Error Leakage using Timing Channel in FHE Ciphertexts from TFHE Library

Bhuvnesh Chaturvedi[1], Anirban Chakraborty[1], Ayantika Chatterjee[2], and Debdeep Mukhopadhyay[1]

[1] Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur
{bhuvneshchaturvedi2512, ch.anirban00727, debdeep.mukhopadhyay}@gmail.com
[2] Advanced Technology Development Centre, Indian Institute of Technology, Kharagpur
cayantika@gmail.com

Abstract. Timing attack is a class of side-channel attacks that aims to leak secret information based on the time it takes to perform different operations. The biggest advantage of a timing attack is that it does not require sophisticated or expensive equipment to be carried out. Side Channels on FHE schemes have been reported on the client side which has the secret key. But the present project aims to delve into the counter intuitive question, can an analysis be performed on the server end which ideally has no information of the secret key. In this report, we investigate when homomorphic operations are performed on the ciphertexts stored in the server, can timing reveal information of the error used to mask the ciphertexts? Finally, can this be utilized to determine the secret key of the ciphering technique?

Keywords: FHE · LWE · Timing Attack · Error Reduction · Post Quantum Cryptography

1 Introduction

The current scenario of data driven applications has given rise to the issue of storing and processing bulk amount of data in bounded time, so that the results can be used to gain competitive advantage. The advent of cloud computing has brought a revolution in this field, with individuals and organizations offloading their storage and processing requirements to third-party servers. One of the major drawbacks of doing so is the lack of trust of clients on the server provider to respect the confidentiality of their data. This lack of trust is due to both external attackers as well as malicious insiders who want to get hold of this data for their own personal gains.

An effective solution to this problem is to encrypt the data before offloading it to the cloud server. However any standard symmetric or asymmetric encryption schemes, like AES, RSA, ECC, etc. cannot be used to encrypt the data as it does not allow computation directly on encrypted data and it needs to be decrypted first. Moreover, these standard schemes will not be secure in a post quantum world as their security relies on underlying hard problems such as integer factorization, discrete logarithm and elliptic curve discrete logarithm problems all of which can be solved in polynomial time by a quantum computer running Shor's algorithm [7].

A plausible answer to the above question is based on the idea of Homomorphic Encryption or HE. A Homomorphic Encryption scheme can be defined as one which allows evaluation of arbitrary functions (or circuits) on encrypted data with the result in encrypted form as well. The security of these schemes are based on worst-case lattice-based hard problems such as Learning With Errors (LWE) [6] or its ring variant, Ring-LWE [5]. These schemes are based on the idea of noise, a small value that is added to the ciphertext. The problem with this introduction of noise is that it increases as computations are performed on ciphertext, either slowly (when additions are performed) or rapidly (when multiplications are performed). Once this noise crosses a pre-determined threshold level, decryption can no longer be performed as the original message becomes unrecoverable. Thus once the noise level reaches or is about to reach the threshold, a refreshing operation needs to be performed to reduce the level of noise back to an acceptable limit.

Gentry proposed a breakthrough result in 2009 [4] called Bootstrapping, where refreshing operation was performed homomorphically using the encryption of the secret key called the bootstrapping key. It resulted in encryption of the original message under a different key with a decreased noise level. This resulted in development of a new class of HE schemes called Fully Homomorphic Schemes or FHE as it allows evaluation of any arbitrary function with no limit on depth of the circuit used to implement the same. The efficiency of any FHE scheme depends on how efficiently bootstrapping can be performed. While the original approaches resulted in computation time in minutes, further improvement brought down this time to milliseconds.

The current fastest scheme in terms of bootstrapping is Torus FHE (TFHE) [2] introduced by Chillotti et. al. in 2016 which computes a homomorphic logic gate in single-core time of about 13 milliseconds. The CPU-based library [1] is open source and is currently being used in various open source projects.

We are looking into the possibility of recovering the error by performing a timing attack on the gate evaluation stage. The intuition behind the idea is that the time required to evaluate a gate is dependent on the complexity of the underlying operation, which in turn decides the level of noise in the computed ciphertext. While we are currently focusing on the TFHE library, similar attacks can be performed on any HE scheme based on the LWE problem.

2 Preliminaries and Notations

In this section, we introduce the idea of LWE problem. We begin with a brief introduction of Torus, the mathematical structure on which the TFHE scheme is based on. This is followed by the LWE ciphertext used to encrypt the plaintext message. Finally we show how the logic gates are performed homomorphically.

2.1 Torus

Torus is defined as a set of real numbers modulo 1, or real values lying between 0 and 1. It is denoted as $\mathbb{T} = \mathbb{R}/\mathbb{Z} = \mathbb{R} \mod 1$. This set $\mathbb{T}$ along with two operators, namely addition '+' and external product '$\cdot$', forms a $\mathbb{Z}$-module. It means that addition is defined over two torus elements which results in a torus element. Also external product is defined as a product between an integer and a torus element which results in a torus element. Product between two torus elements is not defined.

2.2 Learning With Error problem

Let $n \geq 1$ be an integer and $\mathbf{s}$ be a secret sampled uniformly from some set $\mathbf{S} \in \mathbb{Z}^n$. An LWE sample is denoted by a tuple $(\mathbf{a}, b) \in \mathbb{T}^n \times \mathbb{T}$, where $\mathbf{a} \in \mathbb{T}^n$ is chosen uniformly and $b = \mathbf{a} \cdot \mathbf{s} + e$. Here e is an error sampled uniformly from a Gaussian distribution with parameter $\alpha \in \mathbb{R}^+$. LWE problem has the following two variations:

- *Search problem*: having access to polynomially many LWE samples, find $\mathbf{s} \in \mathbf{S}$.
- *Decision problem*: distinguish between LWE samples and uniformly random samples from $\mathbb{T}^n \times \mathbb{T}$.

2.3 LWE Ciphertexts

In TFHE, the secret key is defined as a binary vector of length n, i.e. $\mathbf{s} \in \mathbb{B}^n$, where $\mathbb{B} \in \{0, 1\}$. The value of n depends on the required bit security. A random mask $\mathbf{a}'$ is generated as a vector of torus elements of length n, i.e. $\mathbf{a}' \in \mathbb{T}^n$. The pair $(\mathbf{a}', b)$, where $b' = \mathbf{a}' \cdot \mathbf{s} + e$, acts as a public key which can be used by anyone for encryption. Here $e \in \mathbb{T}$ is a random error sampled uniformly from a Gaussian distribution.

To encrypt a message $\mu \in \mathbb{T}$, a mask of all zeroes with length n, i.e., 0^n, is generated and the plaintext pair $(\mathbf{0}, \mu)$ is added to the public key $(\mathbf{a}', b)$ to obtain the final ciphertext message as $\mathbf{c} = (\mathbf{a}, b)$. Here $\mathbf{a} = \mathbf{a}' + 0^n$ and $b = b' + \mu$. The operations defined over LWE ciphertexts are ciphertext addition, which adds the underlying plaintext messages, and multiplication of ciphertext with an integer constant, which multiplies the underlying plaintext message with the constant. Multiplication between two LWE ciphertexts is not defined as it involves multiplication between torus elements which itself is not defined.

2.4 Gate evaluation

To evaluate a gate homomorphically, TFHE library performs multiplication of input ciphertext with a constant. It then adds (or subtracts) these new ciphertexts to (or from) the gate constant, which is defined for each particular gate. This results in the addition (or subtraction) of the random masks as well as the errors. For example, given two ciphertexts $c_1 = (a_1, b_1)$ and $c_2 = (a_2, b_2)$ where $b_1 = a_1 \cdot s + \mu_1 + e_1$ and $b_2 = a_2 \cdot s + \mu_2 + e_2$, their addition will result in a new ciphertext $c = (a, b)$ where $a = a_1 + a_2$ and $b = a_1 \cdot s + a_2 \cdot s + \mu_1 + \mu_2 + e_1 + e_2 = a \cdot s + \mu + e$. Based on the gate that is being evaluated, one can predict whether $\mu = 0$ or $\mu = 1$. For example, in case of AND gate $Pr[\mu = 0] = \frac{3}{4}$ and $Pr[\mu = 1] = \frac{1}{4}$ while in case of OR gate $Pr[\mu = 0] = \frac{1}{4}$ and $Pr[\mu = 1] = \frac{3}{4}$. Once the message is removed, we will be left with a set of LWE samples. If we can somehow remove the errors from these LWE samples, we will be left with a system of equation of type $b = a \cdot s$. Solving this system of equations will reveal the secret key s.

3 Motivation

The security of all cryptographic schemes based on the LWE problem relies upon the error that is added to the ciphertext. If this error can be removed, then the system of equations reduces from $\mathbf{A} \cdot \mathbf{s} \approx \mathbf{b}$ to $\mathbf{A} \cdot \mathbf{s} = \mathbf{b}$, which is then trivial to solve. Here $\mathbf{A}$ is a matrix of dimension $m \times n$ of known coefficients that are part of the ciphertext, and $\mathbf{b}$ is a vector of length m containing encryptions of underlying plaintext messages. The motivation of our research is to reduce the range of possible errors by observing the timing values of the gate operations generating them. Once the range is small enough, the system of equations can be solved by brute forcing all the possible error values to recover the secret key.

4 Methodology

Our method involves generating a template with a chosen plaintext pair and secret key. We generated $100,000$ ciphertext pairs of the same plaintext pair using the same secret key. Once obtained, we executed all the ten homomorphic gates on each ciphertext pair and recorded the time it took to evaluate the result along with the final error of the ciphertext. We computed these gates 20 times for each ciphertext pairs, which resulted in different timing values but the same final error value. We averaged out these 20 timing values to obtain the final value for our template. This was done just to smooth out any noise in the timing values.

Since we already knew the values of μ_1 and μ_2, we could easily infer the value of μ based on the logic gate being evaluated. Also, since we knew the value of the secret key and the value of $\mathbf{a}$ from the final ciphertext, we were able to compute the final error value as $e = b - \mathbf{a} \cdot \mathbf{s} - \mu$. Once the timing and error value pairs were obtained for each ciphertext pair, we clustered the error values based on their corresponding timing values. For example, all error values were

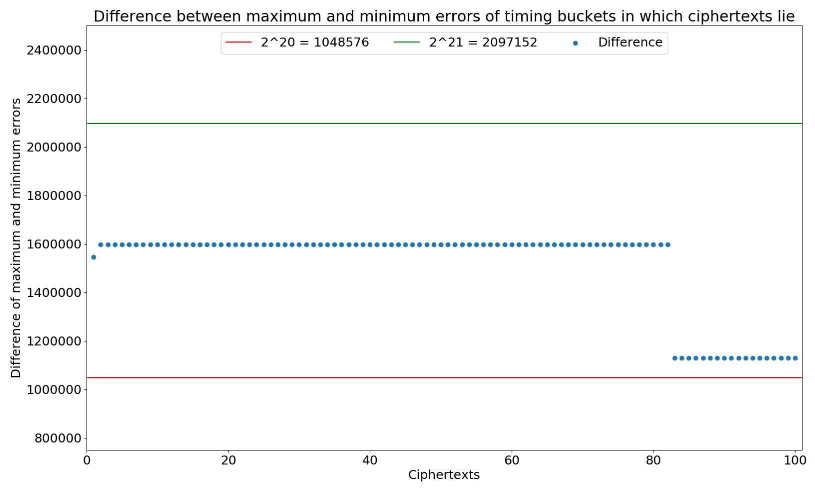

Fig. 1. Plot of difference between maximum and minimum error values for the timing buckets corresponding to 100 different ciphertexts

clustered into one bucket whose timing values lied in the range t_1 to t_2, where $t = t_2 - t_1$ denotes the bucket size. These clusters form our template based on which we tried to estimate the error values of new ciphertext results computed from some other ciphertext pairs.

The error estimation was performed by noting the timing value of the known gate evaluation of new ciphertext pairs and then finding out the cluster in which this timing value lies. Since we already know the error values that lies in this bucket, we can estimate the new value by exhaustively considering all error values belonging to this cluster or which are nearest to them.

To reduce the range of possible errors, we made some observations. First we tried to identify pairs of gates which followed a similar pattern in terms of range of errors. One such pair we observed was that of ANDNY and ORYN gates. The first observation we made was that the difference between the highest and the lowest errors for these gates were no more than 2^{21}. In other words, even if we have to brute force all the possible errors, then for these two gates we only need to check for 2^{21} combinations instead of 2^{32}. This reduced our search space by a factor of 2^{11}. Another observation that we made was that the timing ranges of these gates were same and the errors in these buckets also followed similar ranges.

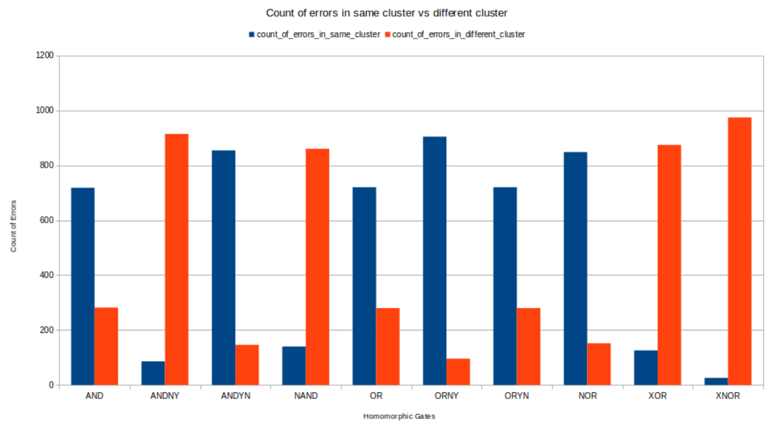

Fig. 2. Plot of gates with count of occurrence of error in same bucket vs different bucket. The size of buckets is 400. Key used to generate these ciphertexts is same as the one used to generate ciphertexts for template data.

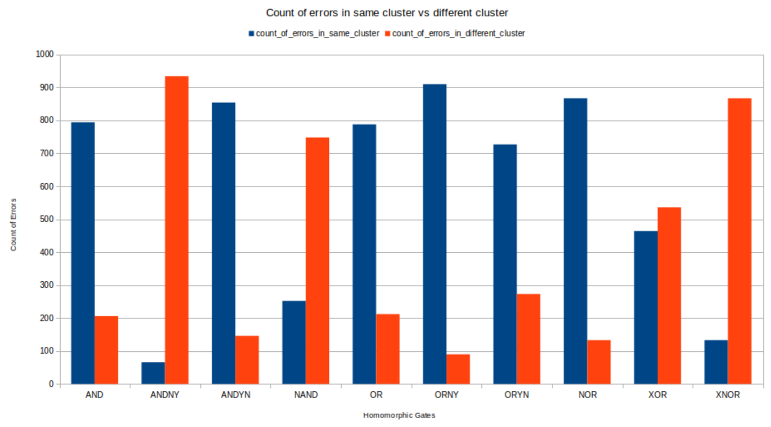

Fig. 3. Plot of gates with count of occurrence of error in same bucket vs different bucket. The size of buckets is 400. Key used to generate these ciphertexts is different from the one used to generate ciphertexts for template data.

5 Experiment and Results

Once the template was generated as described in the above section, we generated two sets of 1000 fresh ciphertexts of the same plaintext message pair using the same and a different, independent secret key. Once obtained, we ran all the 10 homomorphic gates on the new ciphertext pairs and recorded their timing and error values. Next we checked whether the corresponding error value lies in the cluster of its corresponding timing value or some other cluster. This was performed by comparing the error value with values in all clusters, finding the error value which is closest to this value, and then checking that in which cluster this value lies in. If it is the same cluster defined by its timing value, then we say that error has occurred in same bucket, otherwise not.

Table 1 shows the template of 10 ciphertexts, obtained using a different key, computed using ORYN gate. This is to simulate our attack where template is generated using known key and is being matched for unknown key. From the table we can observe that for this particular gate, the size of final range is lowered from 2^{32} to 2^{21}.

Sl. No.	**Error**	**Timing**	**Bucket**	**Min Error**	**Max Error**	**Difference**	**Bits**
1	1073896025	3570	3200 to 3599	1072956736	1074502919	1546183	21
2	1073935127	3716	3600 to 3999	1072988505	1074586560	1598055	21
3	1073967179	3762	3600 to 3999	1072988505	1074586560	1598055	21
4	1073716585	3804	3600 to 3999	1072988505	1074586560	1598055	21
5	1073771144	3836	3600 to 3999	1072988505	1074586560	1598055	21
6	1073663589	3866	3600 to 3999	1072988505	1074586560	1598055	21
7	1074081963	3899	3600 to 3999	1072988505	1074586560	1598055	21
8	1073889558	3938	3600 to 3999	1072988505	1074586560	1598055	21
9	1073756479	3989	3600 to 3999	1072988505	1074586560	1598055	21
10	1073625776	4077	4000 tot 4399	1073206314	1074336274	1129960	21

Table 1. Shows error of 10 ciphertexts along with the range of errors in their corresponding bucket and the number of bits required to represent this range

Figure 1 shows the difference between maximum and minimum error values for the timing buckets corresponding to 100 different ciphertexts. These values lies between $2^{20} = 1048576$, represented by the horizontal red line, to $2^{21} = 2097152$, represented by the horizontal green line. From table 1 and figure 1 we can observe that for this particular gate, the size of final range is lowered from 2^{32} to 2^{21}.

Figure 2 shows the count of occurrence in same bucket and in different for all the ten gates for bucket size of 400 when the key used to generate these ciphertexts is same as the one used to generate ciphertexts for template data. Figure 3 shows the count of occurrence in same bucket and in different for all the ten gates for bucket size of 400 when the key used to generate these ciphertexts is different and independent from the one used to generate ciphertexts for template data. In the figures, the blue bar represents the number of ciphertexts out of 1000 whose error lies in the correct bucket defined by the timing value of the corresponding gate operation. The orange bar represents the number of ciphertexts out of 1000 whose error lies in some other bucket. From the two figures, we can observe that the count of occurrence of error in same bucket is highest for AND gate while it is lowest for ANDNY gate in both the cases.

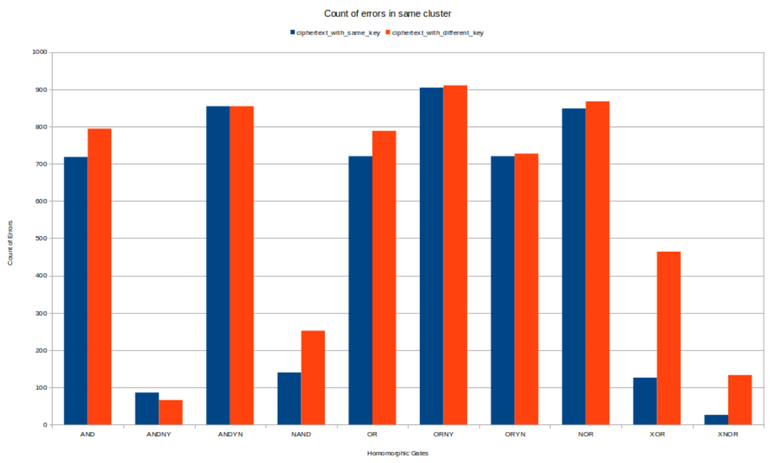

Fig. 4. Plot of gates with count of occurrence of error in same bucket two different sets of ciphertexts, one using same key and the other using different key. The size of buckets is 400.

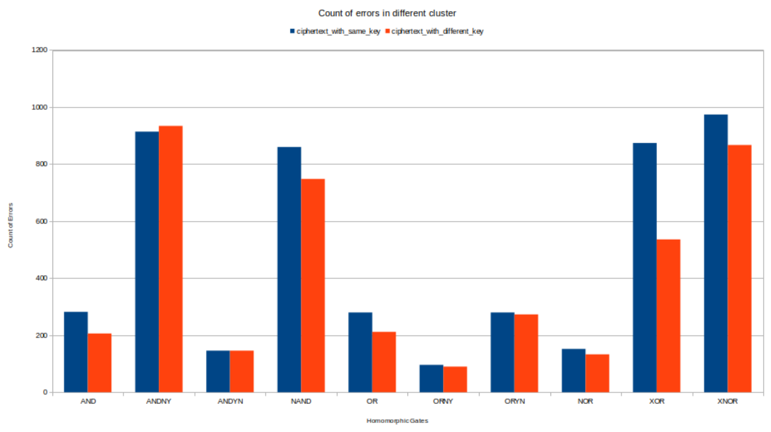

Fig. 5. Plot of gates with count of occurrence of error in different bucket two different sets of ciphertexts, one using same key and the other using different key. The size of buckets is 400.

6 Key Observation

Figure 4 shows the count of occurrence in same buckets for all the ten gates for the two sets of ciphertexts. Similarly figure 5 shows the count of occurrence in different buckets for all the ten gates for the two sets of ciphertexts. from the two figures we can observe that, except for XOR and XNOR, all the gates have similar results for the two sets of ciphertexts in both the cases.

The reason for the above observation is that the errors are generated independently of the secret key. Also while the secret key is generated only once, the errors are generated for each encryption operation. This observation provides a strong support to our template based attack for error recovery. The template can be created using any random key and it will work for other keys as well. Another thing to note from this argument is that creating multiple templates using different keys will not provide much benefits as the results will be similar for each template.

We also observed that for the two gates, i.e. ANDNY and ORYN, the range of errors after gate computation is remaining same even if the error distribution of the input ciphertexts is changing. In other words, when we changed the standard deviation of the error distribution from 2^{-15} as defined in TFHE library for $\lambda = 128-$bit security to some other value, say 2^{-30}, while the error distribution for the ciphertexts generated after LWE encryption changed, the range of errors after computation of the above two gates based on such ciphertexts remained same. This was not the case for other gates. Thus we can focus on these two gates only and leave out the rest.

7 Conclusion and Future Directions

In this report, we have shown a novel timing analysis technique of estimating the error value of the final ciphertext obtained after computation of a homomorphic gate and before the bootstrapping procedure. The proposed attack model takes advantage of the timing channel to infer about the possible range in which the error of the final ciphertext after gate computation may lie in. Since our scheme does not rely on the bootstrapping procedure, this method can also be extended to target LHE schemes where bootstrapping is not used.

An obvious future direction could be to figure out whether this reduction in error ranges can be exploited to recover the secret key. Dachman-Soled *et. al.* [3] proposed an analysis based on side channel "hints". While the authors have proposed four types of hints, this reduction in error range does not directly fall under those categories. So this can be an interesting future direction in the theoretical sense to look for possible methods of solving LWE problem with reduced error ranges.

References

1. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library (August 2016), https://tfhe.github.io/tfhe/
2. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: ASIACRYPT (1). pp. 3–33. Springer (2016). https://doi.org/10.1007/978-3-662-53887-6˙1, https://eprint.iacr.org/2016/870
3. Dachman-Soled, D., Ducas, L., Gong, H., Rossi, M.: Lwe with side information: Attacks and concrete security estimation. In: Advances in Cryptology – CRYPTO

2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II. p. 329–358. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-56880-1“˙12, https://doi.org/10.1007/978-3-030-56880-1_12
4. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing. p. 169–178. STOC '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1536414.1536440, https://doi.org/10.1145/1536414.1536440
5. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. pp. 1–23. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
6. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing. p. 84–93. STOC '05, Association for Computing Machinery, New York, NY, USA (2005). https://doi.org/10.1145/1060590.1060603, https://doi.org/10.1145/1060590.1060603
7. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Review **41**(2), 303–332 (1999), http://www.jstor.org/stable/2653075