case I: User is corrupted. In this case, we prove that our protocol is secure when a user is corrupted. The simulator $\mathcal{A}$ has the decryption key sk of the user and knows the user's biometrics, including other public information generated by the protocol.

Now, the simulator constructs a view for the user, which is computationally indistinguishable from that the user observes when interacting with the honest server based on these values. The simulator works as follows.

1. The simulator obtains a pair of attempt and template biometrics $(\boldsymbol{x}, \boldsymbol{x}^*)$, and the identity $\mathcal{U}$.

2. The simulator computes $\bar{\boldsymbol{x}} = \texttt{Enc}(pk, \boldsymbol{x})$ and $\bar{\boldsymbol{x}}^* = \texttt{Enc}(pk, \boldsymbol{x}^*)$.

3. The simulator computes $\delta = s_0 \cdot d + s_1$ for randomizers $s_0, s_1 \in R_\ell$, and the HD value d between $\boldsymbol{x}$ and $\boldsymbol{x}^*$.

4. The simulator encrypts the value δ into $\bar{\delta} = \texttt{Enc}(pk, \delta)$ and the encryption $\bar{\delta}$ is the simulated output.

Each step of the proposed authentication protocol for the simulator is simulated, which completes the simulation for the compromised user. The transcript is consistent and computationally indistinguishable from the userÂs view when interacting with the honest server.

Case II: Server is corrupted. In this case, we prove that our protocol is secure when a server is compromised by an attacker in the real-world protocol. The simulator $\mathcal{A}$ has the MAC key $mk = (r_0, r_1)$ of the server as well as the user's encrypted biometric attempt $\bar{\boldsymbol{x}}$ and template $\bar{\boldsymbol{x}}^*$ from the protocol. Now, we can construct a simulator that runs as follows.

1. The simulator computes $\bar{d} = \sum_{i=0}^{n-1} (\bar{x}_i - \bar{x}_i^*)^2$.

2. The simulator outputs an encryption $\bar{\delta} = r_0 \cdot \bar{d} + r_1$.

3. After receiving (u, v, G) from the honest user, the simulator runs the remaining steps of the protocol.

4. The simulator outputs his choice of $\beta \in \{0, 1\}$.

Consequently, each step of the proposed authentication protocol is simulated for the simulator, which completes the simulation for the corrupted server. The transcript is consistent and indistinguishable from the user'Âs view because u, v are random elements in the DL group, which is uniformly distributed. Moreover, the set G only contains elements in the DL group $\mathbb{G}_p$. □

5.4 Application to Biometric Authentication

In this section, we describe our biometric authentication protocol by including the enhancements proposed in Section **??**. We argue that our proposed protocol satisfies the security requirements, and we analyze the computation and communication overheads.

5.4.1 How Ghostshell Works

Ghostshell comprises three phases: *setup*, *enrollment*, and *match*. In the setup phase, Ghostshell fixes the system settings by running the key generation algorithm for each underlying cryptographic scheme (see Section **??**). Ghostshell then has a user extract its biometric template, encode the template as a binary string, and then store the encrypted template in the server. In subsequent uses, the user's attempted biometric is compared with the encrypted template.

In the following, we provide detailed descriptions of each phase.

SETUP. Obtaining the security parameter λ, a user generates a pair of keys (pk, sk) for the underlying SHE scheme and determines the system parameters for other primitives (MAC and FKA).

Our authentication system is designed to run on a symmetric variant of SHE, where the server cannot generate an encryption of its choice because it does not know the secret key sk, but it still can perform homomorphic

evaluations using the public key pk. This approach provides several benefits: in terms of security, the server is not allowed to encrypt arbitrary matching results; and in terms of performance, this approach helps to slightly reduce the computational cost.

ENROLLMENT. The user first extracts a biometric feature $\boldsymbol{x}^*$, computes its SHE ciphertext $\bar{\boldsymbol{x}}^* = \mathtt{Enc}(sk, \boldsymbol{x}^*)$, and sends $(\mathcal{U}, \bar{\boldsymbol{x}}^*)$ to the server. After receiving it, the server stores $(\mathcal{U}, \bar{\boldsymbol{x}}^*)$ in its database.

To ensure greater security against the device theft attack discussed in Section 5.3.2, we may encrypt all of the interactions between two participants using a session key determined by an FKA protocol. In this case, the user encrypts $(\mathcal{U}, \bar{\boldsymbol{x}}^*)$ with a symmetric encryption scheme and the server stores $(\mathcal{U}, \bar{\boldsymbol{x}}^*)$ after decryption under the session key.

MATCH. In the first step during this phase, the user encrypts an attempted biometric into $\bar{\boldsymbol{x}} = \mathtt{Enc}(sk, \boldsymbol{x})$ and sends $(\mathcal{U}, \bar{\boldsymbol{x}})$ to the server.

- After receiving it, the server retrieves $\bar{\boldsymbol{x}}^*$ denoted by $\mathcal{U}$ and computes $\bar{d} = \mathsf{HD}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{x}}^*)$. Next, the server and the user jointly execute the Π_{T2} protocol given in Protocol 5.2, i.e.,

$$\Pi_{T2}\langle \mathcal{U}(sk), \mathcal{S}(\bar{d})\rangle \rightarrow (\perp, b = \{\mathsf{accept}, \mathsf{reject}\}).$$

5.4.2 Analysis

PERFORMANCE. Let n be the bit length of $\boldsymbol{x}$ and $\boldsymbol{x}^*$. Assuming a SHE instantiation with N plaintext slots, we only need to keep $\kappa = \lceil n/N \rceil$ SHE ciphertexts instead of n SHE ciphertexts for an n-bit biometric. The user invokes $\mathtt{Enc}$ κ times for the enrollment phase. It is clear that the performance of the match phase is subject to the performance of the subprotocol Π_{T2}. We note that most of the server's time is consumed for performing homomorphic evaluations to compute a HD value. Thus, the next section focuses on the optimization of our algorithm to compute the HD on encrypted biometrics. By contrast, the user's most expensive computation is SHE decryption.

SECURITY. We examine the security of our protocol. In the enrollment phase, biometric privacy is protected by our SHE encryption. In the match phase, a user corrupted by an attacker cannot fool the AS according to Theorem 1. Theorem 2 states that an attacker of the server side cannot learn any information about the user's biometrics except for the verification result and its unavoidable implications. In conclusion, Ghostshell provides security against Threats #1 and #2.

5.4.3 Optimization

In this section, we describe our optimization techniques. These techniques demonstrate feasibility of our solution for practical deployment in real-world systems. Our optimization techniques have two different objectives: speeding up the computational time and reducing the transmission costs.

Speeding Up Computations

Computing the HD between two ciphertexts for n-bit biometrics is the most expensive computation in Ghostshell. A naïve computation of the HD based on ciphertexts requires n homomorphic multiplications and subtractions, and $n-1$ homomorphic additions (see Eq. (**??**)). To improve the computational efficiency, we specifically focus on reducing the number of multiplications of ciphertexts because this operation is about 280× slower than homomorphic addition.

Our technique builds on the SIMD techniques introduced by Smart and Vercauteren [73] (journal version of [72]), who suggested that a plaintext space can be treated as a partition of plaintext spaces of small size, which are called slots, and that a ciphertext carries a vector of plaintexts instead of a plaintext. By adding (resp., multiplying) these packed ciphertexts, we can perform the component-wise addition (resp., multiplication) of two vectors of plaintexts.

Obviously, ciphertext packing and SIMD operations allow efficient ho-

momorphic evaluations of multiple data simultaneously. Multiplication and subtraction of the HD circuit benefits from these two techniques. The problem that we need to address is that the HD circuit also requires the sum of all the components. However, no intrinsic operation supports addition and multiplication across different positions (or slots). Thus, the question is how do we efficiently compute the sum over plaintext slots?

We resolve this problem by using automorphisms as a technique for moving values between the different slots in a given plaintext vector. In particular, performing the automorphism $X \mapsto X^i$ means that a vector of messages (cyclically) rotates to the right i times. We denote the automorphism mapping X to X^i by σ^i and a vector of N bits by $\boldsymbol{w} = (w_0, \ldots, w_{N-1})$.

A simple method for computing the HD for $\boldsymbol{w}$ is to add all of the vectors $\sigma^i(\boldsymbol{w})$ for $i \in [N-1]$. This naïve method requires $N-1$ homomorphic additions and automorphisms, so the complexity remains $O(N)$. To reduce this computational cost, we use a tree structure and we proceed recursively with each of 2^i elements, which yields a binary tree of depth $\lceil \log N \rceil$, where log is the binary logarithm. Based on this method, we can compute the HD value of two independent vectors with one homomorphic multiplication, one homomorphic subtraction, and $O(\log N)$ homomorphic additions and automorphisms. We note that the homomorphic automorphism does not change the noise estimate.

Example 1. Let $N = 8$ and $\boldsymbol{w}_0 = (w_0, w_1, \ldots, w_7)$. If we let $\boldsymbol{w}_1 := \sigma^1(\boldsymbol{w}_0) = (w_7, w_0, \ldots, w_6)$, then we have $\boldsymbol{w}_0 + \boldsymbol{w}_1 = (w_0 + w_7, w_0 + w_1, \ldots, w_6 + w_7)$. Similarly, we obtain $\boldsymbol{w}_2 = (w_5 + w_6, \ldots, w_4 + w_5)$ with $\sigma^2(\boldsymbol{w}_0 + \boldsymbol{w}_1)$ and $\boldsymbol{w}_3 = (w_1 + w_2 + w_3 + w_4, \ldots, w_0 + w_1 + w_2 + w_3)$ with $\sigma^4(\boldsymbol{w}_2)$. Then, we can obtain the HD value $\sum w_i$ by $\sum_{i \in [3]} \boldsymbol{w}_i$.

A Practical Version of the Ironman Protocol

Theoretically, this idea holds for arbitrary values of N. However, the other parameters become bigger when the number of slots N is larger, which could slow down our system. Thus, we have to choose a modest value of N as a

trade-off between the size of the associated parameters and the BGV instantiation's performance using the `HElib` library [47]. To achieve a balance, we take $N = 630$ and $m = 8191$, where m is a system parameter and the m-th cyclotomic polynomial $\Phi_m(X)$ is used to determine the polynomial rings on which the BGV scheme runs.

$n \geq 2048 > N$ for some standard biometric features, so when we implement the ironman protocol Π_{T2}, we need to maintain one or more ciphertexts at a time. For example, four SHE ciphertexts are given for a 2400-bit iris code when $N = 630$; thus, the server should evaluate the HD circuit for the required number of ciphertexts, where the resulting HD value is split and stored in each ciphertext in order. This is why we reformulated the ironman protocol.

As mentioned above, let $\kappa = \lceil \frac{n}{N} \rceil$ denote the minimum number of ciphertexts for carrying a HD value. We only describe the differences compared with the original protocol, as shown in Protocol 5.3. We use $\bar{d} = (\bar{d}[0] \parallel \cdots \parallel \bar{d}[\kappa - 1])$ to denote the splitting of a resulting HD value $\bar{d}$ into κ ciphertexts $\bar{d}[0], \ldots, \bar{d}[\kappa - 1]$, and similarly, we set $u_j = h_j, v_j = {h_j}^{\tau[j]}$, and $G_j = \left\{H_2({h_j}^l)\right\}$ for all $l \in [T_\kappa]$ (see below for T_κ).

protocol Practical Variant of the Ironman Protocol	
syntax: $\langle \mathcal{U}(sk), \mathcal{S}(\bar{x}) \rangle \rightarrow (\perp, b)$ where $b \in \{0,1\}$	
1. $[\mathcal{S}]$	Compute $(r_0[j], r_1[j])_{j \in [\kappa-1]}$ likewise $\forall j \in [\kappa-1], \bar{\tau}[j] \leftarrow r_0[j] \cdot \bar{d}[j] + r_1[j]$
2. $[\mathcal{U} \leftarrow \mathcal{S}]$	$\{(\bar{d}[i], \bar{\tau}[i])\}_{i \in [\kappa]}$
3. $[\mathcal{U}]$	For all $j \in [\kappa-1]$, obtain $(d[j], \tau[j])$, $h_j \xleftarrow{\$} \mathbb{G}$ such that $\langle h_j \rangle = \mathbb{G}$
4. $[\mathcal{U} \rightarrow \mathcal{S}]$	$\{(u_j, v_j, G_j)\}_{j \in [\kappa-1]}$
5. $[\mathcal{S}]$	$\forall j \in [\kappa-1], h_j^* \leftarrow \left(v_j {u_j}^{-r_1[j]}\right)^{\frac{1}{r_0[j]}}$, check if $H_2(h_j^*) \stackrel{?}{\in} G_j$

Protocol 5.3: Adaptation of Π_{T2} to the implementation

Discussion. As described in a previous study [18], we use a matching threshold of $T = 600$ (approximately 30% of the 2048-bit iris code) based on the HD. However, we are restricted to carrying a κ-th of an iris code in a SHE ciphertext, so we use a threshold of T scaled by κ, which is denoted by T_κ. As a result, we need to consider the following practical issues.

- We compute a partial HD value per ciphertext and then compare the resulting HD value with the scaled threshold T_κ. Thus, we set the scaled threshold as $T_\kappa = 150$ for $\kappa = 4$. We consider that an iris code is encrypted separately and sent in four small pieces. There are $\binom{604}{5} \approx 2^{40}$ possible ways that the sum of all HD values obtained from two small pieces of encryptions are less than or equal to $T = 600$. During a session, a dishonest user is very unlikely to correctly manipulate all of the pieces of HD values at its disposal.

- Next, if we use $\mathbb{Z}_{2^t}$ as the plaintext space, then the FPR is $\left(\frac{T_\kappa}{2^t}\right)^\kappa$. For

example, for the plaintext space $\mathbb{Z}_{2^{20}}$ and $\kappa = 4$, the FPR is $\left(\frac{150}{2^{15}}\right)^4 \approx 2^{-31}$.

Ciphertext Compression

Ghostshell requires that a set of encrypted biometrics be sent to the server each time the user attempts the authentication process. Biometrics are usually represented by a lengthy sequence of bits (e.g., 2048 bits for standard iris codes [75]). Thus, encrypting biometrics in a bit-by-bit manner leads to a long transmission delay. Two methods for avoiding long delays are considered: packing numerous plaintext bits into a ciphertext and compressing the ciphertext in a cryptographic manner. The former technique was described in Section 5.4.3, so we only discuss the cryptographic compression of ciphertexts in the following.

The sizes of ciphertexts are designed to be very large to avoid lattice attacks, but they can be a big burden during communication. Coron et al. [30] observed that the size of ciphertexts could be reduced in their integer-based scheme by introducing a pseudorandom number generator (PRG) in their encryption function. They also described an extension of Brakerski and Vaikuntanathan's scheme [13, 14].

We implement their idea in the BGV-type scheme. We recall that for a plaintext x, the BGV ciphertext defined on a polynomial ring $\mathbb{A}_q := \mathbb{Z}_q[X]/\langle \Phi_m(X) \rangle$ has the form of $(a(X), \langle a(X), s(X) \rangle + x + 2e(X))$, where $a(X)$ is a random polynomial, $s(X)$ is a secret key, $e(X)$ is a small error, and all of the components are in $\mathbb{A}_q$. Now, let F be a q-bit PRG with a public random seed ω as its input. We only keep ω rather than $a(X)$, where $H(\omega \parallel i)$ corresponds to each coefficient of X^i for all $i \in [\varphi(m) - 1]$ and a random oracle $H : \{0,1\}^* \to \mathbb{Z}_q$. Thus, the resulting ciphertext size is $\lceil \log q \rceil \varphi(m)$ whereas the original ciphertext size is $2\lceil \log q \rceil \varphi(m)$.

A side effect of ciphertext compression is that we cannot perform homomorphic evaluations based on compressed ciphertexts because the PRG F is not homomorphic. Thus, homomorphically evaluating the ciphertexts

requires the recovery of the original ciphertext.

5.5 Implementation

To validate the impact of our optimization methods and to evaluate the practical utility of our biometric authentication system, we implemented a prototype with the following settings.

Test environment. A prototype of Ghostshell was implemented in C using the NTL library [70] in GMP. In addition, our implementation utilized the `HElib` library for the BGV cryptosystem. Our code was compiled using g++ 4.9.2 on Ubuntu 14.04.2 LTS. We performed all of the timing experiments on a server with 56 2.60 GHz Intel Xenon E5-2697 processors and 264 GB RAM. Our implementation included a symmetric cryptosystem and random oracles. We used AES-CBC and SHA1 for symmetric encryption and hash functions to instantiate the random oracles. To ensure public confidence, we utilized the AES and SHA1 modules provided by OpenSSL.

Parameter selection. To instantiate the BGV scheme via the `HElib` library, we first determined the basic parameters, such as the security parameter λ, the multiplicative depth L, and the plaintext space $\mathbb{Z}_M$. In our implementation, we set $\lambda = 80, L = 5$, and $M = 2^{15}$. As stated in Section 5.4.3, we used the degree of the cyclotomic polynomial $m = 8191$ as the number of plaintext slots $N = 630$.

Care is required when selecting an extension degree t for the plaintext space $\mathbb{Z}_M$. In general, it is preferable to use a large plaintext space, such as $\mathbb{Z}_{2^{50}}$, but this causes many performance issues, especially in SHE settings. For example, if $t \geq 20$, then one homomorphic multiplication consumes two or more levels. Furthermore, performing automorphism operations is no longer free in the noise estimate, so we selected $t = 15$.

If the size of the plaintext space $\mathbb{Z}_M$ becomes small, then the MAC key (r_0, r_1) should be small in length. In our case, $\lceil \log r_0 \rceil = 7$. In this case, the server might have a problem ensuring the integrity of the decryption d

since the FPR on user side would be relatively high. To address this problem, we required that the server send a set of s tags $(\bar{\tau}_0, \ldots, \bar{\tau}_{s-1})$ for $\bar{d}$, where $\bar{\tau}_i = r_{i0} \cdot \bar{d} + r_{i1}$ for uniformly random values r_{i0}, r_{i1} of small size and for each $i \in [s]$. By Lemma 1, it is sufficient to choose s such that $\ell \leq s \cdot \lceil \log r_{i0} \rceil$.

Biometric specification. In our implementation, we used irises as the input biometric. There is no universally best selection among the various biometric modalities, but the iris has been shown to be a superior biometric because of the relatively high accuracy that can be obtained (e.g., see [59,76]).

We represented each iris code as a binary string of 300 bytes. Any wavelet can be adjusted to the output length of the bit sequence, so the participants could choose a suitable iris bit size n depending on the server's environment. In contrast to the iris matching algorithms found in commercial software, our matching procedure is performed only once. The matching process must compensate for misalignment errors due to small rotations, but the overall performance ultimately depends on the computation of the HD, which can be executed in parallel.

5.5.1 Micro-experiments

Test datasets. Various public iris databases are available for use in research, such as CASIA-Iris [63]. CASIA-IrisV4 comprises six subsets: CASIA-Iris-Interval, CASIA-Iris-Lamp, CASIA-Iris-Twins, CASIA-Iris-Distance, CASIA-Iris-Thousand, and CASIA-Iris-Syn. We used CASIA-Iris-Interval, which comprises 2639 iris images from 249 subjects. We employed a public MATLAB code [61] to extract a binary iris template, where the original code used an image of the human eye as the input and produced a binary template with 9600 bits as the output. We modified the code slightly to extract iris templates with 2400 bits.

Basic operations. As shown in Section 5.4.1, the simplicity of Ghostshell is attributable to our exclusive use of SHE. Thus, the performance of Ghostshell is highly dependent on that of SHE. Therefore, our optimization processes

Table 5.4: Performance using 630-bit biometrics

Operations	CPU Time (msec)
Encryption	16.16
Decryption	163.72
Addition	0.05
Multiplication	14.32
Multiplication by constant	0.196

use SIMD and automorphisms extensively to reduce the number of basic operations, such as homomorphic addition and multiplication. In Table 5.4, we show the running times required to compute a ciphertext, decipher the ciphertext, and add and multiply two ciphertexts. Based on this information, we predicted that the total encryption time required for an iris code was four times 16.16 milliseconds, i.e., 64.6 milliseconds, which was confirmed by our experiments. We determined the same results for the other operations with each value of N. Thus, we predict that the total encryption time for an iris code is about four times the encryption time for each N, which was also confirmed by our experiments. We obtained the same results for the other operations and parameters.

HD CIRCUIT EVALUATION. The most demanding computational operation is the evaluation of the HD circuit. We measured the run-time required to compute the HD using the circuit optimized as described in Section 5.4.3.

According to our parameter selection (§**??**), a SHE ciphertext carries N plaintext bits at a time. Hence, we only need to retain $\lceil 2400/N \rceil$ ciphertexts for a 2400-bit iris code. For example, if $N = 630$, then only four ciphertexts are used to represent a 2400-bit iris code. For $N = 630$, the evaluation requires four multiplications and additions, with at most 10 additions and automorphisms. In our experiments, the average time of 0.37 seconds was required to compute a HD value between two encrypted iris codes.

We tested various values of N and corresponding values of m where the

number of ciphertexts retained was determined by N. The experimental results obtained for these selections of N and m are shown in Figure **??**. Each of the multiple ciphertexts corresponding to an iris code could be processed independently, so the computation was executed using thread-level parallelism. However, the results were substantially worse than our expectations based on the performance measures reported in Table 5.4. The main explanation for the computational delay is that the `HElib` library spends much of the processing time on the noise-control mechanism each time the homomorphic operations are performed.

Overall performance. In the setup phase, the user (1) runs the key generation algorithm with the parameters (§**??**) and (2) sends the public parameters required for running AES-CBC, SHA1, and our FKA scheme to the server. Thus, we only describe the case where $N = 630$ because this parameter had the best performance among the selected values of N. As a result, the size of the SHE ciphertexts was 327600 bits with q of 40 bits, the block length of AES was 128 bits, and the number of SHE ciphertexts per iris code was $\kappa = 4$.

After capturing an image and computing its feature vectors, the user (1) encrypts the iris feature code $\boldsymbol{x}^*$ into $\bar{\boldsymbol{x}}^*$ and (2) sends it to the server. This computation is performed only once and it will continue to be used until re-enrollment. The delay from encryption was approximately 65 milliseconds.

After receiving the user's authentication request with $\bar{\boldsymbol{x}}$, the server (1) computes the HD distance $\bar{d} = \mathsf{HD}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{x}}^*)$, (2) generates a tag $\bar{\tau}$ using a pair of randomly chosen keys, and (3) sends the pair of resulting values.

The HD is divided into four chunks, so $\bar{d}$ comprises four SHE ciphertexts, with a total size of 327600 bits and their computation required 0.37 seconds. Note that the server has to perform the homomorphic evaluations after decompression. Using the local parameter $s = 10$ and $\bar{d} := (\bar{d}[0] \parallel \cdots \parallel \bar{d}[3])$, the server randomly chooses $(r_{i0}[j], r_{i1}[j]) \in (\mathbb{Z}_7)^2$ for each $i \in [9], j \in [3]$. Next, the server computes $\bar{\tau}_i[j] = r_{i0}[j] \cdot \bar{d}[j] + r_{i1}[j]$. The total computational time was approximately 0.01 seconds and the total bandwidth requirement

was approximately 6.6 MB. The remaining computational time was significantly less than this computational time. Indeed, the total computation time on the user's side was approximately 5 milliseconds, and the server's computational time for the last verification was about 0.5 milliseconds.

Finally, we observed that different values of N and m had no effects on the performance in the verification step. When the degree of the cyclotomic polynomial m was higher, more computations had to be performed on the ciphertexts, but the performance did not differ with different choices of (N, m) because the verification step occurs after decryption.

5.6 Reusable Fuzzy Extractor for the Hamming Distance

Until now, biometric information is stored in the ciphertext form. However, my concern is the secret key for homomorphic encryption is also stored in the same data storage. Thus, if a secret key is leaked, then every biometric information is also leaked, which implies that security of biometric information also depends on the secret key of homomorphic encryption. In this sense, a technique for generating the same secret key from similar input is necessary and fuzzy extractor can achieve this goal.

Fuzzy extractors [36] extracts a uniformly random R from an input and it is error-tolerant; the same random R is generated whenever new input is not quite different from original input. In the recent Eurocrypt, Canetti *et al.* proposed a reusable fuzzy extractor [20] that is secure even though original input have been used multiple times. Therefore, if we take a biometric data as an input, a person can register many unrelated service providers with his biometric.

5.6.1 Insecurity of Previous Reusable Fuzzy Extractor

The drawback of Canetti's reusable fuzzy extractor we found out is the too-high false accept ratio(FAR), that is, the random number we want to keep secret is easily obtained by an adversary even though he/she do not have a close query to the legitimate input. Here is a theorem explaining the drawback.

Theorem 5.6.1. *Let $\mathcal{M} = \{0,1\}^n$ be the input space of Canetti's reusable fuzzy extractor with parameters n, ℓ, γ, δ, k, t as in Parameter Setting. For an input $W = w_1 \dots w_n$, let $(r, p) \leftarrow \mathtt{Gen}(W)$. If an adversary have a query input $W' = w'_1 \dots w'_n$ with $\mathsf{HD}(W, W') = d > t$, the FAR is at least $1/2$ under the assumption $\delta \leq 1/4$ and $\left(1 - \left(1 - \frac{t}{n}\right)^k\right)^\ell \approx \delta/2$.*

Proof. Using the information about p, we can get v'_i for $1 \leq i \leq \ell$. Then, since the probability that $v_i \neq v'_i$ for each i is $1 - (1 - \frac{d}{n})^k$, the probability there exists i such that $v_i = v'_i$ is

$$1 - \left(1 - \left(1 - \frac{d}{n}\right)^k\right)^\ell,$$

which has a maximum at $d = t + 1$. Thus the FAR δ_2 is

$$\delta_2 = 1 - \left(1 - \left(1 - \frac{t+1}{n}\right)^k\right)^\ell.$$

From the approximation $e^x \approx 1 + x$, we can approximate it as

$$\delta_2 \approx 1 - \exp(-\ell e^{-\frac{(t+1)k}{n}}) \approx 1 - \left(\frac{\delta}{2}\right)^{e^{-\frac{k}{n}}}.$$

Note that we may assume $k \leq n$ without loss of generality, which leads to the following inequality

$$1 - \frac{\delta}{2} \geq 1 - \left(\frac{\delta}{2}\right)^{e^{-\frac{k}{n}}} (\approx \delta_2) \geq 1 - \left(\frac{\delta}{2}\right)^{e^{-1}} \geq 1 - \left(\frac{1}{8}\right)^{\frac{1}{3}} = \frac{1}{2},$$

meaning the FAR is at least $1/2$ as we mentioned above. □

Theorem 5.6.2. *Let $\mathcal{M} = \{0,1\}^n$ be the input space of Canetti's reusable fuzzy extractor with parameters n, ℓ, γ, $\delta = 2^{-\lambda}$, k, t as in Parameter Setting. For an input $W = w_1 \dots w_n$, let $(r,p) \leftarrow \mathsf{Gen}(W)$. If an adversary have a query input $W' = w'_1 \dots w'_n$ with $\mathsf{dis}(W,W') = d > t$, the probability that $\mathsf{rep}(W')$ recovers r is at least $1 - 2^{-(\lambda+1)e^{-(d-t)}}$ under the assumption $\left(1 - \left(1 - \frac{t}{n}\right)^k\right)^\ell \approx \delta/2$.*

Proof. Using the information about p, we can get v'_i for $1 \le i \le \ell$. Then, since the probability that $v_i \neq v'_i$ for each i is $1 - (1 - \frac{d}{n})^k$, the probability there exists i such that $v_i = v'_i$ is

$$1 - \left(1 - \left(1 - \frac{d}{n}\right)^k\right)^\ell,$$

which has a maximum at $d = t + 1$. Thus the probability δ_2 that $\mathsf{rep}(W')$ recovers r is

$$\delta_2 = 1 - \left(1 - \left(1 - \frac{d}{n}\right)^k\right)^\ell.$$

From the approximation $e^x \approx 1 + x$, we can approximate it as

$$\delta_2 \approx 1 - \exp(-\ell e^{-\frac{dk}{n}}) \approx 1 - \left(\frac{\delta}{2}\right)^{e^{-\frac{(d-t)k}{n}}}.$$

Note that we may assume $k \le n$ without loss of generality, which leads to the following inequality

$$1 - \frac{\delta}{2} \ge 1 - \left(\frac{\delta}{2}\right)^{e^{-\frac{(d-t)k}{n}}} (\approx \delta_2) \ge 1 - \left(\frac{\delta}{2}\right)^{e^{-(d-t)}} \ge 1 - 2^{-(\lambda+1)e^{-(d-t)}}.$$

□

5.6.2 Revising Reusable Fuzzy Extractor

In Section 5.6.1, we find out the quality of Canetti's reusable fuzzy extractors is low, that is, an adversary who does not have legitimate input can also

obtain the secret random number with probability more than 1/2. This section addresses the main idea of reducing false accept rate and enhancing the quality of previous reusable fuzzy extractors. Moreover, we give an analysis of our proposal.

5.6.3 Revising Idea

As you can see in Algorithm, a random secret number is revealed when one random subset is equal to the original random subset. Moreover, the message space is $\mathbb{Z}_2$ and thus the probability of the coincidence is not low as expected, which implies an adversary could get the secret random number with more than half probability.

To overcome this shortcoming, we employ Shamir's secret sharing [69] which distributes a secret to participants, each of whom is allocated a share of the secret. The secret can be recovered only if a sufficient number of shares are combined together and individual shares are of no use on their own. In our case, a secret random number is distributed to each random subset which plays a participant role. Then, the secret random number can be reconstructed only when some random subsets coincide and due to secret sharing, we can expect reusable fuzzy extractors with combining Shamir secret sharing have a better quality than before. By the property of biometrics, templates which is made from extracting biometric is almost the same, but there also exists a slightly difference. Even though iris templates which can be expressed into a bit string are extracted from the same person, the hamming distance of two iris templates is not zero. Thus, two random subset which comes from two iris template, respectively, may not be the same even if two random subsets are indicated the same position. In this sense, we apply Shamir's (n, τ)-threshold scheme instead of the secret sharing where $\tau < n$. It makes our construction is still error-tolerant.

5.6.4 Our Construction

In this section, we give our concrete construction of reusable fuzzy extractors which combine Shamir's secret sharing. Since our construction is based on Canetti's scheme and digital lockers, the form is almost the same except distributing a secret. Moreover, parameter setting for $\mathcal{M}, n, \ell$ is also the same as before, but additional parameters p and τ are required in our construction. Concrete parameter setting for p and τ will be dealt in the next section. Our proposed `Gen` algorithm can be seen in Algorithm 5.5.

Revising `Gen`

Input: $W = w_1 \dots w_n$

1. Sample $r \xleftarrow{\$} \mathbb{Z}_p$ and $a_j \xleftarrow{\$} \mathbb{Z}_p$ for $1 \leq j \leq \tau - 1$
2. Set $a(x) \leftarrow r + \sum a_j x^j \bmod p$
3. For $i = 1, \dots, \ell$
   - (i) Sample $x_i \xleftarrow{\$} \mathbb{Z}_p$ and set $y_i \leftarrow a(x_i)$
   - (ii) choose uniformly $j_{i,m} \xleftarrow{\$} \{1, \dots, n\}$ for each $1 \leq m \leq k$
   - (iii) $v_i \leftarrow w_{j_{i,1}} \dots w_{j_{i,k}}$
   - (iv) $c_i \leftarrow \mathtt{lock}(v_i, y_i)$
   - (v) $p_i \leftarrow c_i, (j_{i,1}, \dots, j_{i,k})$
4. Output (r, p) where $p = p_1 \dots p_\ell$

Algorithm 5.5: `Gen` of Revising Reusable Fuzzy Extractors

As you can see in Algorithm 5.5, our construction employs Shamir's (ℓ, τ)-threshold scheme, that is to say, among ℓ subsets, one who can generate at least τ the same subsets can only recover the secret. Line 3.(i) of Algorithm 5.5 implies distributing a secret y_i to each subset and in line 3.(iii) of Algorithm 5.5, an allocated secret y_i is encrypted via digital lockers with a random subset v_i as a secret key.

We give reproducing algorithm `Rep` which is modified for Shamir's threshold scheme. Our construction can be seen in Algorithm 5.6.

Revising `Rep`

Input: $W' = w'_1 \dots w'_n, p = p_1 \dots p_\ell$

1. For $i = 1, \dots, \ell$
   - (i) Parse p_i as $c_i, (j_{i,1}, \dots, j_{i,k})$
   - (ii) $v'_i \leftarrow w'_{j_{i,1}} \dots w'_{j_{i,k}}$
   - (iii) $y_i \leftarrow \mathtt{unlock}(v'_i, c)$
     If $y_i \neq \perp$, then output y_i.
2. Recover r from $\{(x_i, y_i)\}$
3. Output r

Algorithm 5.6: `Rep` of Revising Reusable Fuzzy Extractors

Since $a(x)$ has $\tau - 1$ unknown coefficients, anyone who knows more than τ input and corresponding evaulation value can recover all coefficients of $a(x)$ by polynomial reconstruction and then one can recover r. For this reason, if one can successfully unlock more than τ times, then one can also find $a(x)$ and then r. (See line 1.(iii) and line 2 of Algorithm 5.6.)

5.6.5 Analyisis

In this subsection, we figure out some conditions to get desired FRR and FAR. Since the highest probability that the secret random number r is not recovered by a legitimate user occurs when the distance $\mathsf{HD}(W, W')$ is just the threshold t, let us assume $\mathsf{HD}(W, W') = t$ to consider the FRR and X be the cardinality of a set $\{1 \leq i \leq \ell : v_i = v'_i\}$ where v'_i comes from W' and p in the `Rep` phase. Then X is a bionomially distributed random variable, more precisely, we can denote $X \sim B(\ell, p_1)$ where $p_1 = \left(1 - \frac{t}{n}\right)^k$. Finally, we have an upper bound for the FRR as following:

$$\mathrm{FRR} = \Pr(X \leq \tau) \leq \exp\left(-2 \cdot \frac{(\ell p_1 - \tau)^2}{\ell}\right),$$

or

$$\mathrm{FRR} = \Pr(X \leq \tau) \leq \exp\left(-\frac{1}{2p_1} \cdot \frac{(\ell p_1 - \tau)^2}{\ell}\right),$$

where the first inequality comes from the Hoeffding's inequality and the second comes from the Chernoff's inequality.

To consider the FAR, assume $\mathsf{HD}(W, W') = t + 1$ and let X be the same as the above, that is, $X = |\{1 \leq i \leq \ell : v_i = v'_i\}|$. In this time, X is binomially distributed from $B(\ell, p_2)$ where $p_2 = \left(1 - \frac{t+1}{n}\right)^k$. Moreover, if we define $Y = \ell - X$, Y is a random variable for the number of i such that $v_i \neq v'_i$ and it is also binomially distributed from $B(\ell, 1 - p_2)$. Thus we get an upper bound for the FAR as following:

$$\begin{aligned}
\mathrm{FAR} = \Pr(X > \tau) = \Pr(\ell - Y > \tau) &= \Pr(Y \leq \ell - \tau - 1) \\
&\leq \exp\left(-2 \cdot \frac{(\ell(1 - p_2) - (\ell - \tau - 1))^2}{\ell}\right) \\
&= \exp\left(-2 \cdot \frac{(\tau + 1 - \ell p_2)^2}{\ell}\right)
\end{aligned}$$

by the Hoeffding's inequality, or

$$\mathrm{FAR} \leq \exp\left(-\frac{1}{2p_2} \cdot \frac{(\tau + 1 - \ell p_2)^2}{\ell}\right)$$

by the Chernoff's inequality.

Thus, if we choose parameters such that these upper bounds of FRR and FAR are sufficiently small simultaneously, then we can obtain a reusable fuzzy extractor with good quality. We are expecting we can take such parameters soon although having not found these yet.

Chapter 6

Conclusion

Applying homomorphic encryption to the database is starting point of this thesis. Database stores many information and if database are gathered, one can find out more information than the stored data. In order to protect, data should be encrypted when it stores and then anyone who can access database cannot learn any information from the encrypted data. However, for databases usefulness, database should provide database queries and thus homomorphic encryption is necessary to execute requested query being encrypted.

At first, I suggest an algorithm for homomorphic integer division. Although homomorphic encryption enables to add and multiply two ciphertexts, a division cannot be executed and a division can be widely used in the database queries. To solve this problem, I employ continued fraction to represent a rational number and restoring division algorithm to compute a quotient of two integers. As a result, we can divide two ciphertexts being encrypted.

Second, I propose an algorithm for homomorphic comparison over the real numbers. Similarly, I employ continued fraction to represent a real number to a set of integers and fortunately, comparing two continued fractions is also easy as much as a decimal representation. Thus, I translate this algorithm to the ciphertext domain and I can present concrete algorithms. Furthermore,

it can be utilized to the database sorting and private database queries such as retrieval queries and aggregate queries.

Finally, I propose an algorithm for integrity of homomorphic evaluations. When a data consumer is interested in the result, he will need a proof of correct decryption and I can solve this problem with one-time message authentication code (MAC). My proposal tag function is optimized because it only uses an addition and a multiplication. In addition, I suggest more secure algorithm using message encoding and discrete logarithm in order to prevent a data consumer from having a decryption oracle and it can be applied to the biometric authentication protocol.

I have implemented such algorithms using `HElib` and my code is uploaded in `http://github.com/heewon-chung/`. I would be delighted if my thesis helps a little bit in the development of homomorphic encryption and database security.

Bibliography

[1] FIDO alliance. https://fidoalliance.org.

[2] http://www.reuters.com/article/2015/09/23.

[3] R. Agrawal, D. Asonov, and R. Srikant. Enabling sovereign information sharing using web services. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 873–877, 2004.

[4] ARM. Building a secure system using TrustZone technology. In *http://www.arm.com*, 2009.

[5] M. Blanton and M. Aliasgari. Secure outsourced computation of iris matching. *Journal of Computer Security*, 2012.

[6] M. Blanton and P. Gasti. Secure and efficient protocols for iris and fingerprint identification. In *ESORICS*, pages 190–209, 2011.

[7] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private database queries using somewhat homomorphic encryption. In *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, pages 102–118, 2013.

[8] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005.

[9] D. Boneh, K. Lewi, M. Raykova, A. Sahai, M. Zhandry, and J. Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 563–594, 2015.

[10] J. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014.

[11] J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In M. Stam, editor, *IMA International Conference on Cryptography and Coding (IMACC) 2013*, volume 8308 of *LNCS*, pages 45–64. Springer, 2013.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[13] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[14] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology–Crypto*, pages 505–524, 2011.

[15] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner. GSHADE: faster privacy-preserving distance computation and biometric identification. In *IH & MMSec*, pages 187–198, 2014.

[16] J. Bringer, H. Chabanne, and A. Patey. Practical identification with encrypted biometric data using oblivious RAM. In *ICB*, pages 1–8, 2013.

[17] J. Bringer, H. Chabanne, and A. Patey. SHADE: Secure HAmming DistancE computation from oblivious transfer. In *FC*, pages 164–176, 2013.

[18] J. Bringer, M. Favre, H. Chabanne, and A. Patey. Faster secure computation for biometric identification using filtering. In *ICB*, pages 257–264, 2012.

[19] R. Canetti and R. R. Dakdouk. Obfuscating point functions with multibit output. In *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*, pages 489–508, 2008.

[20] R. Canetti, B. Fuller, O. Paneth, L. Reyzin, and A. D. Smith. Reusable fuzzy extractors for low-entropy distributions. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 117–146, 2016.

[21] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In *Financial Cryptography and Data Security*, pages 35–50, 2010.

[22] G. S. Çetin, Y. Doröz, B. Sunar, and E. Savas. Depth optimized efficient homomorphic sorting. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 61–80, 2015.

[23] A. Chatterjee, M. Kaushal, and I. Sengupta. Accelerating sorting of fully homomorphic encrypted data. In *Progress in Cryptology - INDOCRYPT 2013 - 14th International Conference on Cryptology in India, Mumbai, India, December 7-10, 2013. Proceedings*, pages 262–273, 2013.

[24] N. Chenette, K. Lewi, S. A. Weis, and D. J. Wu. Practical order-revealing encryption with limited leakage. In *Fast Software Encryption*

- *23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, pages 474–493, 2016.

[25] J. H. Cheon, M. Kim, and M. Kim. Search-and-compute on encrypted data. In *WAHC*, LNCS 8976, pages 1–18, 2015.

[26] J. H. Cheon, M. Kim, and M. Kim. Optimized search-and-compute circuits and their application to query evaluation on encrypted data. *IEEE Trans. Information Forensics and Security*, 11(1):188–199, 2016.

[27] H. Chung. `http://github.com/heewon-chung/cfhe`.

[28] H. Chung and M. Kim. Encoding rational numbers for fhe-based applications. *IACR Cryptology ePrint Archive*, 2016:344, 2016.

[29] J. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *Advances in Cryptology - CRYPTO 2011*, volume 6841 of *LNCS*, pages 487–504. Springer, 2011.

[30] J. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology–Eurocrypt*, pages 446–464, 2012.

[31] A. Costache, N. Smart, V. Vivek, and A. Waller. Fixed point arithmetic in SHE scheme. In *IACR Cryptology ePrint Archive*, volume 2016, page 250, 2016.

[32] G. Couteau. Efficient secure comparison protocols. *IACR Cryptology ePrint Archive*, 2016:544, 2016.

[33] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *ACISP*, pages 416–430, 2007.

[34] V. Dimitrov, L. Kerik, T. Krips, J. Randmets, and J. Willemson. Alternative implementations of secure real numbers. In *ACM CCS*, pages 553–564, 2016.

[35] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[36] Y. Dodis, L. Reyzin, and A. D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques*, pages 523–540, 2004.

[37] L. Ducas and D. Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 617–640. Springer, 2015.

[38] F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1155–1166, 2016.

[39] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies, 9th International Symposium, PETS 2009. Proceedings*, pages 235–253, 2009.

[40] M. Franz, B. Deiseroth, K. Hamacher, S. Jha, S. Katzenbeisser, and H. Schröder. Secure computations on non-integer values. In *WIFS*, pages 1–6, 2010.

[41] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[42] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[43] O. Goldreich. *The foundations of cryptography: Volume 2–Basic Applications*. Cambridge University Press, 2004.

[44] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[45] R. Gosper. Continued fraction arithmetic. In *HAKMEM Item 101B, MIT Artificial Intelligence Memo 239*, 1972.

[46] T. Graepel, K. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *ICISC*, volume 7839, pages 1–21, 2013.

[47] S. Halevi and V. Shoup. `HElib`-An implementation of homomorphic encryption.

[48] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, 1979.

[49] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[50] IEEE Std 754-2008. IEEE standard for floating-point arithmetic. Standard, Microprocessor Standards Committee of the IEEE Computer Society, 2008.

[51] A. Jarrous and B. Pinkas. Secure Hamming distance based computation and its applications. In *ACNS*, pages 107–124, 2009.

[52] A. Jäschke and F. Armknecht. Accelerating homomorphic computations on rational numbers. In *ACNS*, pages 405–423, 2016.

[53] C. Karabat, M. Kiraz, H. Erdogan, and E. Savas. THRIVE: threshold homomorphic encryption based secure and privacy preserving biometric verification system. *EURASIP J. Adv. Sig. Proc.*, 2015(71), 2015.

[54] F. Kerschbaum, M. Atallah, D. M'Raïhi, and J. Rice. Private fingerprint verification without local storage. In *ICBA*, pages 387–394, 2004.

[55] N. Kingsbury and P. Rayner. Digital filtering using logarithmic arithmetic. *Electronic Letters*, 7(2):56–58, 1971.

[56] R. Kulkarni and A. M. Namboodiri. Secure hamming distance based biometric authentication. In *International Conference on Biometrics, ICB 2013*, pages 1–6, 2013.

[57] K. Lewi and D. J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1167–1178, 2016.

[58] Y. Liu, Y. Chiang, T. Hsu, C. Liau, and D. Wang. Floating point arithmetic protocols for constructing secure data analysis application. In *KES*, pages 152–161, 2013.

[59] T. Mansfield. Biometric authentication in the real world. http://www.npl.co.uk/biometrics.

[60] C. Marforio, N. Karapanos, C. Soriente, K. Kostiainen, and S. Capkun. Secure enrollment and practical migration for mobile trusted execution environments. In *SPSM*, pages 93–98, 2013.

[61] L. Masek and P. Kovesi. MATLAB source code for a biometric identification system based on iris patterns. *The School of Computer Science and Software Engineering, The University of Western Australia. 2003.*, 2003.

[62] NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. In *FIPS 202*, 2015.

[63] C. A. of Science-Institute of Automation. CASIA iris database. http://biometrics.idealtest.org.

[64] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[65] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. SCiFI: A system for secure face identification. In *IEEE Symposium on Security and Privacy*, pages 239–254, 2010.

[66] P. Pullonen and S. Siim. Combining secret sharing and garbled circuits for efficient private IEEE 754 floating-point computations. In *Financial Cryptography and Data Security - FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*, pages 172–183, 2015.

[67] N. Ratha, J. Connell, and R. Bolle. Enhancing security and privacy in biometrics-based authentication systems. *IBM Systems Journal*, 40(3):614–634, 2001.

[68] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[69] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[70] V. Shoup. `NTL`-A library for doing number theory. 2009.

[71] G. Simmons. Authentication theory/coding theory. In *Advances in Cryptology–Crypto*, pages 411–431, 1984.

[72] N. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Cryptology ePrint Archive*, 133, 2011.

[73] N. Smart and F. Vercauteren. Fully homomorphic SIMD operations. *Designs, Codes and Cryptography*, 71(1):57–81, 2014.

[74] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.

[75] E. Tabassi, P. Grother, and W. Salamon. *IREX II-IQCE Iris quality calibration and evaluation performance of iris image quality assessment algorithms*. NIST Interagency Report 7820. 2011.

[76] S. Thavalengal, P. Bigioi, and P. Corcoran. Iris authentication in handheld devices - considerations for constraint-free acquisition. *IEEE Trans. Consumer Electronics*, 61(2):245–253, 2015.

[77] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–Eurocrypt*, pages 24–43, 2010.

국문초록

동형 암호는 복호화를 수행하지 않더라도 암호문 간의 연산을 가능하게 해주기 때문에 최근에 주목을 받고있는 암호 시스템이다. 이러한 이유로 여러 분야에서 동형암호를 활용하기 위해 많은 연구들이 이뤄지고 있는데, 본 학위 논문에서는 데이터베이스에 동형 암호를 접목시키는 것에 초점을 두고 연구를 하였다.

나눗셈은 우리 실생활에서 많이 쓰이는 연산이기 때문에 상당히 중요한 연산 중 하나이지만, 아직까지 동형 암호에서 나눗셈에 대한 연산을 지원해주고 있지 않다. 본 학위 논문에서는 이를 해결하기 위해 연분수를 도입하여 동형 암호를 활용한 정수 나눗셈에 대해서 연구를 하였다.

한편, 현재까지 알려진 동형암호들은 정수만을 평문으로 할 수 있기 때문에 실수로 이뤄져있는 데이터베이스에 대해서 암호화를 하기 위해서는 어려움이 많았다. 하지만 마찬가지로 연분수 개념을 적용하여 실수를 작은 정수들로 표시하고, 각 정수들을 암호화하는 방법을 통해 실수를 암호화하였다. 게다가 연분수로 표현된 암호화된 실수에 대해서 동형 암호 기반 상등 그리고 비교 대소에 대한 알고리즘을 제시했다. 그리고 제안한 알고리즘을 기반으로 가장 많이 쓰이는 쿼리에 적용시킬 수 있었고, 그 결과 데이터베이스가 실수로 이뤄져있더라도 안전한 서비스를 제공할 수 있음을 확인했다.

마지막으로 데이터 소유주가 아닌 데이터베이스 관리자가 계산한 결과값을 알아야하는 경우 메세지 인증 코드를 이용하여 데이터 소유주가 제대로 복호화를 해서 관리자에게 전송을 했는지 확인하는 알고리즘에 대해서도 제안을 했다. 그리고 제안한 알고리즘을 바탕으로 생체인증을 기반으로 하는 안전한 인증 프로토콜을 설계하였다.

주요어휘: 동형 암호, 연분수, 실수연산, 데이터베이스 쿼리, 생체인증
학번: 2013-30900