Practical Fully Homomorphic Encryption without Noise Reduction

Dongxi Liu

CSIRO, Marsfield, NSW 2122, Australia
dongxi.liu@csiro.au

Abstract. We present a new fully homomorphic encryption (FHE) scheme that is efficient for practical applications. The main feature of our scheme is that noise reduction considered essential in current FHE schemes, such as boot strapping and modulus switching, is not needed in our scheme, because it allows arbitrarily large noises in its ciphertexts. A ciphertext in our scheme is a vector with its dimension specified as a security parameter of the encryption key. The dimension of ciphertexts does not change with homomorphic operations and all ciphertext elements are in a finite domain, so our scheme is compact. In addition, our scheme can directly encrypt big integers, rather than only bit messages.
We proved the hardness of recovering encryption keys from any number of ciphertexts with chosen plaintexts and then the semantic security of our scheme. The hardness of recovering keys from ciphertexts is based on the approximate greatest common divisors problem. We implemented a prototype of our scheme and evaluated its concrete performance extensively from the aspects of encryption, decryption, homomorphic operations, and bitwise operators over ciphertexts. The efficiency of our scheme is confirmed by the evaluation result.

Keywords: Fully Homomorphic Encryption, Implementation, Practical Efficiency

1 Introduction

Homomorphic encryption has the attractive property of permitting computation over encrypted data. An encryption scheme is fully homomorphic if the encrypted data can be homomorphically added and multiplied. In a theoretical breakthrough work, Gentry described the first fully homomorphic encryption (FHE) scheme [8]. Following Gentry's scheme, there are currently a number of FHE schemes with various improvements, such as weaker security assumptions, shorter public keys, or better asymptotic efficiency [1, 4, 7, 16, 9, 2, 3].

However, the current FHE schemes are still believed not suitable for practical applications due to their poor concrete performance [10, 7]. It has been realized that the poor performance is mainly caused by the noise reduction mechanisms in the current FHE schemes [2]. One noise reduction mechanism is bootstrapping, which is used in many FHE schemes [1, 4, 7, 16, 9]. As evaluated in [11], a bootstrapping operation might take about 5.5 minutes.

In the FHE scheme proposed by Brakerski, Gentry and Vaikuntanathan [2, 3], they described another noise reduction mechanism called modulus switching, which can achieve better asymptotic performance than bootstrapping. However, there is no implementation and concrete performance evaluation in [2, 3] and it is not clear whether their scheme is efficient enough for practical applications. In addition to the performance problem, noise reduction increases the complexity of current FHE schemes for understanding and implementing. In [13], Naehrig, Lauter and Vaikuntanathan only implemented the somewhat version of the FHE scheme proposed by Brakerski and Vaikuntanathan in [1].

Noise reduction is essential for current FHE schemes because their ciphertexts cannot be decrypted correctly if the noises in ciphertexts reach a certain limit. Since homomorphic operations (in particular homomorphic multiplications) can increase the noises in the resulting ciphertexts, noise reduction mechanisms must be applied together with homomorphic operations to reduce the noises accumulated in the resulting ciphertexts.

In this paper, we propose a symmetric FHE scheme that allows arbitrarily large noises in a ciphertext and the resulting ciphertexts generated through any number of homomorphic operations can always be decrypted correctly regardless of the amount of noises accumulated in them. Hence, our scheme does not need any noise reduction mechanism.

A ciphertext in our scheme is a vector, with the dimension specified as a parameter of the encryption key. The ciphertexts can be homomorphically added or multiplied, and these homomorphic operations do not change their dimensions. In addition, the ciphertext elements are defined in a finite domain specified by a modulus. Hence, our scheme is compact. The features of no noise reduction and compactness make our scheme efficient and also make it simple to implement and use.

We proved the security of our scheme from two aspects. First, we proved that the secret components in an encryption key cannot be recovered from any number of ciphertexts with chosen plaintexts by the adversary. This proof is based on the hardness of the approximate greatest common divisors problem. Then, we proved the semantic security of our scheme; intuitively, our scheme is semantically secure because the noises in a ciphertext can be arbitrarily large and if the secret key components cannot be found from ciphertexts, the large noises cannot be removed by the adversary to distinguish ciphertexts.

A prototype of our scheme has been implemented in Java and extensively evaluated on a Dell XPS 13 laptop from multiple perspectives: the performance of encryption and decryption, the performance of homomorphic operations, and the performance of bitwise operators over ciphertexts. A keyword-based 1-out-of-n oblivious transfer protocol is also implemented as an application of our scheme to evaluate its usability in practical applications.

In our experiments, the key is configured to have a space bigger than 2^{128} and to generate ciphertexts having six dimensions. Our evaluation shows that our scheme has better encryption and decryption performance than AES 128 provided in the Java SunJCE package. For homomorphic operations, our scheme takes about 1 second to evaluate the high degree polynomial $\sum_{i=1}^{1000} x_i^i$, and 0.5 second to compare the bitwise encryption of two integers with 1000 bits. In the test of retrieving a value from a table with 1000 entries based on an encrypted keyword, our scheme takes about 2 or 3 seconds, depending on whether the keyword is represented in 16 bits or 32 bits. The efficiency of our scheme makes it suitable for many practical applications, such as statistics over encrypted data [13] and query over encrypted relational databases [15, 14].

The rest of this paper is organized as follows. We give an overview of our scheme in Section 2, followed by its construction in Section 3 and correctness proof in Section 4. We prove the security of our scheme in Section 5, with the implementation and evaluation result presented in Section 6. In Section 7, we conclude the paper.

2 An Overview of Our Scheme

Let q be a prime and $\mathbb{Z}_q$ be the set of integers modulo q. The modulus q is public in our scheme. Given the secret key $K(n)$, an integer $v \in \mathbb{Z}_q$ in our scheme is encrypted into the ciphertext $(c_1, ..., c_{n+1})$, where $c_i \in \mathbb{Z}_q$. The dimension of a ciphertext is determined by the parameter n in the key $K(n)$. The security parameter of our scheme includes n and q, which both determine the size of key space, and they are chosen by users who want to encrypt their data.

We denote the encryption and decryption operations of our scheme by the following notations.

$$Enc(K(n), v) = (c_1, ..., c_{n+1})$$
$$Dec(K(n), (c_1, ..., c_{n+1})) = v$$

For the key $K(n)$, our scheme provides a public evaluation key $PEK = \{pek_{ij} | 1 \leq i \leq n+1, 1 \leq j \leq n+1\}$, which is used when performing homomorphic multiplication. A PEK element pek_{ij} is obtained from the encryption $Enc(K(n), sek_{ij})$, where $sek_{ij} \in \mathbb{Z}_q$ is a secret evaluation key element, derived from $K(n)$. Hence, pek_{ij} is also a $n+1$ dimensional vector.

In the following, we suppose the ciphertexts $C = (c_1, ..., c_{n+1})$ and $C' = (c'_1, ..., c'_{n+1})$ are encrypted with the key $K(n)$ for discussing the homomorphic properties of our scheme.

2.1 Homomorphic Addition

The homomorphic addition of C and C' is defined as a vector addition, that is, $C \oplus C' = (c_1 + c'_1 \texttt{ mod } q, ..., c_{n+1} + c'_{n+1} \texttt{ mod } q)$. Our scheme supports additive homomorphism by ensuring the following condition.

$$Dec(K(n), C \oplus C') = Dec(K(n), C) + Dec(K(n), C') \texttt{ mod } q$$

Let $d \in \mathbb{Z}_q$ and $d \odot C = (d * c_1 \texttt{mod } q, ..., d * c_{n+1} \texttt{mod } q)$. According to the additively homomorphic property, we have $Dec(K(n), d \odot C) = d * Dec(K(n), C) \texttt{ mod } q$.

2.2 Homomorphic Multiplication

With the public evaluation key PEK, the homomorphic multiplication of C and C', denoted by $C \otimes_{PEK} C'$, is defined by the following expression.

$$((c_1 * c'_1) \odot pek_{11}) \oplus ((c_1 * c'_2) \odot pek_{12}) \oplus ... \oplus ((c_{n+1} * c'_{n+1}) \odot pek_{(n+1)(n+1)})$$

Our scheme is multiplicatively homomorphic by ensuring the following condition.

$$Dec(K(n), C \otimes_{PEK} C') = Dec(K(n), C) * Dec(K(n), C') \texttt{ mod } q$$

Thus, we can evaluate any degree of polynomials over ciphertexts by using homomorphic operations $\oplus$, $\odot$ and $\otimes_{PEK}$. Note that our scheme does not need any noise reduction schemes (e.g., boot strapping and modulus switching) to be used together with homomorphic operations. Moreover, our scheme is compact, since all homomorphic operations do not change the dimension of resulting ciphertexts and each ciphertext element is an integer in $\mathbb{Z}_q$.

3 Construction of Our Scheme

We define the structure of key $K(n)$, the encryption and decryption algorithms, and the secret and public evaluation keys in this section. All random integers are uniformly sampled from $\mathbb{Z}_q$.

3.1 Structure of $K(n)$

A key $K(n)$ in our scheme is a tuple of four secret components $(\Gamma, \Pi, \Theta, \Phi)$. For a key $K(n)$, we require $n \geq 3$, such that there exist integers $h \geq 2$ and $m \geq 1$, satisfying $n = h + m$. The component Γ is a list $[k_1, ..., k_n]$, where each k_i is a tuple of random integers in $\mathbb{Z}_q$.

$$k_i = \begin{cases} (a, s_{i1}, ..., s_{im}, t_i) & \text{if } 1 \leq i \leq h; \\ (s_{iu}, ..., s_{im}, t_i) & \text{if } h+1 \leq i \leq n-1 \text{ and } u = i - h + 1; \\ (t_i) & \text{if } i = n. \end{cases}$$

As shown by the above definition of k_i, different choices of h and m can lead to different structures of Γ. On the other hand, h and m can be derived from the structure of Γ, so they do not need to be explicitly kept in the key. For correctness, we require $a \neq 0$ and $t_i \neq 0$ for $1 \leq i \leq h$.

The component Π of key $K(n)$ is a random permutation of the set $\{1, ..., n+1\}$. Suppose $\Pi = \{d_1, ..., d_{n+1}\}$. Then, we have the notation $\Pi(i) = j$, such that $i = d_j$. The component Θ is a list of l random integers in $\mathbb{Z}_q$, and we require $l \leq n - 2$. The component Φ is a list consisting of $l + 1$ ciphertexts, which will be defined in the next section.

3.2 Encryption

We have two levels of encryption. The lower level encryption can only be used a limited number of times (at most $n-1$ times) for a key $K(n)$ for the security reason to be discussed below, while the upper level encryption can be used any number of times to encrypt values in $\mathbb{Z}_q$, including secret evaluation keys.

3.3 Lower Level Encryption

The lower level encryption is denoted by $Enc_l(K(n), v) = (c_1, ..., c_{n+1})$. The lower level encryption algorithm only uses the components Γ and Π in $K(n)$ to define each c_i, as shown below, where r_1, ...,r_h, rs_1,..., rs_m, rv_1,...,rv_{h-1} and rr are random integers uniformly sampled from $\mathbb{Z}_q$, and $S(i) = \Sigma_{j=1}^{m} s_{ij} * rs_j$.

$$c_{\Pi(i)} = \begin{cases} a * t_i * (v + \Sigma_{j=1}^{h-1} rv_j) + S(i) + t_i * (r_i - r_h) \bmod q & \text{if } i = 1; \\ a * t_i * (-rv_{i-1}) + S(i) + t_i * (r_i - r_{i-1}) \bmod q & \text{if } 2 \leq i \leq h; \\ rs_u + \Sigma_{j=u+1}^{m} s_{ij} * rs_j + t_i * rr \bmod q & \text{if } h+1 \leq i \leq n-1 \\ & \quad \text{and } u = i - h; \\ rs_m + t_i * rr \bmod q & \text{if } i = n; \\ rr \bmod q & \text{if } i = n+1. \end{cases}$$

In our scheme, the lower level encryption is only used to generate the component Φ in $K(n)$ (as a part of key generation). Suppose in $K(n)$, we have $\Theta = [\theta_1, ..., \theta_l]$, where $l \leq n-2$. Then, we generate $\Phi = [\phi_1, ..., \phi_{l+1}]$, with ϕ_i defined below.

$$\phi_i = \begin{cases} Enc_l(K(n), \theta_i) & \text{if } i \leq l; \\ Enc_l(K(n), 1) & \text{if } i = l+1. \end{cases}$$

Hence, the lower level encryption is used at most $n-1$ times. Note that ϕ_{l+1} is an encryption of the integer 1. Θ and Φ in $K(n)$ will be used in the upper level encryption.

3.4 Upper Level Encryption

We denote the upper level encryption by $Enc(K(n), v) = (c_1, ..., c_{n+1})$. That is, the encryption described in Section 2 is the upper level encryption. Let $\Theta = [\theta_1, ..., \theta_l]$ and $\Phi = [\phi_1, ..., \phi_{l+1}]$ in $K(n)$. Then, the upper level encryption is defined as

$$Enc(K(n), v) = (ru_1 \odot \phi_1) \oplus (ru_2 \odot \phi_2) \oplus ... \oplus (ru_{l+1} \odot \phi_{l+1})$$

where ru_1,...,and ru_l are random integers uniformly sampled from $\mathbb{Z}_q$ and $ru_{l+1} = v - \Sigma_{i=1}^{l} ru_i * \theta_i \bmod q$.

3.5 Decryption

Our decryption algorithm recovers a value v from a ciphertext $(c_1, ..., c_{n+1})$ with the key $K(n)$ by taking the following steps.

$$\begin{array}{lll}
(1)\ RR & = c_{\Pi(n+1)} \bmod q; & \\
(2)\ RS_m & = c_{\Pi(n)} - t_n * RR \bmod q; & \\
(3)\ RS_u & = c_{\Pi(i)} - t_i * RR - \Sigma_{j=u+1}^{m} s_{ij} * RS_j \bmod q, & \text{for } u \text{ from } m-1 \text{ to } 1 \\
 & & \text{and } i = u + h; \\
(4)\ F & = \Sigma_{i=1}^{h}((c_{\Pi(i)} - \Sigma_{j=1}^{m} s_{ij} * RS_j)/t_i) \bmod q; & \\
(5)\ v & = F/a \bmod q. &
\end{array}$$

The correctness condition $a \neq 0$ and $t_i \neq 0$ $(1 \leq i \leq h)$ for $K(n)$ ensures the validity of decryption steps (i.e., no division by zero). We prove the correctness of our scheme in the next section.

In the above definition, the decryption algorithm is described in five steps by using intermediate variables, such as RS_u and F. Actually, we can fuse these steps by replacing each intermediate variable (F, RS_i or RR) by its definition recursively until all intermediate variables are removed. After fusion, we get a linear form of the decryption algorithm.

$$v = dk_1 * c_{\Pi(1)} + ... + dk_{n+1} * c_{\Pi(n+1)} \bmod q$$

where $dk_1, ...,$ and dk_{n+1}, called linear decryption keys, are defined over Γ in $K(n)$. The linear decryption keys will be used to define the secret and public evaluation keys in the next section.

From the linear form of decryption, we can see if the lower level encryption is used n times or more for a key $K(n)$, then the linear decryption keys can be obtained by solving n equations under chosen-plaintext attacks. For this security reason, our scheme requires the lower level encryption be used at most $n-1$ times and only used for generating Φ. The upper level encryption works by randomly combining vectors in Φ (i.e., using the ciphertexts in Φ as a basis), so the upper level encryption does not generate independent vectors and can be securely used any number of times.

As an example, we fuse the decryption steps for the key $K(5)$, with $h = 3$ and $m = 2$, and get the following six linear decryption keys dk_i $(1 \leq i \leq 6)$, where $w = a * (t_1 * t_2 * t_3) \bmod q$.

$$\begin{aligned}
dk_1 &= t_2 * t_3 / w \bmod q \\
dk_2 &= t_1 * t_3 / w \bmod q \\
dk_3 &= t_1 * t_2 / w \bmod q \\
dk_4 &= -(t_2 * t_3 * s_{11} + t_1 * t_3 * s_{21} + t_1 * t_2 * s_{31}) / w \bmod q \\
&= -(dk_1 * s_{11} + dk_2 * s_{21} + dk_3 * s_{31}) \bmod q \\
dk_5 &= (t_2 * t_3 * s_{11} + t_1 * t_3 * s_{21} + t_1 * t_2 * s_{31}) * s_{42} / w \\
&\quad -(t_2 * t_3 * s_{12} + t_1 * t_3 * s_{22} + t_1 * t_2 * s_{32}) / w \bmod q \\
&= -dk_4 * s_{42} - (dk_1 * s_{12} + dk_2 * s_{22} + dk_3 * s_{32}) \bmod q \\
dk_6 &= (t_2 * t_3 * s_{11} + t_1 * t_3 * s_{21} + t_1 * t_2 * s_{31}) * (t_4 - s_{42} * t_5) / w \\
&\quad +(t_2 * t_3 * s_{12} + t_1 * t_3 * s_{22} + t_1 * t_2 * s_{32}) * t_5 / w \bmod q \\
&= -dk_4 * (t_4 - s_{42} * t_5) + (dk_1 * s_{12} + dk_2 * s_{22} + dk_3 * s_{32}) * t_5 \bmod q
\end{aligned}$$

To define secret evaluation keys in the next section, we need to permutate the keys $dk_1, ..., dk_{n+1}$ into $dk'_1, ..., dk'_{n+1}$, such that $dk'_j = dk_i$ if $\Pi(i) = j$ for $1 \le i \le n+1$ and $1 \le j \le n+1$. Since $dk'_j * c_j = dk_i * c_{\Pi(i)}$, the linear form of decryption can be written into the following new form, where Π is not needed any more.

$$v = dk'_1 * c_1 + ... + dk'_{n+1} * c_{n+1} \bmod q$$

3.6 Secret and Public Evaluation Keys

Recall that our scheme provides a public evaluation key $PEK = \{pek_{ij} | 1 \le i \le n+1, 1 \le j \le n+1\}$ for homomorphic multiplication. An element in PEK is an encryption of a secret evaluation key element sek_{ij}, that is, $pek_{ij} = Enc(K(n), sek_{ij})$.

Given $Dec(K(n), C) = v$ and $Dec(K(n), C') = v'$, our scheme ensures $Dec(K(n), C \otimes_{PEK} C') = v * v' \bmod q$. Let $C = (c_1, ..., c_{n+1})$ and $C' = (c'_1, ..., c'_{n+1})$. From the multiplicatively homomorphic property, we can determine the following condition that must be satisfied by sek_{ij}.

$$\begin{aligned}
&Dec(K(n), C \otimes_{PEK} C') \\
&= Dec(K(n), ((c_1 * c'_1) \odot pek_{11} \oplus ... \oplus (c_{n+1} * c'_{n+1}) \odot pek_{(n+1)(n+1)})) \\
&= \Sigma_{i=1}^{n+1} \Sigma_{j=1}^{n+1} (c_i * c'_j) * Dec(K(n), pek_{ij})) \bmod q \\
&= \Sigma_{i=1}^{n+1} \Sigma_{j=1}^{n+1} (c_i * c'_j) * sek_{ij} \bmod q \\
&= v * v' \bmod q
\end{aligned}$$

The secret evaluation key is derived from the linear form of our decryption algorithm. For $Dec(K(n), C) = v$ and $Dec(K(n), C') = v'$, we have their corresponding linear decryption forms.

$$\begin{aligned} v &= \Sigma_{i=1}^{n+1} dk'_i * c_i \bmod q \\ v' &= \Sigma_{j=1}^{n+1} dk'_j * c'_j \bmod q \end{aligned}$$

By multiplying v and v', we can get the following equations.

$$\begin{aligned} & v * v' \bmod q \\ &= (\Sigma_{i=1}^{n+1} dk'_i * c_i) * (\Sigma_{j=1}^{n+1} dk'_j * c'_j) \bmod q \\ &= \Sigma_{i=1}^{n+1} \Sigma_{j=1}^{n+1} (dk'_i * c_i * dk'_j * c'_j) \bmod q \\ &= \Sigma_{i=1}^{n+1} \Sigma_{j=1}^{n+1} (c_i * c'_j * dk'_i * dk'_j) \bmod q \end{aligned}$$

Thus, by defining $sek_{ij} = dk'_i * dk'_j \bmod q$, we have $\Sigma_{i=1}^{n+1} \Sigma_{j=1}^{n+1} (c_i * c'_j * sek_{ij}) = v * v' \bmod q$, meaning that this definition of sek_{ij} satisfies the required condition. Note that each sek_{ij} is also a value in $\mathbb{Z}_q$, so it can be encrypted as other plaintext values.

4 Correctness of Our Scheme

We prove the correctness of additive homomorphism for our lower level encryption algorithm, and then extend the proof to other cases.

4.1 Additive Homomorphism

Suppose N values v_b ($1 \le b \le N$) are encrypted into N ciphertexts $(c_1^b, ..., c_{n+1}^b)$, under the key $K(n)$, with the lower level encryption algorithm. For correctness proof we do not care the number of times the lower level encryption is used. In the following, we prove that $Dec(K(n), (\Sigma_{b=1}^N c_1^b \bmod q, ..., \Sigma_{b=1}^N c_{n+1}^b \bmod q)) = \Sigma_{b=1}^N v_b \bmod q$.

Let r_1^b, r_2^b,...,r_h^b, rs_1^b, ..., rs_m^b, rv_1^b,...,rv_{h-1}^b and rr^b be the random numbers used in the encryption of v_b. Then, we have the following definition for the sum of each ciphertext element. Let $S(i) = \Sigma_{j=1}^m s_{ij} * (\Sigma_{b=1}^N rs_j^b)$.

$$\Sigma_{b=1}^N c_{\Pi(i)}^b = \begin{cases} a * t_i * (\Sigma_{b=1}^N (v_b + \Sigma_{j=1}^{h-1} rv_j^b)) + S(i) + t_i * (\Sigma_{b=1}^N (r_1^b - r_h^b)) \bmod q, \\ \qquad \text{if } i = 1; \\ a * t_i * (\Sigma_{b=1}^N (-rv_{i-1}^b)) + S(i) + t_i * (\Sigma_{b=1}^N (r_i^b - r_{i-1}^b)) \bmod q, \\ \qquad \text{if } 2 \le i \le h; \\ \Sigma_{b=1}^N rs_u^b + \Sigma_{j=u+1}^m s_{ij} * (\Sigma_{b=1}^N rs_j^{(b)}) + t_i * (\Sigma_{b=1}^N rr^b) \bmod q, \\ \qquad \text{if } h+1 \le i \le n-1 \text{ and } u = i - h; \\ \Sigma_{b=1}^N rs_m^b + t_i * (\Sigma_{b=1}^N rr^b) \bmod q, \quad \text{if } i = n; \\ \Sigma_{b=1}^N rr^b \bmod q, \quad \text{if } i = n + 1. \end{cases}$$

Based on the above definition, the proof below checks each decryption step, eventually showing that the decryption result is $\Sigma_{b=1}^{N} v_b$.

- $RR = \Sigma_{b=1}^{N} c_{\Pi(n+1)}^{b} \bmod q \quad = \quad \Sigma_{b=1}^{N} rr^{b} \bmod q$
- $RS_m = \Sigma_{b=1}^{N} c_{\Pi(n)}^{b} - t_n * RR \bmod q$
  $= \Sigma_{b=1}^{N} rs_m^{b} + t_n * (\Sigma_{b=1}^{N} rr^{b}) - t_n * (\Sigma_{b=1}^{N} rr^{b}) \bmod q$
  $= \Sigma_{b=1}^{N} rs_m^{b} \bmod q$
- For u from $m-1$ to 1 and $i = u+h$, we can recursively have $RS_u = \Sigma_{b=1}^{N} rs_u^{b} \bmod q$.

  $RS_u = \Sigma_{b=1}^{N} c_{\Pi(i)}^{b} - t_i * RR - \Sigma_{j=u+1}^{m} s_{ij} * RS_j \bmod q$
  $= \Sigma_{b=1}^{N} rs_u^{b} + \Sigma_{j=u+1}^{m} s_{ij} * (\Sigma_{b=1}^{N} rs_j^{b}) + t_i * (\Sigma_{b=1}^{N} rr^{d}) - t_i * RR$
  $- \Sigma_{j=u+1}^{m} s_{ij} * RS_j \bmod q$
  $= \Sigma_{b=1}^{N} rs_u^{b} \bmod q$
- Let $F = F_1 + F_{2h}$, where F_1 and F_{2h} are defined below. Then we have $F = a * (\Sigma_{b=1}^{N} v_b) \bmod q$.

  $F_1 = (\Sigma_{b=1}^{N} c_{\Pi(1)}^{b} - \Sigma_{j=1}^{m} s_{1j} * RS_j)/t_1 \bmod q$
  $= (a * t_1 * (\Sigma_{b=1}^{N} (v_b + \Sigma_{j=1}^{h-1} rv_j^{b})) + S(1) + t_1 * (\Sigma_{b=1}^{N} (r_1^{b} - r_h^{b}))$
  $- \Sigma_{j=1}^{m} s_{1j} * RS_j)/t_1 \bmod q$
  $= (a * t_1 * (\Sigma_{b=1}^{N} (v_b + \Sigma_{j=1}^{h-1} rv_j^{b})) + t_1 * (\Sigma_{b=1}^{N} (r_1^{b} - r_h^{b})))/t_1 \bmod q$
  $= a * (\Sigma_{b=1}^{N} (v_b + \Sigma_{j=1}^{h-1} rv_j^{b})) + \Sigma_{b=1}^{N} (r_1^{b} - r_h^{b}) \bmod q$
  $= a * \Sigma_{b=1}^{N} v_b + a * \Sigma_{b=1}^{N} \Sigma_{j=2}^{h} rv_{j-1}^{b} + \Sigma_{b=1}^{N} (r_1^{b} - r_h^{b}) \bmod q$

  $F_{2h} = \Sigma_{i=2}^{h} (\Sigma_{b=1}^{N} c_{\Pi(i)}^{b} - \Sigma_{j=1}^{m} s_{ij} * RS_j)/t_i \bmod q$
  $= \Sigma_{i=2}^{h} (a * t_i * (\Sigma_{b=1}^{N} (-rv_{i-1}^{b})) + S(i) + t_i * (\Sigma_{b=1}^{N} (r_i^{b} - r_{i-1}^{b}))$
  $- \Sigma_{j=1}^{m} s_{ij} * RS_j)/t_i \bmod q$
  $= \Sigma_{i=2}^{h} (a * t_i * (\Sigma_{b=1}^{N} (-rv_{i-1}^{b})) + t_i * (\Sigma_{b=1}^{N} (r_i^{b} - r_{i-1}^{b})))/t_i \bmod q$
  $= \Sigma_{i=2}^{h} a * (\Sigma_{b=1}^{N} (-rv_{i-1}^{b})) + \Sigma_{i=2}^{h} (\Sigma_{b=1}^{N} (r_i^{b} - r_{i-1}^{b})) \bmod q$
  $= \Sigma_{i=2}^{h} a * (\Sigma_{b=1}^{N} (-rv_{i-1}^{b})) + \Sigma_{b=1}^{N} (r_h^{b} - r_1^{b}) \bmod q$
- At last, according to the last step of the decryption algorithm, the decryption result is $\Sigma_{b=1}^{N} v_b = F/a \bmod q$.

4.2 Other Cases

Suppose the value v is encrypted into $(c_1, ..., c_{n+1})$ under the key $K(n)$, with the random numbers r_1, r_2,...,r_h,rs_1,..., and rs_m, rv_1,..., rv_{h-1}, and rr. The proof of $Dec(K(n), (d * c_1 \bmod q, ..., d * c_{n+1} \bmod q)) = d * v \bmod q$ has the same structure as the above proof for additive homomorphism. To obtain the proof, we replace $\Sigma_{b=1}^{N} c_{\Pi(i)}^{b}$ in the above proof with $d * c_{\Pi(i)}$, $\Sigma_{b=1}^{N} v_b$ with $d * v$, $\Sigma_{b=1}^{N} rs_j^{b}$ with $d * rs_j$, etc. These proofs imply the correctness of the upper level encryption, since it is defined over homomorphic operations $\oplus$ and $\odot$.

From the proof of $\odot$ operation, by letting $d = 1$, we get the correctness proof of $Dec(K(n), (c_1, ..., c_{n+1})) = v$. The homomorphic multiplication $\otimes_{PEK}$

is defined over $\oplus$ and $\odot$. Hence, the above proofs can establish the correctness of multiplicative homomorphism, together with the correctness of secret evaluation keys, which is discussed in the previous section.

5 Security Analysis

We first prove that it hard to recover the secret components Γ and Φ in a key $K(n)$ from ciphertexts. Then based on the hardness of the key recovery problem, we prove the semantic security of our scheme. The hardness of our key search problem is based on the approximate greatest common divisors (AGCD) problem.

5.1 The AGCD Problem

This problem is proposed by Howgrave-Graham [12]. Given any number of the approximate multiples $a_i = p * q_i + r_i$ of p, where p, q_i and r_i are integers, the problem is to find the hidden common divisor p. Note that q_i and r_i change in each a_i. There are algorithms proposed in [5, 6] to recover p, but this problem is still believed to be hard and used by fully homomorphic encryption schemes, such as [16, 7]. In particular, if r_i can be as large as p, it is impossible to reconstruct p from any number of approximate multiples a_i [6].

5.2 Hardness of Recovering Φ and Γ

The component Φ in a key $K(n)$ is a list of secret vectors that are linearly combined to generate ciphertexts in the upper level encryption. We prove that it is hard to recover the secret vectors in Φ from any number of ciphertexts.

Theorem 1. *Given any number of ciphertexts from the upper level encryption with $K(n)$, it is hard to recover Φ in $K(n)$.*

Proof 1 *Let $\Phi = [\phi_1, ..., \phi_{l+1}]$, where $l \leq n-2$, and $\phi_i = (\phi_{i1}, ..., \phi_{i(n+1)})$. As shown in the upper level encryption, a ciphertext $(c_1, ..., c_{n+1})$ is defined as:*

$$\begin{aligned} c_1 &= \Sigma_{i=1}^{l+1} ru_i * \phi_{i1} \bmod q \\ &... \\ c_{n+1} &= \Sigma_{i=1}^{l+1} ru_i * \phi_{i(n+1)} \bmod q \end{aligned}$$

In the first ciphertext element c_1, ϕ_{i1} $(1 \leq i \leq l+1)$ are the common divisors to be recovered. We prove that it is hard to find the secret value ϕ_{11} from the first element c_1 of any number of ciphertexts.

Let $r_1 = \Sigma_{i=2}^{l+1} ru_i * \phi_{i1}$. Then, we have $c_1 = ru_1 * \phi_{11} + r_1 \bmod q$. Since ru_i $(1 \leq i \leq l+1)$ are random numbers generated for each encryption, r_1 is a number unknown to the adversary and randomly changes for each encryption even if the adversary chooses plaintexts. Moreover, r_1 can be bigger than ϕ_{11}. Hence, it is hard to recover ϕ_{11} from the first element c_1 of any number of ciphertexts according to the hardness of the AGCD problem. The proofs for other secret values in Φ are carried out similarly.

The upper level encryption in our scheme is built over the lower level encryption, in which the Γ component of $K(n)$ is used. The component Γ consists of n tuples of secret values, with each tuple independently used in the lower level encryption. Among a ciphertext, the element $c_{\Pi(n)}$ has the simplest definition which involves the secret value t_n. In this section, we prove that it is hard to recover t_n and the proofs for other secret values in Γ are similar.

Theorem 2. *Given any number of ciphertext elements $c_{\Pi(n)}$ from the upper level encryption with $K(n)$, it is hard to find t_n in Γ.*

Proof 2 *Let $\Phi = [\phi_1, ..., \phi_{l+1}]$, where $l \leq n-2$, and $\phi_i = (\phi_{i1}, ..., \phi_{i(n+1)})$. Thus, from the lower level encryption, we have*

$$\phi_{i\Pi(n)} = rs_{im} + t_n * rr_i \bmod q$$

and the element $c_{\Pi(n)}$ from the upper level encryption is defined as:

$$\begin{aligned} c_{\Pi(n)} &= \Sigma_{i=1}^{l+1} ru_i * \phi_{i\Pi(n)} \bmod q \\ &= \Sigma_{i=1}^{l+1} ru_i * (rs_{im} + t_n * rr_i) \bmod q \\ &= \Sigma_{i=1}^{l+1} ru_i * rs_{im} + t_n * (\Sigma_{i=1}^{l+1} ru_i * rr_i) \bmod q \end{aligned}$$

The rest of the proof is similar to the above one. Since ru_i $(1 \leq i \leq l+1)$ are random numbers generated for each encryption, we know that $\Sigma_{i=1}^{l+1} ru_i * rs_{im}$ is a number unknown to the adversary and randomly changes for each encryption. Moreover, $\Sigma_{i=1}^{l+1} ru_i * rs_{im}$ can be bigger than t_n. Hence, it is hard to recover t_n from the element $c_{\Pi(n)}$ of any number of ciphertexts according to the hardness of the AGCD problem.

For a ciphertext vector, the adversary cannot know exactly which element is $c_{\Pi(n)}$, since he does not know the permutation Π in $K(n)$. Hence, it is harder for the adversary to recover t_n from $c_{\Pi(n)}$.

5.3 Semantic Security

We analyze the semantic security of our scheme by proving the indistinguishability of ciphertexts under chosen-plaintext attacks (i.e., IND-CPA).

Theorem 3. *Given two plaintexts v and v' chosen by a probabilistic polynomial-time adversary, and a ciphertext C that encrypts v or v' with $K(n)$, the adversary can only distinguish whether C encrypts v or v' with a probability negligibly higher than $\frac{1}{2}$.*

Proof 3 *In our scheme, only the element $c_{\Pi(1)}$ in C is dependent on v or v'. Thus, if the adversary cannot distinguish whether $c_{\Pi(1)}$ encrypts v or v', then he cannot distinguish whether C encrypts v or v'. Moreover, the hardness of our key recovery problem proved above shows that other ciphertext elements $c_{\Pi(i)}$ $(2 \leq$*

$i \leq n+1$) cannot be exploited to determine secret key values used in $c_{\Pi(1)}$. Hence, we consider only $c_{\Pi(1)}$ to prove the semantic security of our scheme.

Let $\Theta = [\theta_1, ..., \theta_l]$, $\Phi = [\phi_1, ..., \phi_{l+1}]$, and $\phi_i = (\phi_{i1}, ..., \phi_{i(n+1)})$, where $l \leq n-2$. Thus, from the lower level encryption, by letting $S = \Sigma_{j=1}^{m} s_{1j} * rs_{ij}$, we have

$$\phi_{i\Pi(1)} = \begin{cases} a * t_1 * (\theta_i + \Sigma_{j=1}^{h-1} rv_{ij}) + S + t_1 * (r_{i1} - r_{ih}) \bmod q \ \textit{if}\ 1 \leq i \leq l \\ a * t_1 * (1 + \Sigma_{j=1}^{h-1} rv_{ij}) + S + t_1 * (r_{i1} - r_{ih}) \bmod q \ \ \textit{if}\ i = l+1 \end{cases}$$

Let v'' is either v or v'. Then, $c_{\Pi(1)}$ from the upper level encryption is defined as:

$$\begin{aligned} c_{\Pi(1)} &= \Sigma_{i=1}^{l+1} ru_i * \phi_{i\Pi(1)} \bmod q \\ &= a * t_1 * (\Sigma_{i=1}^{l} ru_i * \theta_i + ru_{l+1} + \Sigma_{i=1}^{l+1} ru_i * (\Sigma_{j=1}^{h-1} rv_{ij})) + \\ &\quad \Sigma_{i=1}^{l+1} ru_i * (\Sigma_{j=1}^{m} s_{1j} * rs_{ij}) + t_1 * (\Sigma_{i=1}^{l+1} ru_i * (r_{i1} - r_{ih})) \bmod q \\ &= a * t_1 * v'' + \Sigma_{i=1}^{l+1} ru_i * (a * t_1 * (\Sigma_{j=1}^{h-1} rv_{ij}) + \\ &\quad \Sigma_{j=1}^{m} s_{1j} * rs_{ij} + t_1 * (r_{i1} - r_{ih})) \bmod q \end{aligned}$$

Note that $v'' = \Sigma_{i=1}^{l} ru_i * \theta_i + ru_{l+1}$ according to the upper level encryption algorithm.

If $v \neq 0$ and $v' \neq 0$, then the expressions of $a * t_1 * v \bmod q$ and $a * t_1 * v' \bmod q$ generate the same value from 1 to $q-1$ with the same probability, since q is a prime and $a \neq 0$ and $t_1 \neq 0$ are uniformly sampled from $\mathbb{Z}_q$. Hence, at this case, the probability of distinguishing whether $c_{\Pi(1)}$ encrypts v or v' is just $\frac{1}{2}$, since the definition of $c_{\Pi(1)}$ differs only on the two indistinguishable expressions when encrypting v or v' . In the following, we discuss the case where $v = 0$ and $v' \neq 0$. The case where $v \neq 0$ and $v' = 0$ is similar.

Let $W = \Sigma_{i=1}^{l+1} ru_i * (a * t_1 * (\Sigma_{j=1}^{h-1} rv_{ij}) + \Sigma_{j=1}^{m} s_{1j} * rs_{ij} + t_1 * (r_{i1} - r_{ih})) - ru_1 * a * t_1 * rv_{11} \bmod q$. Then,at this case, we have either $c_{\Pi(1)} = W + ru_1 * a * t_1 * rv_{11} \bmod q$ or $c_{\Pi(1)} = a * t_1 * v' + W + ru_1 * a * t_1 * rv_{11} \bmod q$, depending on whether v or v' is encrypted. The advantage of distinguishing these two $c_{\Pi(1)}$ is negligible, if the advantage of distinguishing the value of $ru_1 * a * t_1 * rv_{11} \bmod q$ from the value of $a * t_1 * v' + ru_1 * a * t_1 * rv_{11} \bmod q$ is negligible, since the rest part of each $c_{\Pi(1)}$ (i.e., W) is the same.

Further, $a * t_1$ is the common factor in the expressions $ru_1 * a * t_1 * rv_{11} \bmod q$ and $a * t_1 * v' + ru_1 * a * t_1 * rv_{11} \bmod q$. Hence, the values of these two expressions can be distinguished only with a negligible advantage, if the advantage of distinguishing the value of $ru_1 * rv_{11} \bmod q$ from the value of $v' + ru_1 * rv_{11} \bmod q$ is negligible.

The value of $ru_1 * rv_{11} \bmod q$ is from a distribution, where the probability of $ru_1 * rv_{11} \bmod q = 0$ is $\frac{2*q-1}{q*q}$, while the probability of $ru_1 * rv_{11} \bmod q = z$, where $z \neq 0$, is $\frac{q-1}{q*q}$, since ru_1 and rv_{11} are uniformly sampled from $\mathbb{Z}_q$ and q is a prime. In the expression $v' + ru_1 * rv_{11} \bmod q$, v' is selected by the adversary. Hence, *the value of $v' + ru_1 * rv_{11} \bmod q$ is from a distribution, where the probability of $ru_1 * rv_{11} \bmod q = v'$ is $\frac{2*q-1}{q*q}$, while the probability of $ru_1 * rv_{11} \bmod q = z$, where $z \neq v'$, is $\frac{q-1}{q*q}$. That is, the two expressions have different provability for their values 0 and v', and the same probability for other $q - 2$ values.*

Thus, the advantage of distinguishing the value of $ru_1 * rv_{11} \bmod q$ from the value of $v' + ru_1 * rv_{11} \bmod q$ is $2 * (\frac{2*q-1}{q*q} - \frac{q-1}{q*q}) = 2 * (\frac{q}{q*q}) = \frac{2}{q}$. Let the bit length of q be b. Then, the advantage is a negligible function with respect to the bit length b.

Moreover, since the adversary cannot know exactly which element is $c_{\Pi(1)}$, it is harder for him to distinguish whether C encrypts v or v'.

6 Implementation and Evaluation

We have implemented a prototype of our scheme in Java and evaluated its concrete performance on a Dell XPS 13 laptop. In our implementation, all values (plaintexts, keys, random numbers and ciphertext elements) are represented with the Java class `BigInteger` and Eclipse is used to run Java programs.

6.1 Configuration

The public modulus q in our experiment is $q = 100000000000031$, which is a prime and thus a value in $\mathbb{Z}_q$ can be as big as 10^{14}. The key $K(n)$ is configured to have $n = 5$, with $h = 3$ and $m = 2$. Hence, the linear decryption keys dk_i $(1 \leq i \leq 6)$ for our experiment are the same as the example defined in Section 3.5.

We choose $l = 2$, such that $\Theta = [\theta_1, \theta_2]$ and $\Phi = [\phi_1, \phi_2, \phi_3]$. The three independent vectors in Φ mean that three linear decryption keys can be determined, with another three left as free variables. Hence, the space for linear decryption keys is $10^{14*3} = 10^{42} > 2^{128}$.

Homomorphic Enc	AES Enc	Homomorphic Dec	AES Dec
0.22	0.25	0.02	0.17

Table 1. Performance of Encryption and Decryption (seconds)

6.2 Performance of Encryption and Decryption

In this experiment, we evaluate the encryption and decryption performance of our scheme by comparing it with the AES algorithm provided in the Java security package SunJCE. The AES algorithm is configured to run in the CBC block mode with PKCS5 padding and it supports a 10^{128} key space in the SunJCE package.

In our evaluation, we randomly generate 10000 integers, each of which has 5 digits (e.g., 34845), and then use our encryption scheme and the AES algorithm to encrypt each integer, respectively. After all encryptions, we decrypt each corresponding ciphertext with our scheme and AES. Table 1 shows the average time of encryption and decryption performed by our scheme and AES. We can see our scheme is slightly faster than AES for encryption, while much faster for decryption.

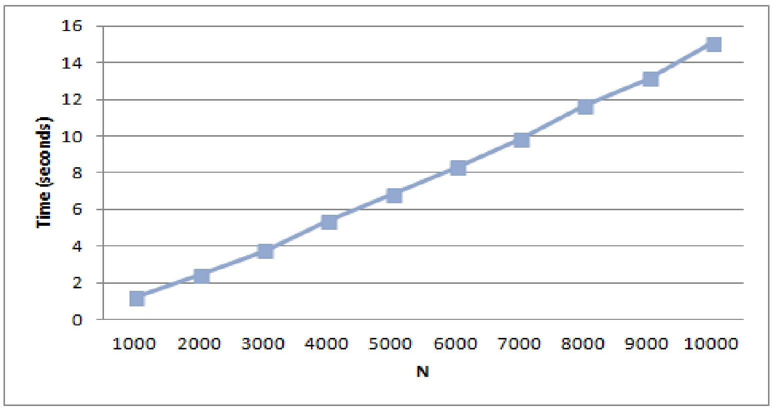

Fig. 1. Time for Running $\Sigma_{i=1}^{N} x_i^i$ over Encrypted x_i

6.3 Performance of Homomorphic Operations

We evaluate the performance of homomorphic addition and multiplication with high-degree polynomials over ciphertexts. The polynomial we used has the form $\Sigma_{i=1}^{N} x_i^i$, where x_i is the encryption of a randomly generated integer of five digits. The exponential function x_i^i is calculated by using the following formula.

$$x_i^i = \begin{cases} x_i (x_i^2)^{\frac{i-1}{2}} \bmod q & \text{if } i \text{ is odd} \\ (x_i^2)^{\frac{i}{2}} \bmod q & \text{if } i \text{ is even} \end{cases}$$

Figure 1 gives the time for evaluating $\Sigma_{i=1}^{N} x_i^i$ from $N = 1000$ to $N = 10000$. This experiment shows that our scheme is efficient to perform a large number of homomorphic operations. For example, the homomorphic operations for calculating $\Sigma_{i=1}^{N} x_i^i$ takes about 1 seconds when $N = 1000$ and about 15 seconds when $N = 10000$. Note that x_i^i is an exponential function with respect to i. The correctness of homomorphic addition and multiplication is also checked in this experiment.

6.4 Operations over Encrypted Bits

Bitwise encryption allows the comparison of encrypted integers. We have implemented a library that has functions to encrypt integers bitwise, and to support comparison, addition, and multiplication with bitwise operators, such as AND, XOR and NOT. In the following, we evaluate the performance of our scheme when comparing bitwise encryption of integers.

Let $V = v_1 v_2 ... v_N$ and $V' = v'_1 v'_2 ... v'_N$ be the binary representation of V and V'. We only report the performance of comparing whether V is bigger than V'. We use the following expression, which returns 1 if $V > V'$, and 0 otherwise.

$$\begin{array}{l}(v_1 \oplus v'_1)v_1 + (v_1 \oplus v'_1 \oplus 1)(v_2 \oplus v'_2)v_2 + ...+ \\ (v_1 \oplus v'_1 \oplus 1)...(v_{N-1} \oplus v'_{N-1} \oplus 1)(v_N \oplus v'_N)v_N\end{array}$$

In the above expression, $\oplus$ is overloaded to represent the XOR operation and defined as $v \oplus v' = v + v' - 2vv'$.

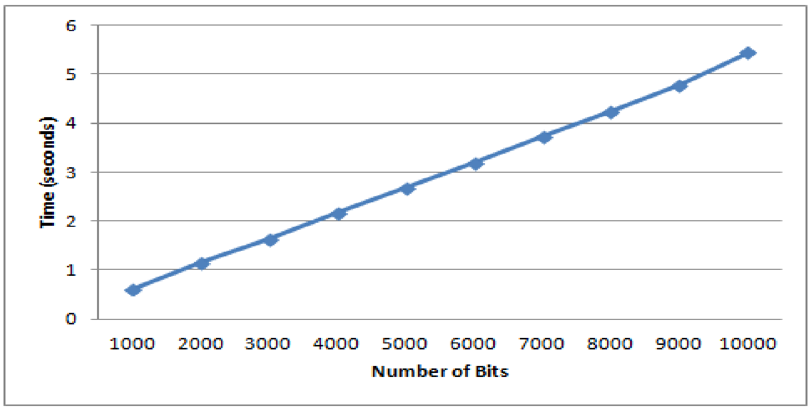

Fig. 2. Performance of Comparing Encrypted Integers

In this experiment, we encrypt each bit v_i and v'_i of V and V', and then evaluate the above expression over encrypted bits. Figure 2 gives the performance with the bit number N increasing from 1000 to 10000. As shown by this experiment, we can efficiently compare bitwise encryption of integers with our scheme.

6.5 An Application: Keyword-Based Oblivious Transfer

As an application example, we implemented a keyword-based 1-out-of-n oblivious transfer protocol with our FHE scheme. In this protocol, we suppose the sender has a table, consisting of T entries, and each entry consists of a keyword KW_i

and a value V_i for $1 \leq i \leq T$. The keyword KW_i is assumed to be unique in the table, and the table is not encrypted by the sender. A receiver may want to retrieve an entry by specifying a keyword KW_r.

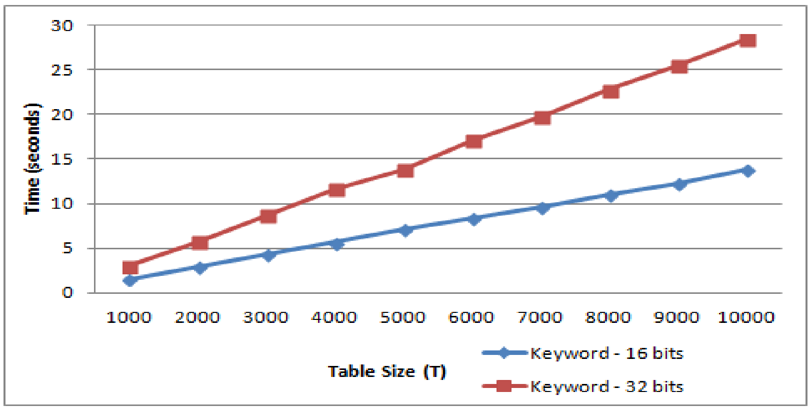

Fig. 3. Performance of Oblivious Transfer

Let $KW_i == KW_r$ denote 1 if the two keywords KW_i and KW_r match, and 0 otherwise. Without considering the oblivious requirement, the sender can answer the receiver's request by sending the result of the following expression.

$$(KW_1 == KW_r) * V_1 + ... + (KW_T == KW_r) * V_T$$

If KW_r is not a valid keyword in the table, the result is 0; otherwise, only the value V_i with the matched keyword is returned, since we assume each keyword is unique.

To make the transfer oblivious, the receiver encrypts the request KW_r in bits with his own key $K(n)$, and then sends the encrypted keyword to the sender, together with the modulus, the public evaluation key, and an encryption of 1 (i.e., $Enc(K(n), 1)$). Note that for an integer v, we have $v \odot Enc(K(n), 1)) = Enc(K(n), v))$. Hence, $Enc(K(n), 1))$ is used by the sender to change a table entry (KW_i, V_i) into an entry encrypted with the receiver's key $K(n)$. Note that KW_i needs to be encrypted in bits for supporting comparison, while V_i is just encrypted as an integer. After such encryption, the above expression can be evaluated over ciphertexts with homomorphic operations.

Figure 3 shows the performance of our oblivious transfer protocol, where keywords are represented with 16 bits or 32 bits. Since KW_r is encrypted, the sender does not know which entry is selected by the receiver. On the other hand, the table entries not matched with KW_r is not included in the result, so the receiver only knows the table entry being selected.

7 Conclusion

In this paper, we presented a new FHE scheme, which allows arbitrarily large noises in ciphertexts. Hence, it does not need any noise reduction mechanism, such as bootstrapping and modulus switching, which is considered as the most essential technique in current FHE schemes. Our scheme is compact, since homomorphic operations do not change the size of ciphertexts. These features makes our scheme efficient and also makes it simple to implement and use in data processing applications.

We proved the security of our scheme from two aspects: the hardness of finding secret key values from ciphertexts and semantic security. The hardness of recovering secret key values from ciphertexts is based on the approximate GCD problem. We implemented a prototype in Java and evaluated the performance on encryption, decryption, homomorphic operations, and bitwise operators over ciphertexts. An 1-out-of-n oblivious transfer protocol has been implemented as an application of our scheme and its performance is also evaluated. Our evaluation confirmed that our scheme is efficient for practical applications.

References

1. Z. Brakerski, , and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*, CRYPTO'11, pages 505–524, Berlin, Heidelberg, 2011. Springer-Verlag.
2. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.
3. Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, July 2014.
4. Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society.
5. Y. Chen and P. Q. Nguyen. Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 502–519, Berlin, Heidelberg, 2012. Springer-Verlag.
6. H. Cohn and N. Heninger. Approximate common divisors via lattices. *IACR Cryptology ePrint Archive*, 2011:437, 2011.
7. J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Proceedings of the 31st annual conference on Advances in cryptology*, CRYPTO'11, pages 487–504, Berlin, Heidelberg, 2011. Springer-Verlag.
8. C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 169–178, New York, NY, USA, 2009. ACM.

9. C. Gentry and S. Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 107–109, Washington, DC, USA, 2011. IEEE Computer Society.
10. C. Gentry and S. Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Proceedings of the 30th Annual international conference on Theory and applications of cryptographic techniques: advances in cryptology*, EUROCRYPT'11, pages 129–148, Berlin, Heidelberg, 2011. Springer-Verlag.
11. S. Halevi and V. Shoup. Bootstrapping for helib. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques,*, pages 641–670, 2015.
12. N. Howgrave-Graham. *Cryptography and Lattices*, volume 2146 of *Lecture Notes in Computer Science*, chapter Approximate Integer Common Divisors, pages 51–66. 2001.
13. M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.
14. R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 85–100, 2011.
15. S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 289–300, 2013.
16. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 24–43, Berlin, Heidelberg, 2010. Springer-Verlag.