Exposing Side-Channel Leakage of SEAL Homomorphic Encryption Library

Furkan Aydin
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC, USA
faydn@ncsu.edu

Aydin Aysu
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC, USA
aaysu@ncsu.edu

ABSTRACT

This paper reveals a new side-channel leakage of Microsoft SEAL homomorphic encryption library. The proposed attack exploits the leakage of ternary value assignments made during the Number Theoretic Transform (NTT) sub-routine. Notably, the attack can steal the secret key coefficients from a *single* power/electromagnetic measurement trace. To achieve high accuracy with a *single-trace*, we build a novel machine-learning based side-channel profiler. Moreover, we implement a defense based on random delay insertion based defense mechanism to mitigate the shown leakage. The results on an ARM Cortex-M4F processor show that our attack extracts secret key coefficients with 98.3% accuracy and random delay insertion defense does not reduce the success rate of our attack.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures** .

KEYWORDS

Homomorphic encryption; SEAL; number theoretic transform; side-channel attacks; machine learning

ACM Reference Format:
Furkan Aydin and Aydin Aysu. 2022. Exposing Side-Channel Leakage of SEAL Homomorphic Encryption Library. In *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security (ASHES '22), November 11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3560834.3563833

1 INTRODUCTION

Fully homomorphic encryption (FHE) allows arbitrary computations on encrypted message without the need for decryption [13]. FHE is useful, e.g., for cloud computing where the untrusted cloud can compute on encrypted data and the user, who holds the secret key, can decrypt the returned result. Therefore, HE preserves the privacy and confidentiality of data while allowing computations in untrusted environments. Although FHE is an evolving approach with mathematically provable security guarantees, their physical implementations can have vulnerabilities. For example, the first successful side-channel attack on FHE [2] has recently been demonstrated, revealing the encrypted message by exploiting the side-channel leakage of Gaussian sampling operations.

In this work, we reveal a new side-channel vulnerability of Microsoft SEAL—an FHE software library [28]. SEAL has recently gained significant recognition in the literature and has been used in many applications [3, 10, 20]. Our attack focuses on the Number Theoretic Transform (NTT) function of SEAL executed during the key generation. We first show that the NTT processes ternary values (-1, 0, or +1) that correspond to the secret key coefficients. Then, we build a side-channel attack that can extract this information from NTT operations. The challenge in attacking this stage is being limited to a *single-trace* measurement. We address this challenge by developing a multi-stage neural network based side-channel classifier. Finally, we implement a defense based on random delay insertion for the NTT and assess its effectiveness against our *single-trace* attack.

Our work is different from earlier *single-trace* side-channel attacks on the NTT [10, 18, 23, 25]. The timing leakage analysis by Drucker *et al.* achieve a low success rate of 9% [10] because the attack only focuses on branch executions of butterfly in NTT. This attack is inapplicable to SEAL because its butterfly unit is constant-time. Kim *et al.* propose an ML-based side-channel attack on NTT [18]. The attack exploits Montgomery reduction operation that does not exist in SEAL's NTT. Primas *et al.* abuse timing side-channel leakage from `DIV` instruction used to perform modular reduction—this vulnerability is also absent in SEAL [25]. Pessl *et al.* improve the attack of Primas *et al.* [23]. This attack may target constant-time NTT implementations as in SEAL but scales inefficiently for large polynomials used in FHE. Our proposed attack is simpler and more efficient compared to this attack because it specifically targets ternary value assignments.

The proposed attack is also different from the earlier *single-trace* analysis of FHE [2] because the earlier attack focuses on an operation that is replaced in SEAL v3.6. By contrast, our target is another operation and it is shown on the latest version of SEAL to date (v4.0). Moreover, our *single-trace* attack is fundamentally different from multi-trace attacks, which can target FHE's decryption operations. We do not address such attacks on decryption in this study since they are relatively straightforward extensions of the recent multi-trace analysis of lattice-based cryptography [27, 29]. *Single-trace* attacks are known to break defenses such as masking that are built for such multi-trace attacks [25].

A summary of our contributions is as follows.

ASHES '22, November 11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9884-8/22/11…$15.00
https://doi.org/10.1145/3560834.3563833

- We reveal a new *single-trace* side-channel leakage of SEAL. We show the processing in NTT function leaks information about the ternary values that can lead to recovering the secret keys in FHE. This vulnerability exists in the latest version (v4.0) of SEAL as of date.
- To effectively extract the side-channel information from a *single-trace* measurement, we propose a two-stage neural network based side-channel profiler. We use two distinct ML classifiers and ensemble results by multiplying guessing scores to improve the guessing success.
- We perform the proposed attack on the ARM Cortex-M4F running SEAL software. The results show that our proposed attack extracts each secret key coefficient with 98.3% accuracy.
- We evaluate random delay insertion countermeasure. We show that random delay insertion defense is susceptible to attacks.

The remainder of the paper is organized as follows. Section 2 provides the background information about FHE, NTT, and our threat model. Section 3 then introduces the proposed ML-based side-channel attack. Section 4 evaluates attack results and countermeasures. Subsequently, Section 5 discusses drawbacks of our attack. Finally, Section 6 concludes the work.

2 PRELIMINARIES

This section provides background information about the FHE, NTT, and threat model.

2.1 Fully Homomorphic Encryption (FHE)

FHE schemes are characterized by four primary functions: key generation, encryption, evaluation, and decryption. The key generation generates secret, public, and evaluation keys. The encryption uses the public key to encrypt user's message. The evaluation takes the encrypted message and the evaluation key to perform homomorphic operations over the encrypted message. The encrypted messages can be processed by others who do not know the secret key. Decryption takes the secret key and evaluation output to recover the message.

There are various software and hardware implementations of FHE such as SEAL [28], SEAL-Embedded [22], HElib [14], HEAAN [6], and PALISADE [24]. We specifically focus on SEAL which is compatible with SEAL-Embedded—the first FHE library targeted for embedded devices. While SEAL can support BFV [11] and CKKS [6] schemes of FHE, SEAL-Embedded only supports the CKKS scheme. In this work, our target scheme is CKKS since it is supported by both SEAL and SEAL-Embedded libraries.

CKKS scheme of FHE is constructed based on the Ring Learning with Errors (RLWE) problem [6]. An RLWE sample $b = as+e$ is built by sampling a from R_q (which is the residue ring of R modulo q), noise e sampling over R, and secret key s is chosen from a key distribution over R. In SEAL's CKKS scheme, R_3 is used as secret key distribution. In other words, the secret key is produced from a ternary distribution sampling over $\{-1, 0, 1\}^n$ where modulus n is to be the power of two. This generated secret key is then converted to the NTT domain before performing decryption operations.

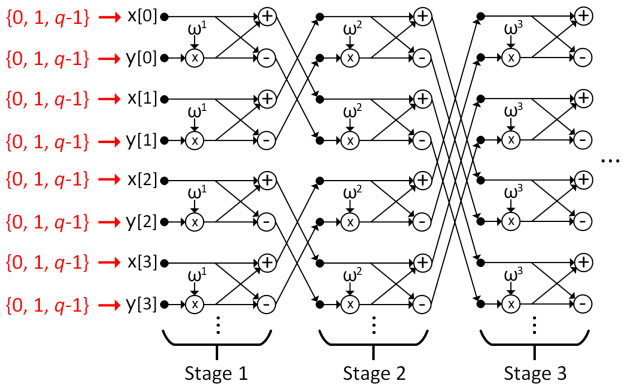

Figure 1: The first few stages of the NTT. Each stage of NTT consists of multiple butterfly operations. Twiddle factors ω are constant in each stage. NTT inputs of SEAL's CKKS scheme can be 0, 1, or q-1.

SEAL and SEAL-Embedded have several parameter settings [22, 28]. In this paper, we have targeted 128-bit security level and n = 4096 which is the default setting of SEAL-Embedded.

2.2 Number Theoretic Transform (NTT)

NTT is basically a form of Fast Fourier Transform (FFT) over finite field. It is used to improve the performance of polynomial multiplication. Its representation is denoted as $\hat{x} = \mathrm{NTT}(x) \in \mathbb{Z}_q^n$ where $x = (x_0,...,x_{n-1}) \in R_q$ denotes vectors of polynomials over R_q. Its formulation is $\hat{x}_i = \sum_{j=0}^{n-1} x\, \omega^{ij}$ where ω is fixed n's primitive root of unity. The powers of ω are called twiddle factors. In our target library configuration, modulus degree n is 4096. SEAL has 4 prime modulus using n = 4096 for key generation. Prime modulus (q) is 109 (30 + 30 + 30 + 19) bits and its coefficient values are 0x3ED00001, 0x3ED30001, 0x3ED60001, and 0x66001.

Although secret key coefficients can be equal to -1, 0, or 1, these values are converted to {0, 1, q-1} form before NTT operations. SEAL use primes at most 30-bits; therefore, NTT inputs can be equal to 0, 1, or 0x3ED00000.

Fig.1 illustrates the first few stages of NTT. NTT consists of $\log_2 n$ stages. In each stage, there are butterfly operations that consist of modular multiplication, addition, and subtraction. SEAL uses the Harvey butterfly structure instead of Cooley-Tukey (CT) [7] and Gentleman-Sande (GS) [12].

2.3 Threat Model

This work presents an attack on the NTT operation of SEAL CKKS scheme's key generation to extract secret key coefficients which are invoked by NTT. Our threat model assumes that SEAL's key generation is performed by the victim's device and the adversary has access to this device. Therefore, the adversary can capture multiple power traces for building a profile of the leakage. We also assume that the adversary knows the executed SEAL code's version and its parameters. Therefore, the adversary can build ML models offline by configuring the device with different keys. During the attack, however, the adversary tries to extract the secret key

```
1  void transform_to_rev
2  (ValueType *values, int log_n,
3  const RootType *roots,
4  const ScalarType *scalar = nullptr) const{
5    size_t n = size_t(1) << log_n;
6    RootType r;
7    ValueType u, v;
8    ValueType *x = nullptr;
9    ValueType *y = nullptr;
10   std::size_t gap = n >> 1;
11   std::size_t m = 1;
12   ...
13   for (std::size_t i = 0; i < m; i++){
14     r = *++roots;
15     x = values + offset;
16     y = x + gap;
17     for (std::size_t j = 0; j < gap; j+=4){
18       u = arithmetic_.guard(*x);
19       v = arithmetic_.mul_root(*y, r);
20       *x++ = arithmetic_.add(u,v);
21       *y++ = arithmetic_.sub(u,v);
22       ...
23     }
24     offset += gap << 1;
25   }
26   ...
27 }
```

Figure 2: SEAL's NTT implementation. The highlighted code lines show the lines we target.

using *only a single-trace* that is captured from the victim's device. Since the key generation will occur only once for each session, the adversary is limited to a *single* power measurement.

3 THE PROPOSED ATTACK

This section presents the proposed attack and related challenges. We discuss target operations and demonstrate vulnerabilities within the implementation of the target operations.

3.1 Target Operations and Vulnerabilities

Our proposed attack focuses on the NTT which takes SEAL's secret key as input and converts them to the NTT domain during the key generation of FHE. Fig.2 shows the related code scripts of SEAL's NTT implementation. x and y pointers correspond to secret key coefficients and r value corresponds to the twiddle factors. The inner loop performs the butterfly operations of NTT. In each iteration of the inner loop, 4 butterfly operations are executed. The gap value is initially equal to 2048 for SEAL's NTT with $n = 4096$. Therefore, there are 2048 butterfly operations in each stage of NTT. The first arithmetic operation of NTT is modular reduction operation—guard which is shown in line 18 of Fig.2. x input coefficients first go through the guard function in line 18 of Fig.2. It contains a simple conditional statement that checks whether x input is greater than two times modulus ($2q$) or not. If the x coefficient is greater than $2q$, it performs a reduction. However, NTT inputs are always in $\{0, 1, q-1\} < 2q$ in the first stage of NTT. Therefore, this guard operation does not change the input values. After the guard operation, y coefficient and twiddle factor (r) go through mul_root function in line 19 of Fig.2. The twiddle factors are public values and pre-calculated before

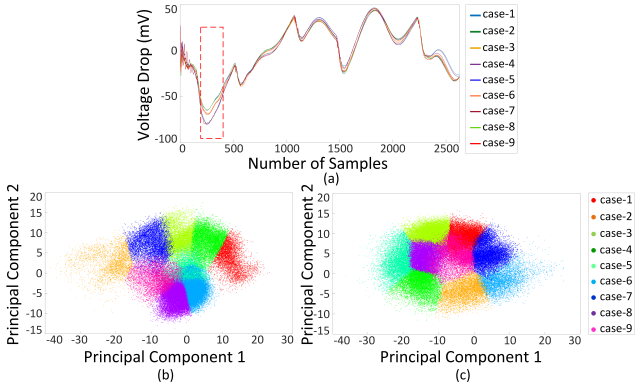

Figure 3: (a) An example of averaged power trace corresponds to an addition operation in the butterfly operation, (b) principal component analysis (PCA) scores for power traces with the samples from 200 to 350 corresponds to addition operations, (c) PCA scores for power traces with samples from 200 to 350 corresponds to subtraction operations.

the NTT operations. Also, they are smaller numbers in the first few stages of the NTT. Since the twiddle factor is updated outside of the inner loop, it is constant in the inner loop. After the multiplication of the twiddle factor and y coefficient, the outputs (u and v) of guard and multiplication operations go through addition and subtraction operations in line 20 and 21 of the Fig.2, respectively.

Since NTT's input coefficients can be 0, 1, or *q-1*, there are only 9 possible input pairs (*i.e.*, cases). For both addition and subtraction operations, their inputs (u and v) in lines 18 and 19 of the Fig.2 depend on NTT's inputs (x and y). Hence, there are 9 distinct inputs for both addition and subtraction operations. However, the inputs of multiplication operation do not depend on x coefficient. Therefore, the identification of NTT's input pairs by analyzing power measurement corresponding to multiplication operation is not possible. Our proposed attack targets only addition and subtraction operations which are highlighted in red color in the Fig.2.

3.2 Determining Point of Interest (POI) Regions

A major challenge in performing our proposed attack is finding the points of interest (POI) region of addition and subtraction operations of each butterfly of NTT. To identify POI regions, we use ML and pre-processing techniques.

Our attack first divides traces into small sampling windows. Each window contains a fixed portion of trace samples and they are labeled as 0 or 1 depending on whether it includes sample points corresponding to the power samples of addition and subtraction operation or not. Power samples in each window and their corresponding labels are fed to the ML for the training. During the test, power samples in each window are fed to our ML classifier in their natural sequence to identify POI regions corresponding to sequential arithmetic operations of NTT. Then, Pearson correlation coefficient [4] is used to validate POI regions.

The number of power samples in each window affects ML results. When the window size is smaller than the power samples

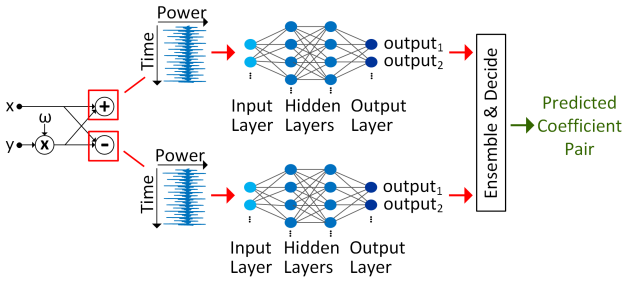

Figure 4: ML pipeline. Two distinct ML classifiers take the power measurements corresponding to the addition and subtraction operations in the NTT. The estimated results are then ensembled to predict the NTT's secret key coefficients.

corresponding to the target arithmetic operations, at least one correct guess for the guess of target trace samples can be determined. Therefore, we select the window size as 1000 for both addition and subtraction operations.

3.3 Exploiting Side-Channel Leakages

The power consumption of the processed data depends on the inputs and operations. To perform a side-channel attack, the adversary needs to model the power consumption of the device. The most well-known power models are Hamming weight, Hamming distance, and identity. According to the used model type, labels of power consumption data can be different. Since there are only 9 possible input pairs of the NTT, we used the identity model and labeled data from 1 to 9.

Fig.3-(a) shows an example of averaged power traces for all 9 input cases of the addition operation. The red dashed rectangle—power samples from 200 to 350 in the Fig.3-(a) indicates the highest leakage points—in other words, the power consumption difference for different cases in this region is highest. We use PCA [16] to see the variation of 9 different input pairs. Fig.3-(b) and (c) show the principal component analysis (PCA) scores for power traces with samples from 200 to 350 corresponding to addition operations and subtraction operations, respectively. Different colors indicate that data for each pair are grouped in a specific region which means it is not impossible to identify all input pairs statistically.

Our ML-based attack takes the whole power consumption trace corresponding to the target addition and subtraction operations rather than a specific portion of trace, automatically analyzes all samples of traces and distinguishes the power traces for all 9 cases.

3.4 Ensembled ML-based Side-Channel Attack

Our proposed attack uses two distinct ML classifiers to estimate input pairs of NTT separately for the addition and subtraction operations. Fig.4 shows our ML pipeline. Each ML classifier takes power traces corresponding to addition and subtraction operations and generates guessing scores. Table 2 shows an example of guessing scores. There are 9 possible guess scores for both addition and subtraction operations. The sum of the scores is 1 for both addition and subtraction operations. The correct pair is 5 in this example.

Table 1: Network Model Structure and Parameters for Addition and Subtraction Operations in NTT's Butterfly

	Model for Addition		Model for Subtraction	
Layer Type	Output Shape	Params. #	Output Shape	Params. #
Input	(None, 2625, 1)	0	(None, 3230, 1)	0
Conv1D-1	(None, 1314, 64)	320	(None, 1614, 64)	320
MaxPooling1D-1	(None, 657, 64)	0	(None, 807, 64)	32896
Conv1D-2	(None, 654, 128)	32896	(None, 804, 128)	32896
MaxPooling1D-2	(None, 327, 128)	0	(None, 402, 128)	0
Conv1D-3	(None, 162, 128)	65664	(None, 399, 128)	65664
MaxPooling1D-3	(None, 162, 128)	0	(None, 199, 128)	0
BatchNorm.	(None, 162, 128)	512	(None, 199, 128)	512
Flatten	(None, 20736)	0	(None, 25472)	0
Dropout	(None, 20736)	0	(None, 25472)	0
Dense	(None, 512)	10617344	(None, 512)	13042176
Output	(None, 9)	4617	(None, 9)	4617

Total parameters for addition: 10,721,353
Total parameters for subtraction: 13,146,185

The highlighted line shows that ML classifiers for addition operations guess case-2 with 0.5236 accuracy and ML classifiers for subtraction operations guess case-5 with 0.8339 accuracy. To decide which guessing is correct, our proposed attack ensembles the results by multiplying both guessing scores in each row in Table 2. The highest guessing score in the ensembled result column shows the correct guess which is case-5 in Table 2.

In our proposed attack, feeding the power samples to ML classifiers in the correct order is crucial. If ML classifiers are fed with random train and test data sets, the guessing scores of ML classifiers can correspond to different input pairs of NTT. To solve this issue, we first randomize power traces corresponding to both addition and subtraction operations at the same time. Then, we split traces for training and testing. Finally, ML classifiers are fed the power traces sequentially. In this way, each guessing score for both addition and subtraction operations matches with their corresponding input pair.

As ML model, we used a convolutional neural network (CNN) architecture which is similar to the work in [17, 19]. Table 1 shows the details of network structures and parameters. There are 3 convolutional and 3 max-pooling layers in total, sequentially as a max-pooling layer after each convolutional layer. After third convolutional and max-pooling layer, there is a batch normalization layer to prevent overfitting on the training. Also, there is a dropout layer that drops connections between neurons with a probability of 0.5 following the batch normalization. The model uses a flatten layer to convert data into a fully layer. There are 2 fully connected layers, including the output layer which has 9 neurons. Output layer uses *Softmax* activation function [5] whereas the remaining layers use *RELU* activation functions [21].

4 EXPERIMENTAL RESULTS

This section describes the measurement setup for our experiments and evaluates the proposed attack and a well-known countermeasure for NTT.

Table 2: An Example of Guessing Scores

case	score for addition	score for subtraction	ensembled result
1	$1.3448e^{-09}$	$1.1463e^{-06}$	$1.5415e^{-15}$
2	0.5236	0.1470	0.0769
3	$4.1054e^{-04}$	$8.3409e^{-04}$	$3.4242e^{-07}$
4	$5.1324e^{-09}$	$4.3093e^{-07}$	$2.2117e^{-15}$
5	0.4755	0.8339	0.3965
6	$2.9400e^{-05}$	0.0066	$1.9546e^{-07}$
7	$5.7768e^{-08}$	$3.6251e^{-06}$	$2.0942e^{-13}$
8	$4.9694e^{-04}$	0.0109	$5.4059e^{-06}$
9	$7.1847e^{-06}$	$7.7821e^{-04}$	$5.5912e^{-09}$

Correct pair: 5

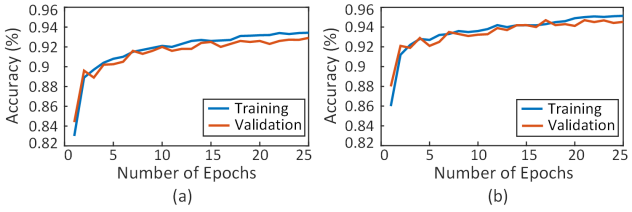

Figure 5: (a) Training and (b) validation accuracy vs number of epochs for ML models of addition and subtraction operations, respectively.

4.1 Evaluation Setup

Our evaluation setup uses a development board which contains a 32-bit ARM Cortex-M4F STM32F417IG microcontroller operating at 12 MHz. Due to our proposed attack focusing on the NTT, we only compile the SEAL's NTT code rather than the SEAL's entire code. We compile the code using `gcc-arm-none-eabi` compiler with `-O0` flag. The total memory requirement of the implemented NTT codes is around 75KB RAM and 315KB flash data storage. Since our device supports up to 196KB RAM and 1024KB flash memory, we do not use any external storage. We collect power measurements with a LeCroy WaveRunner 8104 model oscilloscope (with a 1 GS/s sampling rate) using a Riscure current probe [1].

Our ML setup is a workstation with 64 GB of random access memory (RAM), an NVIDIA 1080Ti graphics card, and an Intel i7 9700K processor. We use tensorFlow-gpu 2.8.0 as the backend, with a keras-gpu 2.8.0 front end to train and evaluate ML models.

4.2 Evaluation of ML-based Side-Channel Attack on the NTT

To evaluate our proposed ML-based side-channel attack, we use a total of 90000 power traces. We used 63000, 13500, and 13500 power traces for training, validation, and testing, respectively.

Fig.5 shows the results of the classification of the models trained with the power traces corresponding to addition and subtraction operations. When the number of epochs increases, training accuracy converges slowly and reaches around 93% and 95% for ML models of addition and subtraction operations, respectively.

[1] https://www.riscure.com/product/current-probe

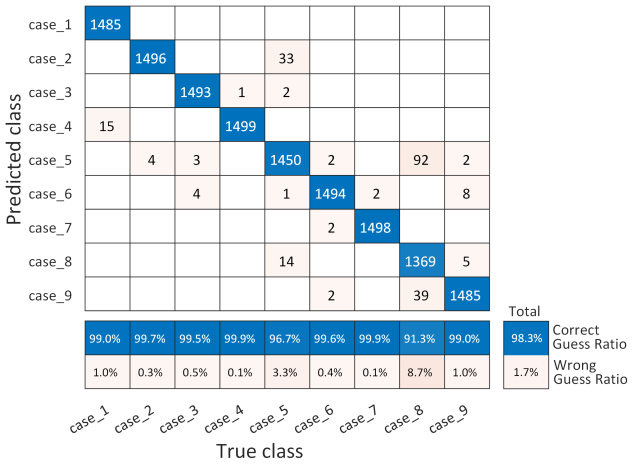

Figure 6: Confusion matrix of ensembled ML-based side-channel attack with 25 epochs.

```
for(std::size_t j = 0; j < gap; j+=4) {
    delay_function();
    u = arithmetic_guard(*x);
    delay_function();
    v = arithmetic.mul_root(*y, r);
    delay_function();
    *x++ = arithmetic_.add(u, v);
    delay_function();
    *y++ = arithmetic_.sub(u, v);
    ...
}
```

Figure 7: Random delay insertion between arithmetic operations in NTT.

Fig.6 shows the confusion matrix of our proposed ensembled ML-based side-channel attack with 25 epochs. When we individually perform ML-based side-channel attack for addition and subtraction operations, the correct guess ratios are 92% and 94%, respectively. With our ensebled ML-based side-channel attack, the total correct guess ratio increases to 98.3%.

4.3 Evaluation of Random Delay Insertion Countermeasure

Random delay insertion method which generates random delays in embedded software increases the attacker's uncertainty about the location of the target operation [8, 9]. To implement this countermeasure into NTT, we write a delay function that selects a random number between 0 and pre-selected threshold value and generates a delay depending on the selected random number. We add the delay function between each arithmetic operation of NTT shown in Fig.7.

To evaluate the random delay insertion countermeasure, we find the position of target operations with ML and then perform the side-channel attack to extract the NTT's secret key. Our attack first divides the power traces into equal trace windows, and then labels the power traces in binary format like in Section 3.2. For example, power trace window corresponding to the addition operation are labeled as 1 and the remaining trace windows are labeled as 0. ML takes power traces and labels to build ML models. Since ML can

estimate wrong results and false positives, the selection of window size by dividing power traces is very crucial. We select the size of window as 1000 that is smaller than the size of power traces corresponding to the target operations. Since the sample size is 2625 for addition and 3230 for subtraction operation, there are 3-4 and 4-5 sequential windows labeled as 1 for each addition and subtraction operations, respectively. ML estimates at least one correct guess for each target point. Then, target POI regions are identified using this ML guess results. Therefore, this countermeasure is not resistant to side-channel attacks.

5 DISCUSSIONS

In this work, we set the operating frequency of the device to 12 MHz. If we increase the operating frequency, the noise of the platform will increase. Hence, attacking may require a great number of traces to build ML models. There are multiple prior works [1, 15] which demonstrate *single-trace* side-channel attacks with lower frequencies, including 8 MHZ to attack on NTT [23].

SEAL supports different configurations with different parameters. Our attack focuses on its 128-bit security level with $n = 4096$. However, depending on the selected configuration setting, there will be a different number of NTT operations and prime modulus. Therefore, we have to build new ML models to perform the attack.

Since the goal of our work is to expose side-channel vulnerabilities of the SEAL and perform a *single-trace* attack on it, we did not concentrate on implementing a resistant countermeasure to our attack. Shuffling countermeasures can be considered a secure defense mechanism to protect the NTT [26]. We intend to implement it in the future.

6 CONCLUSIONS

In this work, we propose a new side-channel attack on an FHE library—SEAL with real power measurements. Specifically, we demonstrate a side-channel leakage coming from the NTT and perform an ensembled ML-based side-channel attack on it. We show that we are able to extract SEAL's secret key coefficients with a 98.3% accuracy. Moreover, we evaluate random delay insertion countermeasure and show that the random delay insertion countermeasure is not a suitable countermeasure to protect the NTT against our attack.

7 ETHICAL DISCLOSURES

We contacted the Cryptography and Privacy Research Group at Microsoft Research to report our preliminary findings and disclosed this paper before publication.

8 ACKNOWLEDGMENTS

This research is based upon work supported by the National Science Foundation under the Grants No. CNS 16-2137283 – Center for Advanced Electronics through Machine Learning (CAEML) and its industry members.

REFERENCES

[1] F. Aydin, A. Aysu, M. Tiwari, A. Gerstlauer, and M. Orshansky. 2021. Horizontal Side-Channel Vulnerabilities of Post-Quantum Key Exchange and Encapsulation Protocols. *ACM Transactions on Embedded Computing Systems* 20, 6 (2021), 1–22. https://doi.org/10.1145/3476799

[2] F. Aydin, E. Karabulut, S. Potluri, E. Alkim, and A. Aysu. 2022. RevEAL: Single-Trace Side-Channel Leakage of the SEAL Homomorphic Encryption Library. In *2022 Design, Automation & Test in Europe Conference Exhibition (DATE)*. 99–117. https://doi.org/10.23919/DATE54114.2022.9774724

[3] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski. 2019. nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 3–13.

[4] E. Brier, C. Clavier, and F. Olivier. 2004. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. 16–29.

[5] D. Campbell, R.A. Dunne, and N. A. Campbell. 1997. On The Pairing Of The Softmax Activation And Cross–Entropy Penalty Functions And The Derivation Of The Softmax Activation Function. In *Australian Conference on Neural Networks*. 181–185.

[6] J.H. Cheon, A. Kim, M. Kim, and Y. Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*. 409–437.

[7] J.W. Cooley and J. W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 19(90). , 297–301 pages.

[8] J.-S. Coron and I. Kizhvatov. 2009. An efficient method for random delay generation in embedded software. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. 156–170.

[9] J.-S. Coron and I. Kizhvatov. 2010. Analysis and Improvement of the Random Delay Countermeasure of CHES 2009. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. 95–109.

[10] N. Drucker and T. Pelleg. 2022. Timing Leakage Analysis of Non-constant-time NTT Implementations with Harvey Butterflies. In *International Symposium on Cyber Security, Cryptology, and Machine Learning (CSCML)*. 99–117.

[11] J. Fan and F. Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. IACR Cryptology ePrint Archive, Report 2012/144.

[12] W.M. Gentleman, G. Sande, and P. Rohatgi. 1966. Fast fourier transforms: for fun and profit. In *In Fall Joint Computer Conference (AFIPS)*. 563–578.

[13] C. Gentry. 2009. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*. 169–178.

[14] S. Halevi and S. Shoup. 2014. Algorithms in HElib. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*. 554–571.

[15] W.-L. Huang, J.-P. Chen, and B.-Y. Yang. 2019. Power analysis on NTRU Prime. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2019, 1 (2019), 123–151. https://doi.org/10.13154/tches.v2020.i1.123-151

[16] I. T. Jolliffe. 2002. *Principal Component Analysis*. Springer New York, NY, 1–488.

[17] P. Kashyap, F. Aydin, S. Potluri, P. Franzon, and A. Aysu. 2020. 2Deep: Enhancing side-channel attacks on lattice-based key-exchange. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 40, 6 (2020), 1217–1229. https://doi.org/10.1109/TCAD.2020.3038701

[18] I. Kim, T. Lee, J. Han, B. Sim, and D. Han. 2020. Novel single-trace ML profiling attacks on NIST 3 round candidate Dilithium. IACR Cryptol. ePrint Arch., Report 2020/1383.

[19] J. Kim, S. Picek, A. Henuser, S. Bhasin, and A. Hanjalic. 2019. Make some noise. Unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2019, 3 (2019), 148–178. https://doi.org/10.13154/tches.v2019.i3.148-179

[20] Q. Li, Z. Huang, W. Lu, C. Hong, H. Qu, H. He, and W. Zhang. 2020. HomoPAI: A secure collaborative machine learning platform based on homomorphic encryption. In *2020 IEEE 36th International Conference on Data Engineering*. 1713–1713.

[21] V. Nair and G.E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *International Conference on Machine Learning (ICML)*. 807–814.

[22] D. Natarajan and W. Dai. 2021. SEAL-Embedded: A Homomorphic Encryption Library for the Internet of Things. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 3 (July 2021), 756–779.

[23] P. Pessl and R. Primas. 2019. More practical single-trace attacks on the number theoretic transform. In *International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*. 130–149.

[24] Y. Polyakov, K. Rohloff, G. W. Ryan, and D. Cousins. 2022. PALASIDE lattice crypto library. https://gitlab.com/palisade/palisade-release/blob/master/doc/palisade_manual.pdf.

[25] R. Primas, P. Pessl, and S. Mangard. 2017. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. 513–533.

[26] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay. 2020. On configurable SCA countermeasures against single trace attacks for the NTT. 123–146.

[27] P. Ravi, S. Roy, A. Chattopadhyay, and S. Bhasin. 2020. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)* 2020, 3 (2020), 307–335. https://doi.org/10.13154/tches.v2020.i3.307-335

[28] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA.

[29] W. Wei X. Zheng, A. Wang. 2013. First-order collision attack on protected NTRU cryptosystem. *Microprocessors & Microsystems* 37, 6-7 (2013), 601–609.