SCIS 2017 2017 Symposium on
Cryptography and Information Security
Naha, Japan, Jan. 24 - 27, 2017
The Institute of Electronics,
Information and Communication Engineers

Timing and Fault Attacks on Lattice-based Cryptographic Libraries

Hyeongcheol An* Sungsook Kim † Jeeun Lee † Rakyong Choi † Kwangjo Kim *†

Abstract: Lattice-based cryptography is based on mathematical hard problem such as Learning with Error(LWE) and Ring-Learning with Error(Ring-LWE). These problems can be used for Fully Homomorphic Encryption(FHE) which attracts a lot of attention in the field of cloud computing environment and big data processing. On the other hand, side-channel attacks on a lot of cryptographic libraries have been tried except lattice-based cryptosystems. We first try to timing attack in lattice-based cryptographic libraries such as FHEW and HElib. Timing attack is simple and relatively easy to approach than other hardware-based side-channel attacks. In this paper, we compare and analyze four lattice-based open cryptographic libraries such as FHEW, HElib, $\Lambda \circ \lambda$, and LatticeCyrpto. In particular, aftter checking the time traces of the FHEW and HElib libraries, we investigate to determine if timing and fault attacks are feasible. As a countermeasure against timing attacks, we present a method for a padding to make of a message constant time encryption.

Keywords: Timing Attack, Fault Attack, Lattice-based Cryptographic Library

1 Introduction

Public key cryptosystems such as RSA and Diffie-Hellman (DH) are based on the difficulty of Integer Factoring Problem(IFP) or Discrete Logarithm Problem(DLP). However, by using a quantum computer, IFP and DLP can be solved within the polynomial time by Shor's algorithm [1]. Therefore, we need for a replacement for the current public key cryptosystems against the quantum computer attack. A lattice-based cryptography is known as one of the post-quantum cryptographies. We focus on lattice-based cryptography which is based on mathematical hard problem such as Learning with Error(LWE) and Ring-Learning with Error(Ring-LWE). In addition, lattice-based cryptography can be used not only for encryption and decryption but also for digital signatures and key exchange protocols. Therefore, we investigate the cryptographic features of four libraries such as FHEW[2], HElib[3], $\Lambda \circ \lambda$[4], and LatticeCyrpto [5].

In addition, Homomorphic Encryption(HE) technology plays an important role in keeping personal information in the cloud computing environment and big data processing. HE can be divided into Fully Homomorphic Encryption(FHE) and Somewhat Homomorphic Encryption(SHE) satisfying the homomorphic property for all operations and specific operations, respectively. The homomorphic properties are that the operations on ciphertext are also matched to the one of plaintexts so that the ciphertext can be computed without decryption. Therefore, this advantage leads to handle data without exposing sensitive personal information in cloud computing environment. Since the process of big data processing and decoding in the server environment is unnecessary, the amount of computation of the server can be reduced.

On the other hand, side-channel attacks on a lot of cryptographic libraries have been tried except lattice-based cryptosystems. We first try to timing attack in lattice-based cryptographic libraries such as FHEW and HElib. Timing attack is simple and relatively easy to approach compared to hardware-based side-channel attacks.

In this paper, we compare and analyze four lattice-based open-source cryptographic libraries such as FHEW, HElib, $\Lambda\circ\lambda$, and LatticeCyrpto. Also, we check whether timing and fault attacks are possible for two lattice-based libraries such as FHEW and HElib libraries and suggest a solution with constant operation time for arbitrary message length.

1.1 Outline of the Paper

In Section 2, we introduce the well-known lattice-based hard problems and HE. Section 3 introduces two publications which tried to attack lattice-based problem or cryptographic primitives by using power and faults attacks. Then, we compare and analyze four different lattice-based cryptographic open-source libraries such as FHEW, HElib, $\Lambda \circ \lambda$, and LatticeCrypto libraries in Section 4. In Section 5, we perform timing and fault attacks on FHEW and HElib libraries. Then, we give our constant time encryption with message padding to prevent against timing attack and concluding remarks in Sections 6 and 7, respectively.

* Grauate School of Information Security, KAIST. 291, Daehak-ro, Yuseong-gu, Daejeon, South Korea 34141. {$anh1026, kkj$}@kaist.ac.kr

† School of Computing, KAIST. 291, Daehak-ro, Yuseong-gu, Daejeon, South Korea 34141. {$kusino, jeeun.lee, thepride, kkj$}@kaist.ac.kr

2 Lattice-based Cryptography

In this section, the well-known lattice-based problems and the HE based on these problems will be described in brief.

2.1 Lattice-based Problems

A lattice Λ can be defined as a discrete subgroup of $\mathbb{R}^m$ with its basis $\mathcal{B}$. A basis $\mathcal{B}$ of Λ is a set of linearly independent vectors $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_m\}$ which spans the lattice Λ and $\mathbf{B} = (\mathbf{b}_1|\mathbf{b}_2|\cdots|\mathbf{b}_m)$ is called a basis matrix of the lattice Λ.

One of the hardness of lattice problem is the worst-case hardness assumption. Of them, Shortest Vector Problem(SVP) is to find the shortest nonzero lattice vector to a given vector. Closest Vector Problem(CVP) is to find the closest lattice vector to a given vector. On the other hand, there are average-case hardness problems which can be reduced to the worst-case hardness problem in lattice, *e.g.,* Short Integer Solution(SIS) and LWE problems.

LWE problem is a different to efficiently distinguish vectors created from a noisy set of linear equations between uniformly random vectors. Solving LWE problem is approximately as hard as solving SVP problem. LWE application serves as a versatile problem used in construction of cryptographic primitives. However, a drawback of using LWE is large key size. On the other hands, Ring-LWE problem requires an algebraic variant of LWE, which operates over elements of polynomial rings instead of vectors.

2.2 Homomorphic Encryption

HE[6] is a form of encryption which allows specific types of computations to be carried out on ciphertexts and generates an encrypted result which, when decrypted, matches the result of operations performed on the plaintexts:

$$H_k(m_1) \circ H_k(m_2) = H_k(m_1 \circ m_2) \tag{1}$$

where, $H_k(m_1)$ is HE of message, m_1 using key, k and $\circ$ is a specific homomorphic operation.

FHE scheme [7] is that given ciphertexts that encrypt $m_1, ..., m_t$, anyone can output a ciphertext that encrypts $f(m_1, ..., m_t)$ for any desired function f, as long as that function can be effectively computed.

All HE schemes based on lattice are from Gentry's seminal paper[7]. Encryption scheme has a noise parameter which is attached to each ciphertext thus encryption outputs a ciphertext with a noise. So, decryption works as long as the noise is less than some threshold. However this noise grows with every operation, resulting decryption error reaches in some bound. This yields SHE scheme that can handle only low-depth circuits. SHE can be modified into FHE by bootstrapping.

Bootstrapping is a technique for refreshing ciphertext of SHE. After refreshing by bootstrapping, the ciphertext is still encrypted form of plaintext with low error. Lattices are a good way for bootstrapping since their complexity is very low and some algebraic operations from the polynomial rings are very efficient. So, many studies about lattice-based cryptographic library with low–cost bootstrapping have been done.

In BGV[8], the essence of the modulus–switching technique means that an evaluator, who does not know the secret key s but instead of only knows a bound on its length, can transform a ciphertext c modulo q into a different ciphertext modulo p while preserving correctness. Here p is smaller than q.

3 Previous Work

Even though lots of researches on lattice-based cryptography have been studied for various cryptographic applications, no extensive effort has been executed in analyzing their vulnerabilities from many side channel attacks.

In this section, we introduce two publications which tried to attack lattice-based problem or cryptographic primitives by using power and faults attacks.

3.1 Simple Power Attack on Ring-LWE

Park *et al.*[9] proposed an idea to execute Simple Power Analysis (SPA) attack to break a public-key cryptosystem [10] based on Ring-LWE problem.

This attack is assumed that the operations for public key encryption and decryption were implemented in 8-bit micro-controller with limited computing power and memory such as Internet–of–Things(IoT) devices.

Simple Power Analysis(SPA) attack initiated by Kocher *et al.*[11] in 1999 needs to sample visually interpreting power traces, or graphs of electrical activity over time. Differential Power Analysis(DPA) attack is a more advanced form of analysis which can allow an attacker to compute the intermediate values within cryptographic computations by statistically analyzing data collected from multiple cryptographic operations.

Ring-LWE cryptosystem is divided into three polynomial –time algorithms, namely, key generation(), encryption(), and decryption(). In encryption(), Number Theoretic Transform(NTT) algorithm is applied in error polynomials e_1, e_2, and e_3, and an original message m. In decryption, Inverse NTT(INTT) is applied by of key and a ciphertext to recover m. NTT is very useful to Ring-LWE cryptosystems based for their implementation. INTT contains modulus calculus (modulus q which is security parameter can have 7,681 and 12,289 for 128-bit and 256-bit level security, respectively.). We can narrow down the range of each coefficients of a secret key by just checking if modulus computation is executed or not. And this difference can be identified by comparing power consumption. After $\lceil \log_2 q \rceil$ iteration are executed, all of coefficients of a secret key, that is an entire secret key, can be revealed.

Their attack is claimed to be a kind of chosen ciphertext attack which is to exploit a vulnerability on decryption operation when Ring-LWE cryptosystem using NTT is implemented in 8-bit microcontroller.

Algorithm 1 describes the brief decrypting operation in LPR[10] before decoding. Note that **step 8** in **Algorithm 1** is a conditional check. If $m[i] \geq q$, an additional modulus operation is required. From this operational differences, they claimed an idea to extract a secret key r with SPA attack.

Algorithm 1: Decrypting operation in LPR

Input : Ciphertext c_1, c_2, Secret Key r
Output: Encoded plaintext m

1 **for** i=*0 to* $n-1$ **do**
2 $\quad c_1[i] \leftarrow r[i] \cdot c_1[i]$
3 $\quad$ **if** $c_1[i] \geq q$ **then**
4 $\quad\quad c_1[i] \leftarrow c_2[i] \mod q$
5 $\quad$ **else**
6 $\quad\quad m[i] \leftarrow c_1[i] + c_2[i]$
7 $\quad$ **end**
8 $\quad$ **if** $m[i] \geq q$ **then**
9 $\quad\quad m[i] \leftarrow m[i] \mod q$
10 $\quad$ **end**
11 **end**
12 return m;

Figure 1 shows difference of power traces depending on modulus operation or not. The solid and dotted curves in Figure 1 represent the power trace of executing and non-executing modulo operation, respectively.

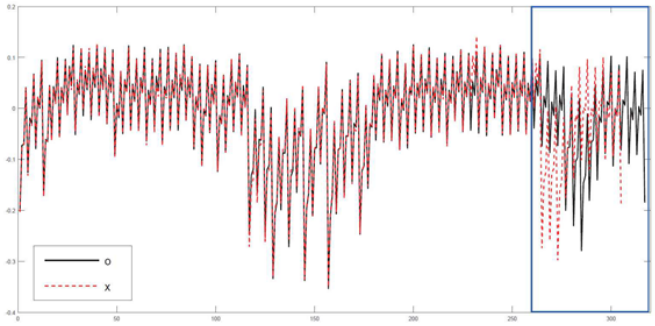

Figure 1: Difference of power traces

3.2 Fault Attack on Lattice-based Signatures

Bindel *et al.* [12] investigated the vulnerability and resistance of three lattice-based signature schemes(BLISS [13], Ring-TESLA[14] and GLP[15]) from three kinds of fault attacks such as randomizing, skipping, and zeroing faults.

They explored this vulnerabilities and suggest their countermeasures during the key generation, digital signing, and verification algorithms.

(i) A randomization fault attack changes the value of a target variable in sub-algorithm randomly. Though attackers do not know the value of a target variable, attackers can know that it was changed within a certain range after fault attack. The following is a randomization attack of the secret polynomial. The signature of a message m is $\sigma = f(z, c)$, where $z = s \cdot c + y$, y is random and c contains hashed value of m with some public informations. If one coefficient of the private key(s) is changed, $\sigma' = f(z', c)$, where z' is a new value changed by coefficient of s by fault attack. The difference between the original hashed value and the faulted hashed value can reveal the coefficient of private key s.

(ii) Skipping fault attack is to skip or ignore the selected lines of the program code. For example, during key generation in GLP, the public key is computed as follows : first a and s are multiplied and saved in the value b ($b = as$). Afterwards, the error e is added to b ($b = b + e$). Hence, skipping the second operation yields $b = as$ and an attacker can easily recover secret key s by Gaussian reduction.

(iii) Zeroing fault attack assumes that attackers can set a whole or a part of a variable. For example, attackers can zerorize all the coefficients of the secret polynomial. If this zeroing is done during key generation, it induce $s = 0$, hence the value $b = e \mod q$ is returned and attackers can understand the error polynomial e. Only knowing e can generate forged signatures in GLP without private information which is a kind of universal forgery of a signature scheme.

Their research showed that all examined signature schemes and their implementations are vulnerable to all three kinds of considered fault attacks. Bindel *et al.* [12] summarized the comparison of three scheme with respect to their vulnerability as shown in Table 1. They also suggested some countermeasures of each attack, emphasizing that we have to be pre–cautious against all three kinds of attacks, not only one.

Table 1: Comparison of three schemes with respect to their vulnerability to three attacks

Fault Attack	Algorithm	GLP	BLISS	Ring-TESLA
Rand. of secret	S	•	•	○
Rand. of error	S	○	○	○
Rand. of modulus	S	○	○	○
Rand. of randomness	S	○	○	○
Skip of mode-reduction	KG	○	-	○
Skip of addition	KG	•	•	•
Skip of rejection	S	(•)	(•)	(•)
Skip of addition	S	•	○	○
Skip of mod-reduction	S	○	-	○
Skip of correct-check	V	•	•	•
Skip of size-check	V	•	•	○
Zero. of secret	KG	•	-	○
Zero. of randomness	S	•	•	•
Zero. of hash value	S	○	○	○
Zero. of hash polynomial	V	•	•	•

Rand.: randomization, Skip.: skipping, Zero.: zeroing.
Key generation(KG), Signature generation(S), and verify(V)
•: vulnerable, ○: not vulnerable, -: not applicable
(•): vulnerable with a huge number of needed faults

4 Lattice-based Cryptographic Library

In this section, we introduce four lattice-based cryptographic libraries such as HElib, FHEW, $\Lambda \circ \lambda$, and LatticeCrypto.

4.1 HElib Library

HElib library[16] is written in C++ and uses the NTL library[17] and GMP library[18]. It implements HE, specifically the BGV encryption scheme[8].

This library has two levels. In the lower level, HElib is the cryptosystem itself that means the hardware platform of homomorphic computations. The platform defines a set of operations that can be applied homomorphically and specify their evaluation cost. The higher level contains the algorithm over the platform *e.g.,* routing algorithms, simple linear functions, and polynomial evaluation. This algorithms are necessary for bootstrapping.

HElib library is divided into two layers (Math layer and Crypto layer). Figure 2 shows a block diagram of the HElib library. In Math layer, bluestein and CModulus class are used to compute polynomials in FFT using NTL library whereas PAlgebra and PAlgebraMod class are for encoding and decoding operations. The main part of Math layer is DoubleCRT class that manipulates polynomials in Double-CRT representation.

The Crypto layer implements actual BGV homomorphic cryptosystem. This layer can also be partitioned into the Ctxt module implements ciphertext and homomorphic operations. FHE module implements the keys and key-switching matrices. KeySwitching module implements some strategies for deciding how key-switching matrices generate.

Since Gentry proposed FHE with bootstrapping, drastic changes were made in FHE scheme. Gentry used bootstrapping procedure to reset noise of ciphertexts thus keeping on computing encrypted data. Though many different FHE was suggesed, their common concept still prefers to using bootstrapping. Recent bootstrapping implementation of FHE is HElib library of Halevi and Shoup which requires about 6 - 30 min.

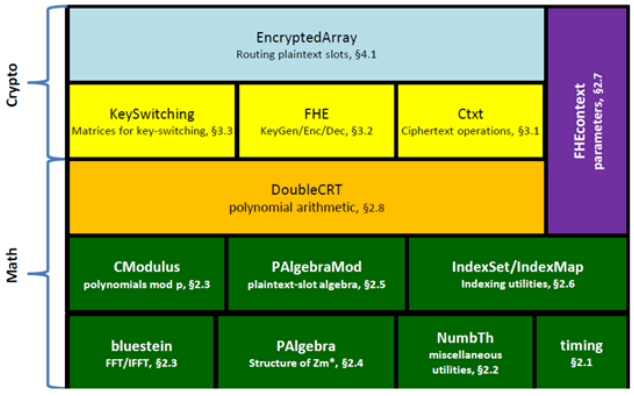

Figure 2: A block diagram of the HElib library

4.2 FHEW Library

FHEW library[19] is introduced by Ducas and Micciancio in 2015. This library uses C and C++ language and has less than 600 lines which are much simpler compared to other libraries (HElib library is about 20,000 lines). And this library uses the Fastest Fourier Transform in the West (FFTW library) to compute FFT for faster computation.

FHEW library reduces the time for bootstrapping procedure dramatically, about 0.6 sec (note that HElib library takes 6 min at least and even though this was much better than previous implementations at that time). For efficient HE, they developed two new techniques:

(i) A new, cheap NAND gate.

It starts from LWE encryption with message space $\mathbb{Z}_t (t > 2)$. And the main idea is to use different message space, for input space $\mathbb{Z}_4$ and output space $\mathbb{Z}_2$. This advantage is that the cost of computing Homomorphic NAND is negligible (similar to a single private key cryptographic operation). Also, noise grows only by a small constant factor and simplifies the task done by the Refreshing procedure which will be described later.

In LWE encryption, a set of encryptions of m with an error $e < E$ is denoted by $LWE_s(m, E)$. And FHE bootstrapping requires strong Refreshing like Eq. (2).

$$LWE_s(m, e) \rightarrow LWE_s(m, e'), e' \ll e. \qquad (2)$$

By using new homomorphic NAND gate, new refresh function LightRefresh is suggested as Eq. (3).

$$LWE_s^2(m, q/4) \rightarrow LWE_s^4(m, q/16) \qquad (3)$$

where the previous refresh function(bootstrapping) is Eq. (4).

$$LWE_s^2(m, q/4) \rightarrow LWE_s^2(m, E), E \ll q \qquad (4)$$

Therefore it relaxes the requirement on the Refreshing procedure, potentially making the overall scheme faster.

(ii) A simpler Refreshing procedure using a ring structure.

This is based on the general approach by Alperin-Sheriff and Peikert [20] and a ring variant of HE by Gentry *et al.* [21]. And their main idea is to implement arithmetic mod q in the exponent. Using a ring structure in encryption operation allows that each cyclic group element is encoded by a single ciphertext, rather than a vector of

Table 2: Comparison the lattice-based cryptographic libraries

Library	HElib	FHEW	$\Lambda \circ \lambda$	LatticeCrypto
Publication	2013	2014	2015	2016
Based Library	NTL, GMP	FFTW3	None	None
Language	C and C++	C and C++	Haskell and C++	C and Assembly
Error Sampling	Discrete Gaussian Distribution	Discrete SubGaussian Distribution	Continuous Gaussian Distribution	Discrete Gaussian Distribution
Platform	Linux	Linux	Windows, Linux	Windows, Linux
Hard Problem	Ring-LWE	Ring-LWE, LWE	Ring-LWE	Ring-LWE

ciphertexts. In general we can get generic $\tilde{\Omega}(n)$ speed-up from ring structure. From this idea, they claimed that an additional $\tilde{\Omega}(\log_3 q)$ speed-up by embedding can be achevied. And error after bootstrapping can be reduced by $O(\sqrt{n}\log n)$.

Using two techniques above, both homomorphic NAND and refreshing operations take 0.61 sec on a single standard 64-bit core running at 3GHz.

4.3 $\Lambda \circ \lambda$ Library

$\Lambda \circ \lambda$ library[22] is written in Haskell and C ++ language, and most of the code is written in Haskell which is a general-purpose purely functional programming language with the advantage that the length of the code is relatively short. The number of entire source code line is 4,991 lines, which means that the source code is about 7 times shorter than HElib library.

This library consists of four layers[22] in total, namely, Integer layer, Tensor layer, Cyclotomic layer, and Cryptography layer. Integer and Tensor layers implement the necessary computation function to increase the computation speed in the encryption and decryption operations. Cyclotomic layer implements Ring-LWE-based lattice cryptography and Gaussian error sampling algorithm. Cryptography layer implements the HE. This includes GSW and BGV used in HElib and FHEW libraries, and additionally implements GHS[23].

$\Lambda \circ \lambda$ library uses a continuous Gaussian distribution in the error sampling algorithm. In general, the discrete Gaussian distribution is used in the error sampling process of the Ring-LWE scheme.

4.4 LatticeCrypto Library

Microsoft's LatticeCrypto library[24] does not need NTL or GMP libraries on other libraries, and is implemented using C and assembly language. Hence, this library works well in Windows and also in Linux platform. However, this library designs for only key exchange protocol. Therefore, this library cannot be applied to various cryptographic primitives such as encryption or decryption and also digital signature. Compared to the NewHope[25] key exchange protocol, the count of computation is reduced by about 1.4 times[24]. This is because NTT reduces the computational complexity by using a new modular reduction method.

Table 2 shows in brief comparison of the lattice-based cryptographic libraries.

5 Timing and Fault Attacks

In this section, we perform timing and fault attacks of the FHEW and HElib libraries among four libraries examined in Section 4. This is because the $\Lambda \circ \lambda$ library is written in Haskell. We excluded this library from the experiment because we cannot establish the same environment for our experiment. Also, LatticeCrypto library is excluded from our experiment due to its limited operation such as key exchange protocol only.

5.1 Experimental Setup

The experimental environment is as follows: Intel(R) CPU i7-5500, RAM 16GB and test on Ubuntu v16.04. The compiler also uses gcc v5.4.0. We download reference FHEW and HElib libraries in GitHub.

5.2 Experimental Result

5.2.1 FHEW Library

First, we measure the time in the reference FHEW library. Second, we measure the time while changing the value of the Gaussian distribution to 0 or 1. Finally, we measure the time by changing the value of a random polynomial's each coefficient from minimum value 0 or maximum value $2^9 - 1$. Also, we test the different key size such as $n = 500$ in LWE scheme and $N = 1,024$ in Ring-LWE scheme. FHEW library is implemented only single bit encryption. We experiment with two kinds of secret keys such as LWE and Ring-LWE.

Therefore, we first fix the length of the message, and adjust the parameters to check how to affect the execution time. Table 3 shows the result in FHEW library timing measurement. Since the encryption time is flat to our experimental setup, we operate 1,000,000 encryptions to each set and add all the execution times. In Table 3, there is no significant time difference obtained even when the sampling value of the Gaussian distribution is fixed and changing $a[i]$. The time difference is not clearly identified when the experiment is performed by modify the values of $a[i]$.

Therefore, we try the time difference according to the message length. Because FHEW library supports single bit operations, we should call the encryption function multiple times if the message length increases. Since the maximum input size is 32-bit, we increase the message length to 32-bit. Figure 3 shows the time difference of FHEW library when message length is

Table 3: Timing measurement in FHEW library

Test Set	Original		GaussDist=0		GaussDist=1		$a[i]$=0		$a[i]$=2^9-1	
Plaintext Input (1-bit)	0	1	0	1	0	1	0	1	0	1
Summed Enc. time of (1) (Sec)	2.463	2.485	2.479	2.476	2.463	2.484	2.482	2.478	2.467	2.466
Summed Enc. time of (2) (Sec)	2.564	2.510	2.512	2.814	2.502	2.532	2.502	2.606	2.525	2.541

(1): Using LWE key, (2): Using Ring-LWE key
$a[i]$: random monic polynomial's each ith coefficient value

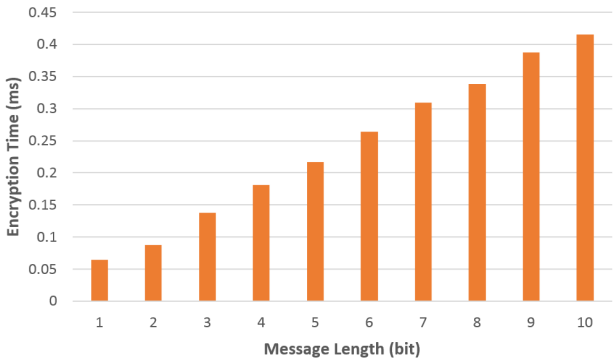

Figure 3: Time difference of FHEW (1 to 10-bit)

changed from 1-bit to 10-bit. In addition, Figure 4 shows the length of the message in 10-bit increments, measured up to the maximum input size of 32-bit. We find that the execution time increases linearly.

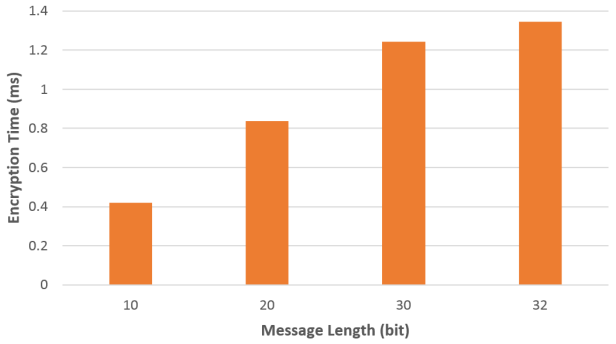

Figure 4: Encryption time of FHEW (10 to 32-bit)

5.2.2 HElib Library

In HElib library, we measure the time difference in evaluation operation by changing degrees of message polynomial. The experiment performs to measure the time by increasing the degree from 1 to 20. In HElib library, we input a random message. The parameter values are fixed in plaintext size $p^r = 3^2 = 9$, and circuit depth $L = 11$. Figure 5 shows the evaluation time difference according to degrees of message polynomial. Unlike FHEW library, we check that increasing the degree of message polynomial does not increase the time linearly.

However, this result cannot be applied to get the plaintext or secret key directly.

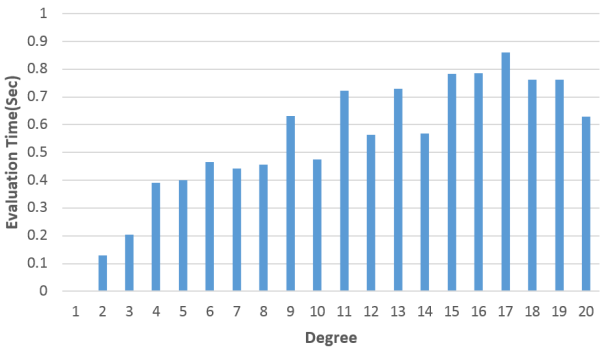

Figure 5: HElib timing result

5.3 Analysis of Time Traces

In general, if a secret key or plaintext is exposed from time traces, this can be generated by logical **if()** statement. In logical **if()** statement, we can easily check additional operations. Therefore, we can expose the secret key or plaintext. **Algorithm 2** does not have a logical **if()** statement in the encryption.

Algorithm 2: FHEW original Encrypt()

Input : const SecretKey sk, int Message m
Output: Ciphertext ct

$ct = (m \bmod 4) * q/4$+Sample(GaussDist.);
for $i < 500$ **do**
  $a[i]$=rand()mod q;
  return $ct = ct + (a[i] \cdot sk[i]) \bmod q$;
end
return Ciphertext ct;

If no logical **if()** statement, encryption performs the same operation on all secret keys and plaintext. However, in our experiment, the clock speed of Intel(R) i7 5500 CPU is 2.4GHz at normal state and 3.0GHz at turbo boost state. A single 500-bit multiplication operation is performed within about 0.87μs during the turbo boost state. For encryption operation of FHEW library, we multiply secret key and random polynomial. After this, the sum of the message and the sampling value adds from the previous value. Therefore, the message and the secret key are combined by multiplication.

We use gettimeofday() as time measurement function in Linux. This function can measure time up to ms. However, the actual operation time is less than ms, so we cannot confirm the timing difference experiment

that we tried. To summarize, if a logical **if()** statement exists and each operation is different, a timing attack can be made using the difference in operation time. However, FHEW and HElib libraries do not have a logical **if()** statement, so we cannot check the time differences in these two libraries.

5.4 Fault Attack

Three kinds of fault attacks is suggested by Section 3.2. We apply those fault attacks in FHEW and HElib libraries. Randomizing fault attack is occurred by specific discrete Gaussian distribution with bounded error, like GLP and BLISS. However, FHEW and HElib libraries use general discrete Gaussian distribution, like Ring-TESLA. We expect that randomizing fault attack will not be applied to FHEW and HElib libraries.

Skipping and zeroing fault attacks can be applied to FHEW and HElib libraries beacause these libraries are public open source, *e.g.,* skipping Sample(GaussDist.) in **Step 1**, **Algorithm 2** can reveal secret key using Gaussian reduction.

6 Making Constant Time Encryption

Here, we discuss a method for padding the message. In order to make a constant time encryption despite of arbitrary message length.

6.1 Message Padding

The time difference according to the message length of FHEW library can expose the length information of plaintext. Therefore, we propose message padding to prevent this problem. **Algorithm 2** describes the FHEW original Encrypt() code. As mentioned Section 5, the original code of FHEW library supports only single bit operation. Therefore, we propose message padding in order not to leak information about message length. **Algorithm 3** is the pseudo code of FHEW library with message padding. The maximum length of the message is limited to 32-bit, which is the maximum input size of FHEW library.

Algorithm 3: Message padding in FHEW

Input : const SecretKey sk, int Message m
Output: Ciphertext $ct[n]$

```
for i < 500 do
    m = m mod 10;
    c[i] = m;
     ct[i] = (c[i] mod 4)*q/4+Sample(GaussDist.);
    a[i]=rand()mod q
    return ct[i] = ct[i] + (a[i] · sk[i]) mod q ;
end
return Ciphertext ct[n];
```

6.2 Comparison with Original FHEW Library

We compare the computation time with the original FHEW library using the message padding in Section 6.1. Figure 6 shows comparison with original FHEW library and message padding. The encryption time increases linearly in the original FHEW library. However, after message padding, a constant computation time regardless of message length will take. Since we have the message padding for 32-bit, encryption time increases than before the padding.

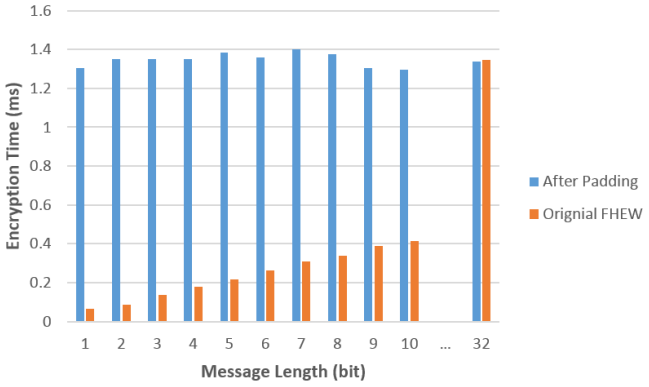

Figure 6: Comparison with Original FHEW Library

7 Concluding Remarks

In this paper, we compare four lattice-based cryptographic libraries. We also analyzed two libraries, HElib and FHEW libraries, in order to execute timing attack on FHEW and HElib libraries. Thus, we can modify the values of Gaussian distribution and a random polynomial in FHEW library but cannot get the definite time difference. However, FHEW library implements only single bit operation. We proposed message padding to prevent information leakage on message length. As a result, we confirm the experiment with constant time for the arbitrary message. We encounter some limitations. It is difficult to encrypt the variable message length. Also, there is a disadvantage that the length of the input message is limited to 32-bit when message padding is implemented.

Also, we analyze the result that there is no time difference. In General, the time difference in timing attack occurs in a logical **if()** statement. Since the logical **if()** statement does not use in these libraries, we found that there is no difference in operation time.

As future work, we will experiment a resource-constrained to make the time differences to be identified easily.

Acknowledgements

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. NRF-2015R1A2A2A01006812).

References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Pro-*

ceedings, 35th Annual Symposium on Foundations of Computer Science, FOCS 1994, pp. 124–134, IEEE, 1994.

[2] "FHEW library." https://github.com/lducas/FHEW/.

[3] "HElib library." https://github.com/shaih/HElib/.

[4] "$\Lambda \circ \lambda$ library." https://github.com/cpeikert/Lol/.

[5] "LatticeCrypto library." https://www.microsoft.com/en-us/research/project/lattice-cryptography-library/.

[6] X. Yi, R. Paulet, and E. Bertino, *Homomorphic encryption and applications*. Springer, 2014.

[7] C. Gentry, "A fully homomorphic encryption scheme," in *Ph.D Thesis, Stanford University*, 2009.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS 2012*, pp. 309–325, ACM, 2012.

[9] A. S. Park, Y. S. Won, and D. G. Han, "Chosen ciphertext SPA attack on Ring-LWE cryptosystem," in *Conference on Information Security and Cryptography - winter*, Yonsei Univ., 12. 03. 2016. (in Korean).

[10] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Proceedings, Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2010*, pp. 1–23, Springer, 2010.

[11] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology, CRYPTO 1999*, pp. 388–397, Springer, 1999.

[12] N. Bindel, J. Buchmann, and J. Kramer, "Lattice-based signature schemes and their sensitivity to fault attacks," in *Cryptology ePrint Archive, Report 2016/415*, 2016. http://eprint.iacr.org/2016/415.

[13] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky, "Lattice signatures and bimodal gaussians," in *Advances in Cryptology, CRYPTO 2013*, pp. 40–56, Springer, 2013.

[14] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer, and G. A. Marson, "An efficient lattice-based signature scheme with provably secure instantiation," in *International Conference on Cryptology in Africa, AFRICACRYPT 2016*, pp. 44–60, Springer, 2016.

[15] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," in *International Workshop on Cryptographic Hardware and Embedded Systems, CHES 2012*, pp. 530–547, Springer, 2012.

[16] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology, CRYPTO 2014*, pp. 554–571, Springer, 2014.

[17] "NTL library." http://www.shoup.net/ntl/.

[18] "GMP library." http://www.gmplib.org.

[19] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Proceedings, Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2015*, pp. 617–640, Springer, 2015.

[20] J. Alperin-Sheriff and C. Peikert, "Faster bootstrapping with polynomial error," in *Advances in Cryptology, CRYPTO 2014*, pp. 297–314, Springer, 2014.

[21] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Advances in Cryptology, CRYPTO 2013*, pp. 75–92, Springer, 2013.

[22] E. Crockett and C. Peikert, "$\Lambda \circ \lambda$: A functional library for lattice cryptography," in *Cryptology ePrint Archive, Report 2015/1134*, 2015. http://eprint.iacr.org/2015/1134.

[23] C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Proceedings, Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT 2012*, pp. 465–482, Springer, 2012.

[24] P. Longa and M. Naehrig, "Speeding up the number theoretic transform for faster ideal lattice-based cryptography," in *Cryptology ePrint Archive, Report 2016/504*, 2016. http://eprint.iacr.org/2016/504.

[25] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe, "Post-quantum key exchange - a new hope," in *Cryptology ePrint Archive, Report 2015/1092*, 2015. http://eprint.iacr.org/2015/1092.