RevEAL: Single-Trace Side-Channel Leakage of the SEAL Homomorphic Encryption Library

Furkan Aydin†, Emre Karabulut†, Seetal Potluri†, Erdem Alkim‡, Aydin Aysu†
†Department of Electrical and Computer Engineering, North Carolina State University, NC, USA
‡Department of Computer Science, Dokuz Eylul University, Izmir, Turkey

***Abstract*—This paper demonstrates the first side-channel attack on homomorphic encryption (HE), which allows computing on encrypted data. We reveal a power-based side-channel leakage of Microsoft SEAL prior to v3.6 that implements the Brakerski/Fan-Vercauteren (BFV) protocol. Our proposed attack targets the Gaussian sampling in the SEAL's encryption phase and can extract the entire message with a *single* power measurement.**

Our attack works by (1) identifying each coefficient index being sampled, (2) extracting the sign value of the coefficients from control-flow variations, (3) recovering the coefficients with a high probability from data-flow variations, and (4) using a Blockwise Korkine-Zolotarev (BKZ) algorithm to efficiently explore and estimate the remaining search space. Using real power measurements, the results on a RISC-V FPGA implementation of the SEAL (v3.2) show that the proposed attack can reduce the plaintext encryption security level from 2^{128} to $2^{4.4}$. Therefore, as HE gears toward real-world applications, such attacks and related defenses should be considered.

I. Introduction

Homomorphic encryption (HE) is a form of encryption that allows computations on encrypted data without knowing the secret key. Thus, data can remain confidential while it is processed, enabling useful computations to be accomplished with data residing in untrusted environments. This is achieved by a public-key encryption scheme where the data is encrypted first with a public (encryption) key and then by homomorphically evaluating the ciphertext. The result of such computations remains in encrypted form, and can exclusively be revealed by the owner of the corresponding secret (decryption) key. HE has been applied to the evaluation of various algorithms on encrypted financial, medical, and genomic data [1]–[4]. Such applications typically envision a user (i.e., device) that encrypts/decrypts information and a cloud (i.e., server) that performs homomorphic evaluations.

To accelerate and spread the employment of HE, the Cryptography and Privacy Research Group at Microsoft Research has developed the SEAL, which aims to provide a high-performance and easy-to-use HE software library. SEAL indeed became a popular library and has been used by many works [5], [6] including the Intel Neural Network Compiler nGraph [7], [8]. Although HE is a promising cryptographic primitive to protect data against mathematical cryptanalysis, its implementation may be vulnerable to physical attacks, e.g., to software fault injection attacks [9].

In this paper, we propose the first side-channel attack on HE. The proposed attack reduces the plaintext encryption security level to extract the plaintext messages by targeting the device-side encryption operations. We perform a power-consumption-based side-channel attack that abuses a vulnerability during the Gaussian sampling sub-routine. Unfortunately, the existing Gaussian sampling code in SEAL (v3.2)[1] has operations conditioned on the sampled coefficient's sign. But such an attack has to succeed with a *single* power measurement trace because the sampled coefficients change for each encryption. We, therefore, first reveal that the chosen condition (i.e., taken vs. not taken) can be identified from a single-trace by observing the power consumption because the conditions execute different instructions. This can expose if a coefficient is either negative, positive, or zero. We then apply horizontal side-channel analysis within the single-trace to recover the sampled coefficients. Finally, we apply the Blockwise Korkine-Zolotarev (BKZ) algorithm to efficiently explore the remaining search space and estimate the complexity of revealing the message.

Our work is different from earlier side-channel attacks on the Gaussian sampling [10]–[12] because it evaluates specific vulnerabilities of SEAL. By contrast, prior works analyze Cumulative Distribution Table based [10], [12] and Bernoulli based [11] sampling techniques, which are not used in SEAL. These works are thus not directly applicable on SEAL.

Likewise, our work is orthogonal to multi-trace attacks as they do not work by default on the encryption but can instead be useful when targeting decryption. We do not cover attacks on decryption as they are relatively straightforward extensions of earlier multi-trace analysis of lattice-based cryptography [13], [14]. At the same time, our work is orthogonal to prior single-trace side-channel attacks on other building blocks of lattice-based cryptography that have targeted the Number Theoretic Transform (NTT) [15], [16], rejection [17], polynomial multiplication [18]–[20], and message encoding/decoding [14], [21]. Although such attacks have not been exclusively shown on HE (or on SEAL), defenses built exclusively for their extension on HE will fail against our attack targeting Gaussian Sampling operations.

A summary of our contributions is as follows.

- We propose the first side-channel attack on HE. The proposed attack can be used to extract the plaintext messages that are being encrypted in the Brakerski/Fan-Vercauteren (BFV) scheme of HE. The attack is orthogonal to the possible extensions of earlier attacks on HE.
- We identify the vulnerabilities in the Gaussian sampling sub-routine of SEAL—a major HE library. We demon-

[1]SEAL v3.6 and later use different sampling algorithm: if the version number is not mentioned it means Microsoft SEAL prior to version 3.6.

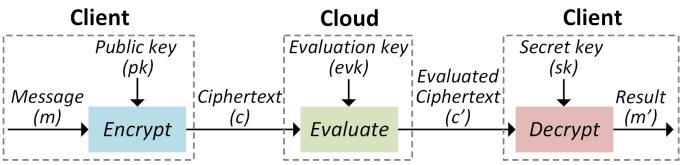

Fig. 1. Homomorphic encryption functions at the client and the cloud. The public key (*pk*) encrypts the message (*m*) to generate the ciphertext (*c*) and the secret key (*sk*) decrypts the received *homomorphically evaluated* ciphertext (*c'*) – both operations execute on the client's device while *homomorphic evaluations* execute in the cloud.

strate that the sign and the value of the sampled coefficients can be extracted from a single-trace.

- We apply the proposed attack on a RISC-V softcore processor on an FPGA running the SEAL (v3.2) software. The results showed that our attack reduces the security of the plaintext encryption from 2^{128} to $2^{4.4}$.

II. Preliminaries

This section provides the basics of HE along with a mathematical background and the threat model.

A. Homomorphic Encryption (HE)

Fig. 1 outlines HE, which comprises a set of four functions *KeyGen, Encrypt, Decrypt,* and *Evaluate*.

- *KeyGen:* Client generates public key *pk*, secret key *sk*, and evaluation key *evk*. The public key *pk* is the key used for encryption. The *pk* can be shared and used by anyone to encrypt messages. The secret key *sk* is used for decryption. The evaluation key *evk* is used for evaluation. *KeyGen* is omitted in figure for simplicity.
- *Encrypt:* This function takes as input the public key and any message *m*, performs encryption, and outputs a ciphertext *c*.
- *Evaluate:* Cloud takes as input the evaluation key *evk*, the ciphertext *c*, performs computations on the ciphertext, and produces an evaluation output *c'*.
- *Decrypt:* Client takes input secret key *sk*, evaluation output *c'*, and produces result *m'*. If the scheme operates correctly, result *m'* is the correct result of the operation on *m*.

There are many HE libraries such as SEAL [22], HElib [23], PALISADE [24], and HEAAN [25]. Specifically, we focus on SEAL and the HE protocol implementation of BFV [26] in SEAL, which is a popular implementation.

The BFV scheme is based on the Ring-LWE problem of ideal lattices. The plaintext and ciphertext spaces are the rings R_t and R_q. The elements are thus polynomials and the arithmetic operations are polynomial addition and polynomial multiplication with polynomials of degree n. The q parameter denotes modulus in the ciphertext space (coefficient modulus) of the form $q_1 \times ... \times q_k$, given k is the size of the ciphertext, and where q_i $(1 \leq i \leq k)$ is prime. The t parameter denotes modulus in the plaintext space (plaintext modulus). We use $\lfloor \cdot \rfloor$, $\lfloor \cdot \rceil$, and $[\cdot]_q$ to denote rounding down, round to nearest integer, and reduction by modulo q, respectively. By $a \xleftarrow{\$} S$, we denote that a is uniformly sampled from the finite set S. χ denotes a Gaussian sampling, and $\Delta = \lfloor q/t \rfloor$. The key values *sk* and encryption samples u are sampled with a random distribution R_2. The key generation, encryption and decryption operations in BFV is as follows.

- **SecretKeyGen:** Sample $s \xleftarrow{\$} R_2$ and output sk=s.
- **PublicKeyGen:** Sample $a \xleftarrow{\$} R_q$, and $a \leftarrow \chi$. Output pk=$([-(as+e)]q, a)$.
- **Encrypt:** Form $m \in R_t$, let pk= (p_0, p_1). Sample $u \xleftarrow{\$} R_2$, and e_1,$e_2 \leftarrow \chi$. (c_0, c_1)=$([\Delta \cdot m + p_0 \cdot u + e_1]_q, [p_1 \cdot u + e_2]_q)$.
- **Decrypt:** Output $[\lfloor \frac{t}{q}[c_0 + c_1 s]_q \rceil]_t$.

SEAL has several parameter settings. In this paper, we have targeted 128-bit security and set n=1024. SEAL also supports n= 2048, 4096, 8192, 16384, and 32768. We used the default value of the sampling parameter, which is $3.19 \approx 8/\sqrt{2\pi}$. Therefore, each sampled coefficient is between -41 and 41. Although we show our attack with this particular configuration, our attack is applicable to all security levels and values of n.

B. Threat Model and Comparison to Earlier Work

This paper describes an attack on the encryption procedure of SEAL in which the adversary tries to learn the plaintext message being encrypted. We assume the adversary knows SEAL software and its encryption parameters. This is a fair assumption given the publicly available source code [27]. Moreover, the adversary has physical access to the target device and can obtain power measurements while the device processes the encryption operations. The adversary can profile the target device before running the actual attack but does not know the message values when running the attack. These are common assumptions in template attacks. Since secret and error values are freshly computed for each new encryption operation, the adversary has to perform the attack with a *single power measurement trace*.

We show the side-channel vulnerability of Gaussian sampling but we do not claim that it is the only vulnerable operation. Other parts of the encryption such as the NTT [15], [16], rejection [17], message encoding/decoding [14], [21], and polynomial multiplication [18]–[20] can be broken by extending these earlier attacks. Likewise, decryption operations can be targeted by simply extending earlier multi-trace attacks [13], [14] to HE. We chose to attack Gaussian sampling because a single-trace on it can evade the defenses that are built for multi-trace attacks (e.g., masking) and for single-trace attacks that target other operations.

III. The Proposed Attack

This section presents the proposed attack strategy for recovering the plaintext messages which are encrypted with SEAL. We first identify the target operation and illustrate how attacking it enables full message recovery. We then identify the vulnerable points within the implementation of this target operation.

A. The Target Operation and Rationale

The proposed attack targets SEAL's Gaussian sampling operation because extracting sampled coefficient exposes the private message.

$$(c_0, c_1) = ([\Delta \cdot m + p_0 \cdot u + e_1]_q, [p_1 \cdot u + e_2]_q) \quad (1)$$

```
1  void Encryptor::set_poly_coeffs_normal
2  (uint64_t *poly, random, &context_data) const
3  {
4    ...
5    RandomToStandardAdapter engine(random);
6    ClippedNormalDistribution
7    dist(0,
8      parms.noise_standard_deviation(),
9      parms.noise_max_deviation());
10   for (size_t i = 0; i < coeff_count; i++)
11   {
12     int64_t noise = dist(engine);
13     if (noise > 0)
14     {
15       for (j = 0; j < coeff_mod_count; j++)
16         poly[i + (j * coeff_count)] = noise;
17     }
18     else if (noise < 0)
19     {
20       noise = - noise;
21       for (j = 0; j < coeff_mod_count; j++)
22         poly[i + (j * coeff_count)] =
23            coeff_modulus[j].value() - noise;
24     }
25     else
26     {
27      for (j = 0; j < coeff_mod_count; j++)
28        poly[i + (j * coeff_count)] = 0;
29     }
30   }
31 }
```

Fig. 2. SEAL's reference noise sampling implementation. The highlighted code lines shows the lines we target.

Equation (1) describes SEAL's BFV encryption scheme where m is the private message, while c_0 and c_1 are the encrypted ciphertexts. This scheme encrypts the private message m using the public keys p_0, p_1, encryption sample u, and error polynomials e_1, e_2. The error polynomials e_1, e_2 have non-uniform coefficients sampled from a Gaussian distribution, while polynomial u has uniform coefficients. Since the public keys and ciphertexts are known, an adversary can extract the private message *m* if it can first recover the error polynomials.

$$u = [(c_1 - e_2)/p_1]_q \tag{2}$$

Equation (2) re-formulates the term u in the calculation c_1. Inserting this formulation of u in the calculation of c_0 and representing m yields to:

$$m = [(c_0 - (p_0 \cdot ((c_1 - e_2)/p_1) - e_1)/\Delta]_q \tag{3}$$

Equation (3) denotes that, using the encryption equations, the private message m can be reformulated with known variables (c_0, c_1, p_0, p_1, q, Δ) and unknown variables (e_1, e_2). Therefore, SEAL's BFV encryption hardness is based on the error polynomials e_1 and e_2. This is why we chose to attack the aforementioned Gaussian sampling operation and extracted the sampled coefficients of error polynomials e_1 and e_2.

B. Identifying the vulnerabilities in the SEAL's Gaussian Sampling Implementation

To sample the error polynomials e_1 and e_2, SEAL's BFV scheme executes the `set_poly_coeffs_normal` function. Fig. 2 shows this function's implementation in C++, which we obtain from the SEAL's official GitHub repository [27]. Note that the implementation has more code lines than shown in Fig. 2 but we omit those that are unnecessary to describe our attack for brevity.

The `set_poly_coeffs_normal` function mainly consists of two sub-operations: sampling from a normal distribution and sign bit assignment, respectively. This implementation first samples double floating point values from the UniformRandomNumberGenerator provided by the C++ standard library implementation, repeat if values are greater than noise_max_deviation, and round them to the nearest integers in `ClippedNormalDistribution` function. The function returns a non-uniform value from the Gaussian distribution with the given standard deviation σ and mean μ parameters that are 3.19 and 0 for SEAL.

The next step is the sign bit assignment. SEAL's Gaussian sampling implementation controls the sign bit assignment with conditional statements—if-elseif-else branches. First, in line 13 of Fig. 2, if the sampled value is positive, the implementation assigns the sampled value to the corresponding coefficient (in line 16). If the sampled value is negative (in line 18), its sign is negated (in line 20). Then, the sampled value is subtracted from the modulus value and assigned to the corresponding coefficient (in line 23). If the sampled coefficient is neither positive nor negative, the else branch executes (in line 25) to assign 0.

We identify three vulnerabilities in the reference implementation that reveal the sampled coefficient of error polynomials. The first vulnerability is the branch operations: if the adversary can identify which branch is taken in Fig. 2, it can recover the coefficient's sign bit (positive or negative) or if the coefficient is equal to zero.

The second vulnerability is the non-uniform value assignment right after the sampling as shown in line 12 of Fig. 2. If the adversary exposes this assignment, it can extract the sampled coefficients of error polynomials. The major challenge in exploiting this vulnerability is the leakage model. If the adversary uses the Hamming weight (HW) model, many possible coefficients have the same HW representation. Therefore, the attack struggles with the false-positives.

The third vulnerability is the negation operation for the negative sampled value, line 20 in Figure 2. The adversary can eliminate some false-positives observed in the second vulnerability by targeting this negation. If any two distinct numbers have the same HW representation, their 2's complement will have different HW values. Hence, the attacker can eliminate false-positive guesses for negative sampled values by combining the second vulnerability with the third one.

`ClippedNormalDistribution` function of SEAL itself can be considered a possible attack point. This function, however, operates with 64-bit numbers internally, which complicates the attack because 2^{64} templates are needed.

C. Pinpointing Regions of Interest and Estimating the Sign

To exploit the described three vulnerabilities, we need to first isolate sampling operations of each coefficient from a full encryption execution. Since SEAL's BFV scheme runs the sampling operation 1024 times[2] to generate one error polyno-

[2]Polynomial degree in SEAL's BFV scheme range is between 1024 and 32768 so the sampling operation can run up to 32768 times.

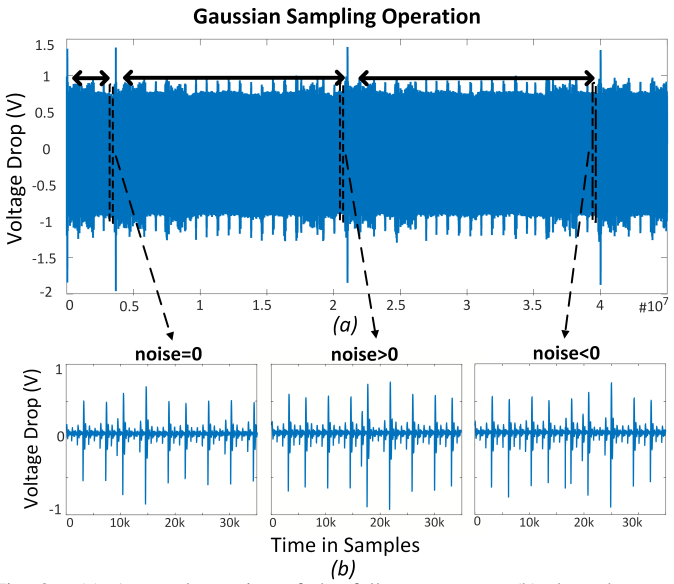

Fig. 3. (a) A sample portion of the full power trace, (b) the sub-traces correspond to the branch execution. Power measurements expose which branch is taken.

mial, isolating the sampling operations of each coefficient is challenging. Another challenge is that the distribution function in line 12 of Fig. 2 shows time-variant execution behavior. The adversary thus cannot simply locate just one iteration and then shift the sampling window for a fixed amount of time to locate other iterations. Therefore, it is essential to find a clear start and an endpoint indicator in a power trace to locate each distribution function call and the sign bit assignment.

The `set_poly_coeffs_normal` function samples the coefficient within a nested loop. Fig. 3 (a) presents a power trace for three coefficient samplings, corresponding to the three iterations of the outermost loop in Fig. 2. The adversary can locate the distribution function calls since there are distinguishable and visible peaks corresponding to the three iterations. The double-headed horizontal arrows show the parts that correspond to each independent coefficient's sampling. Indeed, these peaks are our start and an endpoint indications in the power trace to locate each distribution function call and the sign bit assignment.

Fig. 3 (b) reflects the power consumption sub-traces of three different branch's executions taken for the three different cases (noise = 0, noise $>$ 0, and noise $<$ 0). The adversary is able to distinguish each branch taken case since they have distinct power patterns caused by control flow variations (i.e., different instructions executing). Although we present only three iterations that cover the three branch taken scenarios in Fig. 3 for the ease of visualization, we tested the system with multiple traces to make sure whether there are indications in a power trace to locate each distribution function call and the sign bit assignment. Fig. 3 thus supports our initial claim about *the first vulnerability*.

D. Recovering the Sampled Coefficient Value and Exploring the Remaining Search Space

To exploit the second and third vulnerability, we used a template attack [28]. This attack configures the device with

TABLE I

ATTACK SUCCESS PERCENTAGES (%) FOR EACH COEFFICIENT. THE ROWS SHOW THE SAMPLED COEFFICIENT LABELS FOR THE TEMPLATES AND THE COLUMNS REPRESENT THE SAMPLED COEFFICIENTS.

	…	**-7**	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	…
-14	…	4	0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	…
-13	…	21.9	0.9	0	0.1	0	0	0	0	0	0	0	0	0	0	0	…
-12	…	0.5	0.6	0.2	0	0.1	0	0	0	0	0	0	0	0	0	0	…
-11	…	22.4	0.3	0	0.1	0.1	0	0	0	0	0	0	0	0	0	0	…
-10	…	1	39.6	0.8	0.3	0.1	0	0	0	0	0	0	0	0	0	0	…
-9	…	1.5	1.1	21.8	0	18.9	0	0	0	0	0	0	0	0	0	0	…
-8	…	1	0.3	0	3.6	0	0.3	0.2	0	0	0	0	0	0	0	0	…
-7	…	**41.2**	0.9	0	0.1	0	0	0	0	0	0	0	0	0	0	0	…
-6	…	1.5	**54.2**	0.3	0.2	0.2	0	0	0	0	0	0	0	0	0	0	…
-5	…	1	0.6	**54.9**	0	19.4	0	0	0	0	0	0	0	0	0	0	…
-4	…	1	0.6	0.3	**91**	0	3.4	0.9	0	0	0	0	0	0	0	0	…
-3	…	1.5	0.3	21.5	0.2	**60.7**	0.2	0.1	0	0	0	0	0	0	0	0	…
-2	…	1	0.3	0.2	3.8	0.4	**92.5**	3.1	0	0	0	0	0	0	0	0	…
-1	…	0.5	0	0	0.6	0.1	3.6	**95.7**	0	0	0	0	0	0	0	0	…
0	…	0	0	0	0	0	0	0	**100**	0	0	0	0	0	0	0	…
1	…	0	0	0	0	0	0	0	0	**31.8**	14.9	0.8	10.7	2.7	2.2	2.6	…
2	…	0	0	0	0	0	0	0	0	14.1	**27.7**	1.3	12.3	3	3.3	1.1	…
3	…	0	0	0	0	0	0	0	0	6.7	9.9	**23.5**	7.6	9.3	11.7	5.8	…
4	…	0	0	0	0	0	0	0	0	13.2	13.4	7	**20.6**	5.3	4.8	4.2	…
5	…	0	0	0	0	0	0	0	0	6.4	8.8	13.9	8	**18.1**	10.9	4.2	…
6	…	0	0	0	0	0	0	0	0	4	5.8	12.3	6.8	10.5	**17.2**	5.4	…
7	…	0	0	0	0	0	0	0	0	2.1	1.2	5.7	1.7	7.3	6.8	**16**	…
8	…	0	0	0	0	0	0	0	0	12.4	11	3.6	9.4	6.3	4.1	11.6	…
9	…	0	0	0	0	0	0	0	0	3.1	3	8.1	6.7	11.4	10.6	11.1	…
10	…	0	0	0	0	0	0	0	0	2.7	2.5	10.1	7.1	11.1	9.8	6.3	…
11	…	0	0	0	0	0	0	0	0	0.9	0.4	2.9	1.3	2	4.4	10.1	…
12	…	0	0	0	0	0	0	0	0	1.6	1.1	7.5	5.9	10.8	10.4	7.9	…
13	…	0	0	0	0	0	0	0	0	0.4	0.1	1.8	0.9	0.8	2.4	7.4	…
14	…	0	0	0	0	0	0	0	0	0.6	0	1.5	1	1.3	1.4	6.3	…

all possible secrets (sampled coefficients in our case) and uses template profiling [28] to extract the target device's power measurement behavior. The template is effectively a multivariate distribution that describes the key samples in the power traces. The attack calculates the probability of a given power measurement belonging to each template and chooses the one that maximizes the probability.

Since using the entire power trace makes an impractical template [29], we selected special point of interests (POI) within each power trace that has relatively higher leakage. We used sum-of-squared differences (SOSD) method [30] to identify POIs. Then, we build the templates with the selected POIs and conduct the template attack. Our attack combines the two template results: the template build on the third vulnerability reduces false positives in the second vulnerability.

We use extracted coefficients from the template attack and combine it with the sign information to explore the remaining search space with the BKZ algorithm [31] and estimate the complexity of revealing the private message of SEAL. The next section discusses how we quantify the amount of hints of sampled coefficients and the remaining search space.

IV. Attack Results

A. Experimental Setup

We implemented the hardware on SAKURA-G Board which has a Xilinx Spartan-6 FPGA. We set the operating frequency at a constant 1.5 MHz since this is in the range of RFIDs/MCUs [32], [33]. Note that there are multiple prior works [18]–[20] on single-trace side-channel attacks demonstrated with similar frequencies. We used RISC-V based architecture and

specifically selected the PicoRV32 core [34] for the implementation with the RV32IM configuration that supports 32-bit based integer and standard extension for integer multiplication and division.

The SAKURA-G Board has a designated SMA port that provides the power drop across a shunt resistor of 1Ω on the main supply line. To obtain power measurements, we use a PicoScope 6424E model oscilloscope at 1GS/s.

B. Side-Channel Analysis Accuracy

SEAL's BFV scheme generates fresh samples for each message. Our attack thereby aims to recover the error polynomial's coefficients from a single side-channel measurement. We ran the Gaussian sampling operation 220,000 times to create template profiles sampled coefficients. We then captured 25,000 power traces for the attack stage. Note that the actual attack only uses a single measurement but we ran the attack many times to get a statistical estimate of our success rate.

Table I shows our attack scores for different coefficients. The columns represent the actual value of the sampled coefficients during the attack stage, while the rows show the predicted value. Although the range is between -41 and 41, we observed values between -14 and 14 with 220,000 tests—Table I shows attack scores for coefficients between -7 and 7 for brevity.

Our attack has 100% success rate for guessing the sign of the coefficients. Table I shows that, as expected, the negative values are more accurately extracted given the third vulnerability (negation operation) we exploit. For example, HWs of coefficient -2 and -3 are the same but the HWs of 2 and 3 are different. Therefore, when -2 and -3 are negated, they will manifest different power consumption behaviors.

C. Integrating 'LWE with Hints'

To estimate the remaining search space based on our attack results, we applied a recent technique [31], which integrates side-channel information into the Learning-with-Errors instance to analyze its effect on the cryptanalytic security of the scheme. The framework extends the primal attack reductions by inserting distorted bounded distance decoding (DBDD) problem before the unique shortest vector problem (uSVP). It then reports the hardness of the uSVP instance as the block size of the BKZ algorithm. To translate the block size to the bit security, related SVP hardness must be calculated. Their seminal paper reports that bikz corresponds to $2.98\times$ of the bit-level security [31][3]. **We follow the same approach, apply their code on our results, and report security results in the same manner.**

Embedding LWE instance into DBDD allows analyzing security effects of information gained by side-channel leakages. To this end, the framework [31] is able to integrate the following information into the DBDD instance:

- Perfect hints: $\langle s, v\rangle = l$
- Modular hints: $\langle s, v\rangle = l \bmod k$
- Approximate hints: $\langle s, v\rangle = l + \epsilon_\sigma$
- Short vector hints: $v \in \Lambda$

[3]For example, we choose SEAL parameter set for 128-bit security level which corresponds to 382.25 bikz.

TABLE II
GUESSING PROBABILITIES DERIVED FROM SELECTED MEASUREMENTS. WE FOLLOW THE EARLIER METHOD [31], THE ATTACK IS "NOT" DONE JUST FOR -2 TO +2 BUT FOR THE FULL SET OF COEFFICIENTS; RESULTS GIVEN HERE IS FOR A SUBSET AS IN [31] FOR SIMPLICITY.

	...	-2	-1	0	1	2	...	centered	variance
0	...	0	0	1	0	0	...	0	0
1	...	0	0	0	≈ 1	$2.7 \cdot 10^{-10}$	...	1	$2.7 \cdot 10^{-10}$
-1	...	0	1	0	0	0	...	-1	0
2	...	0	0	0	$2.8 \cdot 10^{-53}$	≈ 1	...	2	0
-2	...	1	0	0	0	0	...	-2	0

TABLE III
COST OF ATTACK WITH/WITHOUT HINTS FOR SEAL-128 PARAMETER SETS.

	SEAL-128
Attack without hints (bikz)	382.25
Attack with hints (bikz)	12.2

TABLE IV
COST OF ATTACK WITH/WITHOUT HINTS USING "ONLY" BRANCH VULNERABILITY FOR SEAL-128 PARAMETER SETS

	SEAL-128
Attack without hints (bikz)	382.25
Attack with hints (bikz)	253.29
Attack with hints & guesses (bikz)	252.83
Number of guesses	1
Success probability	20%

For the single-trace attack, we focus on perfect and approximate hints since the perfect hints allow integrating information gained by noisy power leakages with high guessing confidence. Approximate hints allow integrating the same information when the guessing confidence is lower.

The framework takes the scores of each measurement and creates probabilities for each output. Then, it generates n secret values and selects measurements for those values uniformly at random. Finally, the probability tables for those measurements are integrated into DBDD instance in order to estimate the hardness of the problem. Our experiments in Table I show that some coefficients can be guessed correctly with very high probability, while others have relatively lower probability. Thus the framework uses them as the perfect hints or approximate hints in accordance with their probabilities. Table II shows the probabilities of guesses for several secrets as an example. The probability of the correct guesses in the Table II is very close to 1. Therefore, the framework selects those measurements as prefect hints.

For simplicity, Table II only includes the guessing probabilities of $(-2, 2)$ intervals since they are more frequently observed. We have noticed that some possibilities rounded up to 1 or down to 0 because of the floating-point precision, therefore we marked those values with $\approx$ sign.

Note that Table I shows the success rates when most likely template is selected as the result while Table II computes the success rate from the possibilities of each template without selecting any candidates as in [31]. Thus there might be a slight difference between them. Table III shows the cost of primal attack for the smallest parameter set of SEAL-128 where $q = 132120577$, $n = 1024$, and $\sigma = 3.2$. Since our attack can guess most of the coefficients of the secret with high confidence, i.e., the distribution has a variance that is very close if not equal to 0, the cost when the hints are used is only 12.2 bikz, which can be interpreted as complete break of the scheme (i.e., security level of about $2^{4.4}$ [31]).

We also analyze the case where the adversary only leverages the branch-related vulnerability. In this scenario, the adversary can guess the sign of the coefficient and if the coefficient is equal to zero correctly[4]. The cost of the primal attack is given in Table IV. The hints reduce security bikz from 382.25 to 253.29 (equivalently, from a security level of 2^{128} to $2^{84.34}$ [31]). Therefore, **signs alone cannot recover** the plaintext message. The second and third vulnerabilities we identify should be used for a successful attack.

V. Discussions

A. Potential Defenses

Our goal is to inform the single-trace side-channel vulnerabilities and to quantify if they have the potential to lead to a successful attack. We urge the developers to incorporate some form of countermeasures to prevent the attacks we propose. Such defenses may involve shuffling or other forms of randomization/obfuscation. We do not recommend masking-based defenses as they are known to be susceptible against single-trace side-channel attacks [15]. The combination of various defense approaches is most likely the best option for both single-trace and multi-trace side-channel attacks.

Microsoft SEAL has been patched since the beginning of this research. SEAL v3.6 update uses an iterator function instead of using if-else conditions we analyzed [35]. Therefore, SEAL v3.6 and later versions may have a different vulnerability, which is left for future work.

B. Drawbacks of Our Attack

Template attacks need profiling and the ability to configure keys. They may require a great number of traces to create a good template. They also need to take the curse of dimensionality into account [36]. We limit our attack to a single device, cross-device attacks may need a more complicated, machine-learning-based profiling [20]. Since the noise of the platform increases with the operating frequency of the device, we set the operating frequency of the design to a constant 1.5 MHz. Attacking devices with higher clock frequency may require utilizing more advanced measurement equipment. Likewise, attacking more secure versions (196-bit or 256-bit) is likely to be harder because of the increased precision and number of coefficients.

VI. Conclusions

So far, the research on HE has been on making it more practical given its high computational overhead. However, as HE is now starting to move into real-world applications, more focus is needed on their implementation security. This paper proposes the first side-channel attack on Microsoft SEAL, which is a major HE software library. We demonstrate the unique vulnerabilities of the Gaussian sampling sub-routine in SEAL and validate the practicality of our attack with real measurements. Therefore, some form of countermeasure is needed. Since we apply a single-trace side-channel analysis, masking would not be a viable option to mitigate this vulnerability. We thus encourage countermeasures based on shuffling and better software coding practices to eliminate conditional executions on sensitive values.

[4]Our attack can guess the correct branches with 100% success rate.

VII. Ethical Disclosures

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

VIII. Acknowledgements

This research is based upon work supported by the National Science Foundation under the Grants No. CNS 16-244770 – Center for Advanced Electronics through Machine Learning (CAEML) and its industry members.

References

[1] K. Lauter et al, "Private computation on encrypted genomic data," in *progress in Cryptology – LATINCRYPT*, 2014, pp. 3–27.
[2] A. Acar et al, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–35, 2018.
[3] J. H. Cheon et al, "Homomorphic computation of edit distance," in *Financial Cryptography and Data Security*, 2015, pp. 193–212.
[4] J. J. Sim et al, "Achieving GWAS with homomorphic encryption," *BMC Medical Genomics*, vol. 13, no. 90, pp. 1–12, 2020.
[5] Q. Li et al, "HomoPAI: A secure collaborative machine learning platform based on homomorphic encryption," in *IEEE ICDE*, 2020, pp. 1713–1713.
[6] W. Briguglio et al, "Machine learning in precision medicine to preserve privacy via encryption," *ArXiv*, vol. abs/2102.03412, 2021.
[7] F. Boemer et al, "nGraph-HE: a graph compiler for deep learning on homomorphically encrypted data," in *ACM CF*, 2019, pp. 3–13.
[8] F. Boemer et al, "A high-throughput framework for neural network inference on encrypted data," in *ACM WAHC*, 2019, pp. 45–56.
[9] I. Chillotti et al, "Attacking FHE-based applications by software fault injections," IACR Cryptol. ePrint Arch., Report 2016/1164, 2016.
[10] S. Kim et al, "Single trace analysis on constant time CDT sampler and its countermeasure," *Applied Sciences*, vol. 8, no. 10, pp. 1–16, 2018.
[11] T. Espitau et al, "Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers," in *ACM CCS*, 2017, pp. 1857–1874.
[12] C. Zhang et al, "A flexible and generic gaussian sampler with power side-Channel countermeasures for quantum-secure internet of things," *IEEE IoT Journal*, vol. 7, no. 9, pp. 8167–8177, 2020.
[13] X. Zheng et al, "First-order collision attack on protected NTRU cryptosystem," *Microprocessors & Microsystems*, vol. 37, no. 6-7, pp. 601–609, 2013.
[14] P. Ravi et al, "Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs," in *CHES*, 2020, pp. 307–335.
[15] R. Primas et al, "Single-trace side-channel attacks on masked lattice-based encryption," in *CHES*, 2017, pp. 513–533.
[16] I.-J. Kim et al, "Novel single-trace ML profiling attacks on NIST 3 round candidate Dilithium," IACR Cryptol. ePrint Arch., Report 2020/1383, 2020.
[17] A. P. Fournaris et al, "Profiling Dilithium digital signature traces for correlation differential side channel attacks," in *SAMOS*, 2020, pp. 281–294.
[18] W.-L. Huang et al, "Power analysis on NTRU prime," in *CHES*, 2020, pp. 123–151.
[19] A. Aysu et al, "Horizontal side-channel vulnerabilities of post-quantum key exchange protocols," in *IEEE HOST*, 2018, pp. 81–88.
[20] P. Kashyap et al, "2Deep: Enhancing side-channel attacks on lattice-based key-exchange," *IEEE TCAD*, vol. 40, no. 6, pp. 1217–1229, 2020.
[21] B.-Y. Sim et al, "Single-trace attacks on the message encoding of lattice-based KEMs," *IEEE Access*, vol. 8, pp. 183175–183195, 2020.
[22] K. Laine, "Simple encrypted arithmetic library 2.3.1," Technical report, 2018.
[23] S. Halevi et al, "Faster homomorphic linear transformations in HElib," in *CRYPTO*, 2018, pp. 93–120.
[24] Y. Polyakov et al, "PALASIDE lattice crypto library," https://gitlab.com/palisade/palisade-release/blob/master/doc/palisade_manual.pdf, 2021.
[25] J. H. Cheon et al, "Bootstrapping for approximate homomorphic encryption," in *Eurocrypt*, 2018, pp. 360—-384.
[26] J. Fan et al, "Somewhat practical fully homomorphic encryption," IACR Cryptol. ePrint Arch., Report 2012/144, 2012.
[27] "Microsoft SEAL (release 3.2)," Retrieved from https://github.com/Microsoft/SEAL, Aug. 2019, Microsoft Research, Redmond, WA.
[28] S. Chari et al, "Template attacks," in *CHES*, 2002, pp. 13–28.
[29] C. Rechberger et al, "Practical template attacks," in *WISA*, 2004, pp. 440–456.
[30] G. Fan et al, "How to choose interesting points for template attacks more effectively?" in *INTRUST*, 2014, pp. 168–183.
[31] D. Dachman-Soled et al, "LWE with side information: Attacks and concrete security estimation," in *CRYPTO*, 2020, pp. 329–358.
[32] "STMicroelectronics 8-bit MCUs," Retrieved from https://www.st.com/en/microcontrollers-microprocessors/stm8-8-bit-mcus.html, Sep. 2021.
[33] "MaximIntegrated secure MCUs," Retrieved from https://para.maximintegrated.com/en/search.mvp?fam=micros&1233=Secure, Sep. 2021.
[34] C. Wolf, "Picorv32," https://github.com/cliffordwolf/picorv32.
[35] "Microsoft SEAL (release 3.6)," Retrieved from https://github.com/Microsoft/SEAL, Nov. 2020, Microsoft Research, Redmond, WA.
[36] L. Lerman et al, "Template attacks vs. machine learning revisited and the curse of dimensionality in side-channel analysis," *J Cryptogr Eng*, vol. 8, pp. 301–313, 2018.