Research Article

Improved Cryptanalysis of a Fully Homomorphic Symmetric Encryption Scheme

Quanbo Qu,1 Baocang Wang,1,2 Yuan Ping,2 and Zhili Zhang2

1State Key Laboratory of Integrated Service Networks, Xidian University, Xian 710071, China
2School of Information Engineering, Xuchang University, Xuchang 461000, China

Correspondence should be addressed to Zhili Zhang; zlzhangxc@163.com

Received 5 March 2019; Accepted 22 April 2019; Published 2 June 2019

Homomorphic encryption is widely used in the scenarios of big data and cloud computing for supporting calculations on ciphertexts without leaking plaintexts. Recently, Li et al. designed a symmetric homomorphic encryption scheme for outsourced databases. Wang et al. proposed a successful key-recovery attack on the homomorphic encryption scheme but required the adversary to know some plaintext/ciphertext pairs. In this paper, we propose a new ciphertext-only attack on the symmetric fully homomorphic encryption scheme. Our attack improves the previous Wang et al.’s attack by eliminating the assumption of known plaintext/ciphertext pairs. We show that the secret key of the user can be recovered by running lattice reduction algorithms twice. Experiments show that the attack successfully and efficiently recovers the secret key of the randomly generated instances with an overwhelming probability.

1. Introduction

With the rapid development of big data, the significance of privacy and security issues was highly regarded. A series of cryptographic applications, such as fair electronic transaction [1], outsourcing data classification [2], lightweight security system of Internet of Things [3], mobile Ecommerce [4], and data mining based on homomorphic encryption, have been proposed.

Homomorphic encryption schemes allow users to meaningfully calculate ciphertexts without knowing the underlying plaintexts. For example, the RSA cryptosystem [5] (Pallier cryptosystem [6], respectively) only supports homomorphic multiplications (additions, respectively) on ciphertexts. In 2009, Gentry [7] designed the first fully homomorphic encryption scheme with ideal lattices. Thereafter, significant efforts had been performed to improve the efficiency of homomorphic encryption schemes [8–10]. However, all the known fully homomorphic encryptions are criticized for the high ciphertext expansion and ciphertext refreshing costs and hence cannot be directly used in practice. So researchers designed some cryptographic schemes with homomorphic properties dedicated to some concrete computing scenarios [11–15].

Recently, Li et al. [16] designed a symmetric homomorphic encryption scheme for outsourced databases that allow multiple data owners to efficiently share their data securely without compromising the privacy of the data. However, Wang et al. [17] observed that if some plaintext/ciphertext pairs were successfully overdropped, one can efficiently recover the corresponding secret key of the scheme from the obtained plaintext/ciphertext pairs. We show that the secret key of the user can be recovered by running lattice reduction algorithms twice. Experiments show that the attack successfully and efficiently recovers the secret key of the randomly generated instances with an overwhelming probability.

This paper is organised as follows. In Section 2, we review Li et al.’s symmetric homomorphic encryption scheme and
introduce the concept of lattice. In Section 3, we propose our attack and give the experimental results. In Section 4, we conclude our work.

2. Preliminaries

2.1. Notations. In this paper, the symbol $\mathbb{Z}$ is used to denote the ring of integer. Matrices are represented with bold upper-case characters like $\mathbf{B}$, while vectors are represented with bold lower-case characters like $\mathbf{v}$. All of the vectors in this paper are represented as row vectors. The symbol $\|\mathbf{v}\|$ means the length of vector $\mathbf{v}$ under the Euclidean norm, while the symbol $|p|_2$ means the bit length of integer $p$.

The symbol $a \ll b$ means "much less than", $i.e.$ if $a \ll b$, the ratio $a/b$ is a negligible function $\text{negl}(\lambda)$ of the security parameter $\lambda$. In mathematics, a negligible function $\mu(x)$ means that for any polynomial function $\text{poly}(x)$, there exists an integer $N_c$ such that for any $x > N_c$,

$$\mu(x) < \frac{1}{\text{poly}(x)}. \quad (1)$$

2.2. Symmetric Homomorphic Encryption. The symmetric homomorphic encryption scheme proposed by Li et al. comprises these three algorithms as follows:

(i) **Key generation algorithm KeyGen**

$$(s, q, p) \leftarrow \text{KeyGen}(\lambda). \quad (2)$$

Input a security parameter $\lambda$, this algorithm outputs a secret key $SK = (s, q)$ and a public parameter $p$, where $q \ll p$.

(ii) **Encryption algorithm E**

$$c = E(SK, m, d) = s^d (rq + m) \pmod{p}. \quad (3)$$

Input a secret key $SK$, a plaintext $m \in \mathbb{F}_q$ and a parameter $d$, this algorithm outputs a ciphertext $c = E(SK, m, d)$. Notice that the parameter $r$ should satisfy $|r_2| + |q_2| < |p|_2$.

(iii) **Decryption algorithm D**

$$m = D(SK, c, d) = (c \times s^{-d} \pmod{p}) \pmod{q}. \quad (4)$$

Input a secret key $SK$, a ciphertext $c \in \mathbb{F}_p$ and the ciphertext’s degree $d$, this algorithm outputs a ciphertext $c \leftarrow E(SK, m, d)$. The proof of the correctness is simple:

$$D(SK, c, d) = (c \times s^{-d} \pmod{p}) \pmod{q}$$

$$= \left( (s^d (rq + m) \pmod{p}) \times s^{-d} \pmod{p} \right) \pmod{q}$$

$$= (rq + m) \pmod{q}$$

$$= m.$$ 

Notice that the correctness of $(rq + m)(\pmod{q}) = m$ requires $q \ll p$ and $|r_2| + |q_2| < |p|_2$.

The symmetric homomorphic encryption scheme proposed by Li et al. supports homomorphic addition and multiplication and is used to construct their secure outsourced comparison scheme and privacy-preserving mining solutions. Though our attack needs no homomorphic properties, we still list a brief proof, for the reason that it implies the setting of parameters.

(i) **Homomorphic addition**: For the ciphertext $c_1, c_2$ of two plaintexts $m_1, m_2$, we have

$$(c_1 + c_2) \pmod{p} = s^d (r_1q + m_1) \pmod{p}$$

$$+ s^d (r_2q + m_2) \pmod{p} \quad (6)$$

$$= s^d ((r_1 + r_2)q + m_1 + m_2) \pmod{p}, \quad d_1 = d_2.$$

The correct decryption of $c_1 + c_2$ requires $d_1 = d_2$ and $(r_1 + r_2)q + m_1 + m_2 < p$.

(ii) **Homomorphic multiplication**: For the ciphertext $c_1, c_2$ of two plaintexts $m_1, m_2$, we have

$$(c_1 \times c_2) \pmod{p} = s^d (r_1q + m_1) \pmod{p}$$

$$\times s^d (r_2q + m_2) \pmod{p}$$

$$= s^{d_1 + d_2} (r_1r_2q^2 + r_1qm_2 + r_2qm_1 + m_1 \times m_2) \pmod{p} \quad (7)$$

$$= s^{d_1 + d_2} ((r_1r_2q + r_1m_2 + r_2m_1)q + m_1 \times m_2) \pmod{p}.$$ 

The correct decryption of $c_1 \times c_2$ requires $(r_1r_2q + r_1m_2 + r_2m_1)q + m_1 \times m_2 < p$.

2.3. Lattice. An $m$-dimension lattice $\mathcal{L}$ can be regarded as a set of all integer coefficient linear combinations of basis vectors $\{b_1, b_2, \ldots, b_m\}$, that is $\mathcal{L}(\mathbf{B}) = \{v = \sum_{i=1}^{m} a_i b_i | a_i \in \mathbb{Z}, \ b_i \in \mathbb{R}^m\}$. If $m = n$, we call that $\mathcal{L}$ is a full-rank lattice.

One of the most famous problems involving lattice is the shortest vector problem (SVP). Given a basis of a lattice, the goal is to find one non-zero vector, which has the shortest length $\|\mathbf{v}\|$. Some approximation algorithms are usually used for solving SVP as oracles, such as LLL and BKZ algorithms. The LLL algorithm is developed by A. K. Lenstra, H. W. Lenstra, Jr. and L. Lovasz [18] in 1982. Given a basis of a lattice, the LLL algorithm outputs a reduced basis, which has a smaller size by Gram-Schmidt orthogonalization. It has various applications in cryptanalysis and other fields, such as integer programming and finding integer relations.

3. The Proposed Attack

3.1. Main Idea. Define $r_1q + m_1$ as $a_i$, and $c_i c_1^{-1}$ as $f_i$. Our attack has two stages. In the first stage, we construct a lattice $\mathcal{L}(\mathbf{F})$
with f’s and run the LLL algorithm to obtain a short vector, which contains a’s. In the second stage, we construct a lattice \( L(A) \) with a’s and run the LLL algorithm again to obtain a short vector which contains r’s. It is obvious that the secret key \( (s^d, q) \) can be computed as \( s^d = ca^{-1} \mod p \) and \( q = a/r \mod p \). Notice that there is no need for the plaintexts \( m_i \)’s in the attack.

### 3.2. Details.

In this part, we give a specification of the attack in Algorithm 1. The input of the attack algorithm contains a set of ciphertexts \( c = \{c_1, \ldots, c_k\} \) and the modular \( p \) of the encryption scheme without plaintexts \( m_i \)’s. The output of the attack algorithm contains \( s^d \) and \( q \) which can be used to decrypt ciphertexts.

In the first stage, the lattice \( L(F) \) is constructed as

\[
F = \begin{pmatrix}
  p & 0 & \cdots & 0 \\
  0 & p & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & p \\
\end{pmatrix}
\begin{pmatrix}
  v_1 \\
  v_2 \\
  \vdots \\
  v_k \\
\end{pmatrix}
= \begin{pmatrix}
  v_{1} \\
  v_{2} \\
  \vdots \\
  v_{k} \\
\end{pmatrix}.
\] (8)

Thus a short vector \( v \in L(F) \) could be expressed as

\[
v = x_1v_1 + x_2v_2 + \cdots + x_kv_k = (x_1p - x_kf_2, x_2p - x_kf_3, \ldots, x_{k-1}p - x_kf_{k-1}, x_k).
\] (9)

Through LLI’s algorithm, we could obtain a short vector \( f \in L(F) \).

**Claim 1.** Parameters \( a_1, \ldots, a_k \) are close to \(|f_1|, \ldots, |f_{k-1}| \), while \( a_i \) is close to \(|f_i|, \) where \( f_i \) means the \( i \)-th entry of the output vector \( f \).

**Sketch of Proof.** Suppose that \( f_i a_i = a_i + t_i p \), then we have \( f_i p - t_i a_i = a_i/(a_i p) = 1/p < \text{negl}(\lambda) \). Hence, values of \( f_i/p, \ldots, f_k/p \) are close to those of \( t_2/a_1, \ldots, t_k/a_k \). Because \(|x_{k-1}p - x_k f_i| \) is small, we can obtain that \( x_{k-1}/x_k = f_i/p < \text{negl}(\lambda) \). Hence, values of \( \{f_i/x_i, \ldots, f_k/x_k \} \) are close to those of \( \{f_i/p, \ldots, f_k/p \} \).

Since \( t_2/a_1, \ldots, t_k/a_k \) and \( x_1/x_k, \ldots, x_{k-1}/x_k \) are both close to \( f_2/p, \ldots, f_k/p \), we obtain that values of \( t_1/a_1, \ldots, t_k/a_k \) are close to \( x_1/x_k, \ldots, x_{k-1}/x_k \). With a non-negligible probability, \( x_1, \ldots, x_{k-1} \) equal \( t_2, \ldots, t_k \), and \( x_k \) equal to \( a_i \).

As \(|a_i| \) is close to \(|x_k| \), we have \( x_k = a_i \) or \( x_k = -a_i \).

Considering \(|a_i|, |x| < p \), we believe that \(|a| < p/2 \) then \( a_i = |x_k| \). Similarly, \(|a_i p - x_k f_i| \) is small, we have \(|x_k p - x_k f_i| = (x_k p - x_k f_i) \mod p = -x_k f_i \mod p \) or \( |x_k p - x_k f_i| = -(x_k p - x_k f_i) \mod p = x_k f_i \mod p \). Hence \( |x_k p - x_k f_i| = x_k f_i \mod p \) then \( a_i = a_i f_{i+1} \mod p = a_i f_{i+1} = f_i \), \( i = 1, \ldots, k - 1 \).

In the second stage, the lattice \( L(A) \) is constructed as

\[
A = \begin{pmatrix}
  a_1 & 0 & \cdots & 0 \\
  0 & a_1 & \cdots & 0 \\
  \vdots & \vdots & \ddots & \vdots \\
  0 & 0 & \cdots & a_1 \\
\end{pmatrix}
\begin{pmatrix}
  w_1 \\
  w_2 \\
  \vdots \\
  w_k \\
\end{pmatrix}
= \begin{pmatrix}
  w_1 \\
  w_2 \\
  \vdots \\
  w_k \\
\end{pmatrix}.
\] (10)

Thus a short vector \( w \in L(A) \) could be expressed as

\[
w = y_1w_1 + y_2w_2 + \cdots + y_{k-1}w_{k-1} + y_kw_k
\]

\[
= (y_1a_1, y_2a_1, y_3a_1, \ldots, y_{k-1}a_1, y_ka_1).
\] (11)

Through LLI’s algorithm, we could obtain a short vector \( a \in L(A) \).

**Claim 2.** Parameters \( r_1, r_2, \ldots, r_k \) are close to \(|a_i|, |[(a)_1]| + r_1 \cdot a_1/a_i, \ldots, |[(a)_{k-1}]| + r_1 \cdot a_1/a_i, \) where \( (a)_i \) means the \( i \)-th entry of the output vector \( a \).

**Sketch of Proof.** Considering \( a_i = r_q + m_i \), we have \( a_i/a_i = (r_q + m_i)/(r_q + m_i) = (r_q)/(r_q) = 1/r_q \). Hence, values of \( a_2/a_1, \ldots, a_k/a_1 \) are close to those of \( r_2/r_1, \ldots, r_k/r_1 \).

Because \(|y_{k-1} - y_1 a_1| < \text{negl}(\lambda) \), we can obtain that \( y_{k-1}/y_1 - a_1/a_1 < \text{negl}(\lambda) \). Hence, values of \( y_{1}/y_k, \ldots, y_{k-1}/y_k \) are close to those of \( a_2/a_1, \ldots, a_k/a_1 \). Since \( r_2/r_1, r_3/r_1, \ldots, r_k/r_1 \) and \( y_1/y_k, \ldots, y_{k-1}/y_k \) are both close to \( a_2/a_1, \ldots, a_k/a_1 \), we obtain that values of \( r_2/r_1, \ldots, r_k/r_1 \) are close to \( y_1/y_k, \ldots, y_{k-1}/y_k \). With a non-negligible probability, \( y_1, \ldots, y_{k-1} \) equal \( r_2, \ldots, r_k \), and \( y_k \) equal to \( r_1 \).

Likewise, we have \( r_1 = |(a)_1| \) and \( r_{i+1} = |[(a)_i]| + r_1 \cdot a_1/a_i, \) \( i = 1, \ldots, k - 1 \).

Since we have recovered all the a’s and r’s, the secret key \( (s^d, q) \) could be simply computed as \( s^d = c_0a_0^{-1} \) and \( q = a_0/r_1 \). Parameters \( c_0, a_0 \) and \( r_1 \) are used to compute \( (s^d, q) \) in the algorithm.

**Algorithm 1:** The Attack Algorithm.
Table 1: Results in experiments with different parameters.

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$r$</th>
<th>$m$</th>
<th>$k$</th>
<th>Instances</th>
<th>Successes</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>64</td>
<td>64</td>
<td>$\leq q/256$</td>
<td>20</td>
<td>100</td>
<td>98</td>
<td>28.65s</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>64</td>
<td>$\leq q/6$</td>
<td>20</td>
<td>100</td>
<td>10</td>
<td>29.20s</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>96</td>
<td>$\leq q/256$</td>
<td>20</td>
<td>100</td>
<td>23</td>
<td>39.70s</td>
</tr>
<tr>
<td>256</td>
<td>64</td>
<td>48</td>
<td>$\leq q/256$</td>
<td>20</td>
<td>100</td>
<td>2</td>
<td>25.09s</td>
</tr>
<tr>
<td>256</td>
<td>48</td>
<td>48</td>
<td>$\leq q/3$</td>
<td>20</td>
<td>100</td>
<td>51</td>
<td>20.56s</td>
</tr>
</tbody>
</table>

Table 2: Example for $|p|_2 = 256$.

The Encryption Scheme

KeyGen

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$s$</th>
<th>$d$</th>
<th>$m_1$</th>
<th>$r_1$</th>
<th>$c_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9667660090081618538103422777895287998619</td>
<td>438084870993447026698628675235799451</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17957200991146257161</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5710889004555322387</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4786322678359308</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136839097014700313</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1403306518881428241485832704008411265962</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The Cryptanalytic Algorithm

The First Stage

<table>
<thead>
<tr>
<th>$v_1$</th>
<th>$\ldots$</th>
<th>$v_{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-199463957636154746239994027808094961893</td>
<td>$\ldots$</td>
<td>24572470551643938262358136850318784901</td>
</tr>
</tbody>
</table>

The Second Stage

<table>
<thead>
<tr>
<th>$w_{20}$</th>
<th>$s$</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>136839097014700313</td>
<td>5710889004555322387</td>
<td>17957200991146257161</td>
</tr>
</tbody>
</table>

3.3. Experiments. We run our proposed cryptanalytic algorithm on a personal computer using NTL library [19]. The environment is listed as follows:

(i) CPU: Intel(R) Core(TM) i3-7100 3.90GHz
(ii) RAM: 4.00GB
(iii) OS: Windows 10 64bit

Notice that the output of the attack algorithm is $s = d$. In [16], the parameter $d$ is called ciphertext degree and is believed to be a small positive integer. It means that we could collect enough $d$-degree ciphertexts we need, and it is not difficult to recover $s$ from $s^d$. For the $d$-degree ciphertexts, the encryption and decryption algorithms only require $s^d$ rather than $s$. Thus, it is sufficient to break the scheme if we can recover $s^d$. For convenience, we suppose the parameter $d = 1$ in the encryption algorithm. When $d \neq 1$, our algorithm still works correctly.

The results are given in Table 1. As a result of the approximation, the chance of success is relevant to $|q|_2$, $|r|_2$, and $|m|_2$. The best situation is when $|q|_2 = |r|_2$ and $|m| \ll |r|$. To make it easier to understand our proposed attack, we give an example to illustrate the procedure of the algorithm in Table 2. The parameters are set as $|p|_2 = 256$, $|q|_2 = 64$, $|r|_2 = 64$, and $m \leq q/256$.

Firstly, we compute all the $f$’s with the input ciphertexts $(c_1, \ldots, c_k)$. Secondly, we use LLL algorithm to obtain a short vector for solving all $a$’s. Thirdly, we use LLL algorithm again to obtain a short vector for solving all $r$’s. Finally, we compute secret key $(s, q)$ with $a$’s and $r$’s.

In practice, the first row $f$ (a, respectively) of the reduced basis of $\mathcal{F}(\mathcal{A})$ (respectively) which is a row vector with a short norm; thus we regard it as the short vector $v$ (w, respectively) we need.

The chance of success depends on the bit lengths of $r$, $q$, and $m$. In the first stage, $|a| \ll |p|$ requires $|r|q \ll |p|$. In the second stage, $a_{i} / a_1 = (r_{i}q + m_{i}) / (r_{1}q + m_{1}) \approx (r_{i}q) / (r_{1}q) = r_{i} / r_1$ requires $|m| \ll |r|$. Thus we need to hold $|m| \ll |r|q \ll |p|$. Besides, the recovery of $q$ from $a$ also limit the setting of parameters. Notice that $a = rq + m$, where $0 \leq m < q$. If $m$ <
than a reduced basis. We regard the first row vector of an LLL-reduced basis as the goal short vector. We explain the reason below.

An $\delta$-LLL-reduced $B = \{b_1, b_2, \ldots, b_n\}$ has two important properties:

1. $\forall 1 \leq i \leq n, j < i, |\mu_{ij}| \leq 1/2$,
2. $\forall 1 \leq i \leq n, \delta \|b_i\|^2 = \|a_{i+1,j}\|^2 + \|b_{i+1}\|^2$,

where $B = \{b_1, b_2, \ldots, b_n\}$ is the Gram-Schmidt orthogonalization of $B = \{b_1, b_2, \ldots, b_n\}$, and the coefficient $\mu_{ij} = \langle b_i, b_j \rangle / \langle b_i, b_i \rangle$.

From these two properties, we can conclude that

$$\|b_i\| \leq 2^{(n-1)/2}\lambda_1(\mathcal{L}),$$

where $\lambda_1(\mathcal{L})$ is the length of the shortest non-zero vector in $\mathcal{L}(B)$. Please refer to [21] for more detailed introduction and proof.

The efficiency of our attack algorithm is mainly subject to the parameter $k$. Smaller $k$ implies a greater chance of success for the reason that $a$'s and $r$'s can be recovered from the LLL algorithm easier while the runtime of the LLL algorithm rises rapidly. In our experiment, we recommend that $k$ should be set as 20 considering both the chance of success and the runtime. In addition, the chance of success is also limited by sizes of $r, q$, and $m$.

Table 3 gives a comparison of Wang et al.'s attack and ours. Although the bit lengths of the parameters $|p|_2$, $|q|_2$, and $|r|_2$ are close but different, the average time of Wang et al.'s is much less than ours. However, the improved attack algorithm eliminates the assumption of known plaintext/ciphertext pairs, thus a ciphertext-only adversary can break the encryption scheme through this way.

### 4. Conclusion

In this paper, we propose a new attack algorithm on the symmetric homomorphic encryption scheme presented by Li et al. Our attack can recover the secret key pair from several ciphertexts without plaintexts. In our experiment, the attack can be finished during an acceptable period of time with recovering more of the secret key in the generated instances. For the cases $|p|_2 = 256$, $|q|_2 = |r|_2 = 64, m \leq q/256$, the key-recovery cryptanalytic algorithm only takes about 29 seconds. Although the running time and the opportunity of success depend on the sizes of parameters, the attack algorithm can still be used in real practice to recover secret key pairs.

### Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.
Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key R&D Program of China under Grant No. 2017YFB0802000, the National Natural Science Foundation of China under Grant Nos. 61572390, U1736111, the National Cryptography Development Fund under Grant No. MMJ20180111, the Plan For Scientific Innovation Talent of Henan Province under Grant no. 184100510012, the Program for Science & Technology Innovation Talents in Universities of Henan Province under Grand No. 18HASSTIT022, and the Innovation Scientists and Technicians Troop Construction Projects of Henan Province, Science & technology planning project in Henan Province (182102210124).

References
