applied sciences

MDPI

Article

Single Trace Side Channel Analysis on NTRU Implementation

Soojung An [1], Suhri Kim [1], Sunghyun Jin [1], HanBit Kim [1] and HeeSeok Kim [2,*]

[1] Graduate School of Information Security and Institute of Cyber Security & Privacy (ICSP), Korea University, Seoul 02841, Korea; soojung02@korea.ac.kr (S.A.); suhrikim@gmail.com (S.K.); sunghyunjin@korea.ac.kr (S.J.); luz_damoon@naver.com (H.K.)

[2] Department of Cyber Security, College of Science and Technology, Korea University, 2511 Sejong-Ro, Sejong 30019, Korea

* Correspondence: 80khs@korea.ac.kr; Tel.: +82-044-860-1383

Received: 17 September 2018; Accepted: 17 October 2018; Published: 23 October 2018

Abstract: As researches on the quantum computer have progressed immensely, interests in post-quantum cryptography have greatly increased. NTRU is one of the well-known algorithms due to its practical key sizes and fast performance along with the resistance against the quantum adversary. Although NTRU has withstood various algebraic attacks, its side-channel resistance must also be considered for secure implementation. In this paper, we proposed the first single trace attack on NTRU. Previous side-channel attacks on NTRU used numerous power traces, which increase the attack complexity and limit the target algorithm. There are two versions of NTRU implementation published in succession. We demonstrated our attack on both implementations using a single power consumption trace obtained in the decryption phase. Furthermore, we propose a countermeasure to prevent the proposed attacks. Our countermeasure does not degrade in terms of performance.

Keywords: side channel analysis; single trace analysis; post quantum cryptography; NTRU

1. Introduction

The currently used public key cryptography (PKC) such as RSA and Elliptic Curve Cryptography (ECC) are no longer secure if the quantum computer is developed running the Shor algorithm [1–3]. Due to the recent advances in quantum computing, post-quantum cryptography (PQC) is an active area of research. Moreover, the national institute of standards and technology (NIST) announced a project to define new standards for the PKC [4]. There are five categories studied for PQC: lattice-based cryptography, multivariate-based cryptography, hash-based cryptography, code-based cryptography, and isogeny-based cryptography. Among those categories, lattice-based cryptography is one of the prominent candidates due to the fast performance with a practical key size. In the same context, the largest number of candidates submitted to NIST project belong to the lattice-based cryptography. One of the well known lattice-based cryptography is NTRU, which is an abbreviation of N-th degree truncated polynomial ring.

NTRU proposed in 1996 by Hoffstein et al. [5] is an encryption algorithm based on the shortest vector problem. NTRU has attracted much attention to the researchers due to the faster speed than classical cryptosystems by more than two orders of magnitude on the same security level. Regarding the implementation, only the encryption code was open to the public until 2017. After the patent release in 2017, all of the source code was available. Currently, two kinds of implementation are proposed in the literature, one released on GitHub in 2017 and the other submitted on NIST standardization project. We distinguish the prior one as NTRU Open Source and the latter as NTRUEncrypt, and also the

algorithm itself as NTRU. Although NTRU has withstood various mathematical attacks, the security of its implementation is an open question.

After the proposal of the side channel analysis (SCA) by Kocher et al. [6] in 1996, most of the cryptosystems consider a SCA as a de facto standard in nowadays. Additionally, since the resistance of a SCA is included as a requirement in FIPS140-2 (Federal Information Processing Standard publication 140-2), NTRU should consider the resistance against SCA to substitute the RSA and ECC. Moreover, the NIST standardization project suggests a resistance to SCA for the submitted candidates. The SCA is an attack using additional information such as time, sound, and power consumption during the operation of a cryptographic device. Among these methods, power analysis attack such as the differential power analysis (DPA) and simple power analysis (SPA) is known to be the most practical method. The SPA performs by analyzing a single power consumption trace of the device. The DPA is a statistical approach by exploiting a number of power traces related to secret data. Even though the cryptographic algorithm is theoretically safe, the private key or secret message can be exposed by the side-channel leakage when executing the algorithm. In this regard, there are previous studies on SCA on NTRU by Lee et al. [7] and Zheng et al. [8]. They performed DPA on NTRU and revealed the secret key. However, whether other types of power analysis can be performed has not been analyzed so far.

1.1. Our Contribution

In this paper, we propose the first single trace side channel analysis (STA) against on both NTRU Open Source and NTRUEncrypt with experimental results, and propose a countermeasure. Previous SCA on NTRU [7–9] targeted the polynomial multiplication between the cipher-text and the secret key. However, since NTRU was patented until 2017, existing SCAs on NTRU are based on the assumption that publicized polynomial multiplications are used in the decryption process. In this paper, we performed STA on the decryption algorithm used in [10] and on the version submitted in NIST standardization project [11].

Based on the proposed analysis, every NTRU based cryptosystem can be vulnerable to our attack. Moreover, as the PKC is mostly used to exchange the session key between two parties, there might be the case where power consumption trace can only be obtained by one execution of the algorithm. Since we recover with a single power consumption trace, our attack is indeed a threat to these implementations whereas existing DPA cannot be applied in this circumstances. We implement the algorithm on the ATmega128 8-bit processor of the KLA-SCARF AVR [12] and applied the proposed attack. The details of our attack are presented in Section 3.

We propose two versions of countermeasure against the proposed analysis. Although the previous DPA target the different implementation, their method can still be applied on NTRU Open Source and NTRUEncrypt. In this paper, we propose a countermeasure that prevents not only our proposed attack but also the DPA. The proposed countermeasure on NTRU Open Source does not increase the computational cost. Furthermore, the proposed countermeasure on NTRUEncrypt reduces the computational cost approximately by half. The description of our countermeasure is presented in Section 4.

1.2. Organization

This paper is organized as follows. In Section 2, we describe NTRU and its implementation. Also, previous SCAs on NTRU are described. In Section 3, we propose our single trace attack and show experimental results. Next, proposed countermeasures and computation comparisons are in Section 4. We make our conclusion in Section 5.

2. Background

2.1. Algorithm of NTRU

The NTRU is a PKC based on the shortest vector problem whose computational complexity is exponential even in the presence of a quantum computer. The encryption and decryption scheme use polynomial rings in $\mathcal{R} = \mathbb{Z}[X]/(x^N - 1)$, which consist of all polynomials with degree less than N and coefficients in $\mathbb{Z}$. Thus, an element $f \in \mathcal{R}$ can be written as $f = \sum_{i=0}^{N-1} f_i x^i$. The polynomial multiplication in $\mathcal{R}$ is denoted as $\cdot$, and is performed as in Equation (1).

$$\begin{gathered} h = f \cdot g,\ f, g, h \in \mathcal{R} \\ h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i} = \sum_{i+j \equiv k \pmod N} f_i g_j \end{gathered} \tag{1}$$

Let $\mathcal{L}_f$ be a set of $f \in \mathcal{R}$ with $d_f + 1$ coefficients equal to 1 and d_f coefficients to -1 and let $\mathcal{B}_g$ be a set of $g \in \mathcal{R}$ with d_g coefficients equal to 1 and -1, where d_f and d_g are fixed parameter. We express the polynomials in $\mathcal{L}_f$ and $\mathcal{B}_g$ as a trinary polynomial because they consist of only three number of coefficient. The modulus values of integers p and q are used and they satisfy the conditions $gcd(p, q) = 1$ and $p \ll q$. We define f_p as a polynomial in $\mathbb{Z}_p[X]/(x^N - 1)$ obtained by reducing the coefficients of $f \in \mathcal{R}$ modulo p. The inverse of f_p in $\mathbb{Z}_p[X]/(x^N - 1)$ is denoted as f_p^{-1}. The f_q and f_q^{-1} are defined in the same manner.

Key Generation

The private key f is a trinary polynomial selected from $\mathcal{L}_f$ and the public key h satisfies $h = p f_q^{-1} \cdot g \pmod q$, $g \in \mathcal{B}_g$. The public key is used in the data encryption and private key is used in the data decryption.

Encryption

The purpose of encryption is to transport the data by converting a message using the public key of the receiver. Then only an owner of proper private key can decrypt the message. To encrypt a plain-text $m \in \mathcal{B}_m$, we first choose a random polynomial r in $\mathcal{B}_r$ and compute the cipher-text e as Equation (2).

$$e = r \cdot h + m \pmod q \tag{2}$$

The modulus q in the above equation means that each coefficient in a polynomial is reduced modulo q.

Decryption

Decryption is used to recover the message from sender. The received data is usually called as cipher-text. The cipher-text e is decrypted by computing the following equations.

$$a = f \cdot e \pmod q \tag{3}$$

$$m = a \cdot f_p^{-1} \pmod p \tag{4}$$

The correctness of the decryption is confirmed by the Equations (5) and (6).

$$\begin{aligned} a &= f \cdot e \pmod q \\ &= r \cdot h \cdot f + m \cdot f \\ &= pr \cdot g + m \cdot f \pmod q \end{aligned} \tag{5}$$

$$\begin{aligned} a \cdot f_p^{-1} &= (pr \cdot g + m \cdot f) \cdot f_p^{-1} \pmod{p} \\ &= m \cdot f \cdot f_p^{-1} \pmod{p} \\ &= m \end{aligned} \tag{6}$$

Note that by choosing the private key f as $pF + 1$ where $F \in \mathcal{L}_f$, then f_p^{-1} is equal to 1 so that the Equation (4) can be omitted [13]. Both target of this paper (NTRUEncrypt, NTRU Open Source) use this optimization.

2.2. Side Channel Analysis and Related Work

Although an algorithm is mathematically secure, naive implementation can make cryptosystem vulnerable to various attacks. The most important considerations in implementation are random number generator and leakage of the secret information. In most cryptosystems, the quality of the random numbers used directly determines the security of the system. Therefore, a predictable random value (i.e., low entropy source) may weaken the system. The studies for the randomness have done in the respect of entropy [14–19]. However, the analysis herein discusses the implementation in the side of SCA.

The SCA is first introduced by Kocher et al. in 1996 [6]. Subsequently, power analysis attacks such as the simple power analysis (SPA), differential power analysis (DPA), and correlation power analysis (CPA) [20,21] have been proposed. Nowadays, any attack that exploits information gained from the implementation is considered as SCA. This includes cache attack, EM analysis, and attacks that exploit hardware vulnerabilities [22–25]. However, we mainly focus on the power analysis attack. The power analysis attacks rely on the dependency between the power consumption of the device and the operated data during the execution of an algorithm. The SCA is an actual threat since it can recover the private key of the cryptosystem in practical time. To prevent these type of attack, masking and hiding are studied [26]. Masking refers to a method of computing secret information with random values, so that the actual value is unused during the encryption and decryption. Hiding removes the relationship between power consumption and the data. Hiding is one of the hardware level countermeasure which is focused on the security during the operation.

Additionally, the Internet of Things (IoT) devices are advanced nowadays, the security against low-power design is essential [27]. However, the conventional PKC is difficult to implement on the resource-constrained environment. Therefore, there is a research on the physical unclonable function (PUF) as a light-weight authentication security primitive. For example, side-channel resistant PUF was intensively studied in [28].

2.3. Previous Side Channel Analysis on NTRU

The first studied SCA on the NTRU was timing attacks in 2007 [29]. In 2010, Lee et al. introduced a SPA and CPA on NTRU and proposed a countermeasure against the attack [7]. The idea behind the proposed SPA in [7] is that there exists a difference in the power consumption when adding non-zero with zero values and non-zero with non-zero values. They also performed CPA on the multiplication using 1000 traces. Also, a second order CPA is proposed in [7] using 10,000 traces. For the description of the attack, please refer to [7]. To prevent the SPA, they proposed to initialize the temporary buffer with a non-zero value and to randomize the order of computation and data. They also provided countermeasures against CPA such as masking and shuffling.

In 2013, a first-order collision attack was proposed in [8] with the purpose of incapacitating the countermeasure proposed in [7]. Their attack against the first-order countermeasure is an improvement in [8] since the attack is performed with 5,000 traces. The target of the attack was when the same registers are loaded during the multiplication. Overall, the decryption code used for the analysis in [7] and [8] was not an official implementation. Although the proposed attacks can be applied to official implementation, the attack environment is restricted to the case where multiple executions of NTRU with the same key is possible.

3. Proposed Single Trace Side Channel Analysis on NTRU Implementation

In this section, we propose our STA on the two cases of NTRU implementation. For each case, we first describe the implementation and then suggest our STA. Lastly, we present the experimental results on our attack. The purpose of our attack is to recover the private key. Therefore, only the implementation of decryption is introduced in this paper.

3.1. NTRU Open Source

The integral parts of NTRU implementation are the way to store polynomials and a polynomial multiplication.

Representing Polynomials

To store a polynomial f of the private key, NTRU Open Source stores the degree of indeterminant x whose coefficient is -1 or 1. Because the addition is computed according to the degree of -1 and 1, it is possible to operate without the degree of 0. Thus, the private key array first stores all the degree whose coefficient is 1 and then it stores all the degree where its coefficient is -1 in an array. For example, if $f = x^3 - x + 1$, then the array of f would be $\{0, 3, 1\}$. The polynomial in general, is stored such that the coefficient of the xth degree is the xth element in an array. For example, the polynomial $e = 3x^4 - x^2 + 9x - 5$ represent as $\{-5, 9, -1, 0, 3\}$.

Polynomial Multiplication

For efficiency, the private key is set as $f = pF + 1$ and F is divided into three trinary polynomials $F = F_1 \cdot F_2 + F_3$, $F_1, F_2,$ and $F_3 \in \mathcal{L}_F$. The advantage of splitting F, is that it lowers the hamming weight of polynomials so that the multiplication could be speed up [13,30]. Consequently, the decryption of NTRU Open Source performs as in Equation (7) considering the order of multiplication.

$$a = f \cdot e = (1 + pF) \cdot e = (1 + p(F_1 \cdot F_2 + F_3)) \cdot e = e + p(((e \cdot F_1) \cdot F_2) + (e \cdot F_3)) \tag{7}$$

Computation of Equation (7) is represented in Algorithm 1 and algorithm for polynomial multiplication is in Algorithm 2.

Algorithm 1 Decryption in NTRU Open Source

Require: The trinary polynomials F_1, F_2, F_3 and $e \in R$ with degree N ▷ F_1, F_2, F_3 is a private key polynomial satisfied $f = 1 + p(F_1 \cdot F_2 + F_3)$
Ensure: message $m = f \cdot e \pmod q = (1 + (F_1 \cdot F_2 + F_3)) \cdot e \pmod q$
1: $t \leftarrow$ Algorithm 2(F_1, e) ▷ Algorithm 2 is polynomial multiplication
2: $t \leftarrow$ Algorithm 2(F_2, t)
3: $u \leftarrow$ Algorithm 2(F_3, e)
4: **for** $0 \leq i < N$ **do**
5: $\quad v_i \leftarrow (t_i + u_i) \pmod q$ ▷ add t and u
6: **end for**
7: **for** $0 \leq i < N$ **do**
8: $\quad m_i \leftarrow (e_i + p * v_i) \pmod q$ ▷ $*$ is a word multiplication
9: **end for**
10: **return** m

The input b of Algorithm 2 is formed in a way such that the degree having coefficient 1 is stored in ascending order and then degree having -1 is stored. The polynomial multiplication starts with the smallest degree where its coefficient equals to -1 and add cipher-text to the initialized array. Since the result must be reduced modulo $(x^N - 1)$, this implementation performs the addition from the beginning to $(N - 1)$ and restarts for the 0th element in an array when the degree exceeds N. After the modular operation, the sign is reversed and the same steps are repeated on for the degree having

coefficient 1. Lastly, the $(\text{mod } q)$ operation is performed by AND($\wedge$) $(q-1)$ since the q is set as power of 2.

Algorithm 2 Polynomial Multiplication during NTRU Open Source Decryption

Require: Polynomial $e \in R$ with degree N and Private key array b ▷ let b be a information of private key F
Ensure: $H = F \cdot e \ (\text{mod } q)$
1: **for** $i = 0; i < N; i$++ **do**
2: $t_i \leftarrow 0$
3: **end for**
4: **for** $j = d_F + 1; j < 2d_F + 1; j$++ **do** ▷ private key has d_F coefficients equal -1
5: $k \leftarrow b_j$
6: **for** $i = 0; k < N; i$++, k++ **do**
7: $t_k \leftarrow t_k + e_i$
8: **end for**
9: **for** $k = 0; i < N; i$++, k++ **do**
10: $t_k = t_k + e_i$
11: **end for**
12: **end for**
13: **for** $i = 0; i < N; i$++ **do** ▷ This step is because the above process is for -1
14: $t_i \leftarrow -t_i$
15: **end for**
16: **for** $j = 0; j < d_F + 1; j$++ **do** ▷ private key has $d_F + 1$ coefficients equal 1
17: $k \leftarrow b_j$
18: **for** $i = 0; k < N; i$++, k++ **do**
19: $t_k \leftarrow t_k + e_i$
20: **end for**
21: **for** $k = 0; i < N; i$++, k++ **do**
22: $t_k = t_k + e_i$
23: **end for**
24: **end for**
25: **for** $i = 0; i < N; i$++ **do**
26: $H_i \leftarrow t_i \ (\text{mod } q)$ ▷ in the case of q is powering of 2, $\wedge(q-1)$ works for mod q
27: **end for**
28: **return** H

3.1.1. Proposed Method

The idea behind the attack is that the correlation between power consumption traces obtained when performing the same operations is higher than the power consumption trace obtained when performing different operations. Let the power trace obtained during the addition operation be taken as a reference trace R. Let O be the subtraces of the power consumption trace in Algorithm 2. When calculating the correlation between R and O, the correlation coefficient will be obtained when computing Algorithm 2. When plotting the gained coefficients values, then a graph appear like Figure 1. There are peaks, called as high peak herein, which signify the affinity between R and O. Then, we recover the private key polynomial by calculating the distance between the high peaks.

As in Algorithm 2, the additions in steps 4 to 12 and steps 16 to 24 depend on the private value. For example, suppose $N = 11$ and let 5 be the smallest degree when its coefficient equals to -1. Then the steps 6 to 8 are repeated 6 times and steps 9 to 11 are repeated 5 times. Note that, there is a moment when the loop passes to the next loop, then the distance between high peaks is different at that moment. Thus, if the real value is x, so that the interval between $(N-x)$th and $(N-x+1)$th high peak is different from the others. Therefore, we can recover the whole value by applying the same steps for the coefficients -1 and 1.

3.1.2. Experiment

Figure 2 is a full trace of the NTRU Open Source porting on an KLA-SCARF AVR, captured in Lecroy HDO6104A oscilloscope with 250 M sampling rate [12,31]. The parameters for the experiment are $N = 50$, $d_{F_1} = 8$, $d_{F_2} = 8$, $d_{F_3} = 6$ and the private key is as follows.

$$b = \{0x03, 0x01, 0x1e, 0x11, 0x05, 0x06, 0x1a, 0x0e, 0x13, 0x01, 0x28, 0x23, 0x10, 0x29, 0x22, 0x0c, 0x07, 0x08, 0x0b, 0x15, 0x1b, 0x25, 0x2e, 0x2c, 0x18, 0x21, 0x17, 0x2f, 0x19, 0x04, 0x30, 0x00, 0x02, 0x0f, 0x27, 0x2d, 0x12, 0x2a, 0x2b, 0x14, 0x1c, 0x1f, 0x26, 0x20\}$$

We choose these values as we considered them to be suitable in the experimental environment. The first 16 entries of b represent F_1, the next 16 values represent F_2 and the rest of the values represent F_3.

The first step for analysis is discovering a reference trace R by SPA (Figure 3). The length of R is calculated by dividing the full trace length by the total number of operations. After that, the correlation coefficient can be calculated from the trace using the reference. Figure 1 is a part of the result containing the high peaks and the following intervals. There are two indices tagged on each peak, one represents an order of the high peak and the other is a distance between the previous high peak. The 31th peak has different distance than others, so the first degree where coefficient is -1 is $50 - 31 = 19 = 0x13$. With this process, we can recover F_1, F_2, F_3, and the private key.

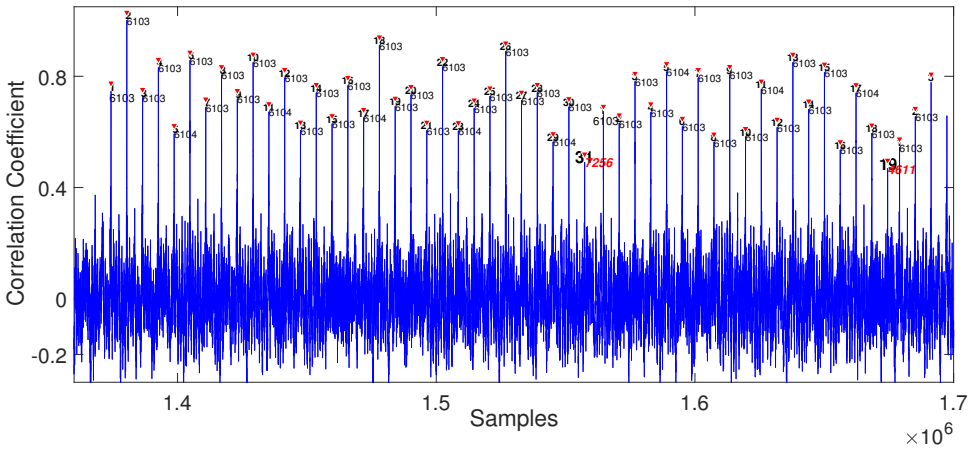


Figure 1. Result: High peaks of the addition and a distance value between the peaks.

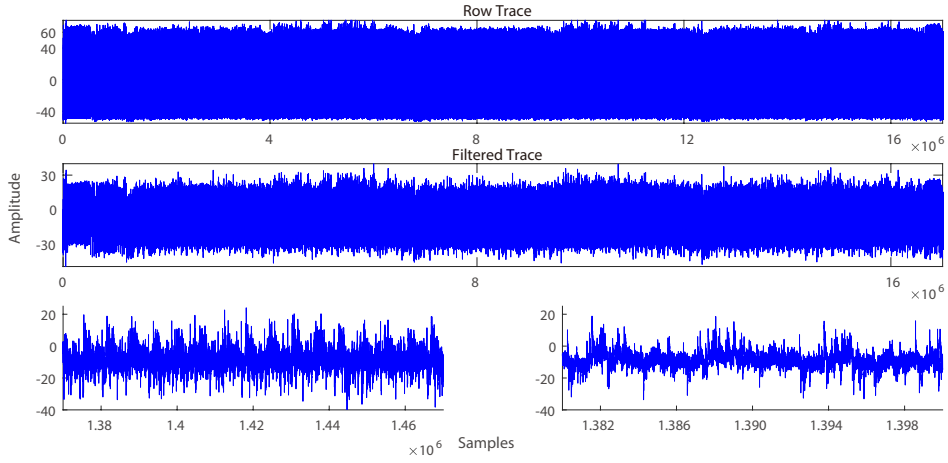


Figure 2. Full trace of NTRU Open Source: The top figure is a raw trace, the middle figure is a filtered trace, and the bottom two are enlarged figure of the filtered trace.

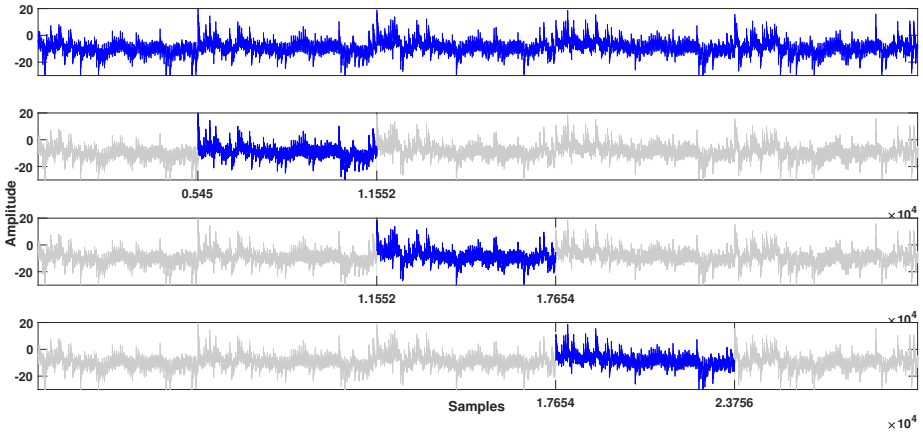


Figure 3. Figuring the reference trace: The three set of an addition operation.

3.2. NTRUEncrypt

Representing Polynomials

In the NTRUEncrypt, the polynomial is represented as the coefficients in order. For example, $F(x) = x^3 + x - 1$ stored as $F=\{-1, 1, 0, 1\}$. Before the polynomial multiplication of cipher-text and private key, there are steps to compute $f = pF + 1$.

Polynomial Multiplication

The the Equation (3) operates using the grade school multiplication. Unlike NTRU Open Source, the polynomial multiplication operates separately. These steps are described in Algorithm 3.

Algorithm 3 Decryption in NTRUEncrypt

Require: Trinary polynomial $F \in \mathcal{L}_f$, cipher-text $e \in \mathcal{R}$
Ensure: message $m = f \cdot e \pmod{q}$
1: **for** $0 \leq i < N$ **do**
2: $f_i \leftarrow F_i \times p$
3: **end for**
4: $f_0 \leftarrow f_0 + 1$
5: **for** $0 \leq j < N$ **do**
6: $t_j \leftarrow e_0 \times f_j$
7: **end for**
8: **for** $1 \leq i < N$ **do**
9: $t_{i+N-1} \leftarrow 0$
10: **for** $0 \leq j < N$ **do**
11: $t_{i+j} \leftarrow t_{i+j} + e_i \times f_j$
12: **end for**
13: **end for**
14: $t_{2N-1} \leftarrow 0$
15: **for** $0 \leq i < N$ **do**
16: $m_i \leftarrow (t_i + t_{i+N}) \pmod{q}$
17: **end for**
18: **return** m

3.2.1. Proposed Method

The proposed method exploits the power consumption of steps 1 to 3 and steps 5 to 13 in Algorithm 3 to recover the trinary polynomial F. When F get recovered, the private key polynomial f is computed by $f = pF + 1$, where p is a public value. The relative order of coefficients -1 is discovered by analyzing the steps 1 to 3 operation. Because F is a trinary polynomial, a constant value

p is multiplied by three values $-1, 0,$ and 1. Since most of the processor apply 2's complement method to express negative value, a hamming weight of -1 is bigger than others. Thus we can observe the high peaks in the power consumption trace when the -1 is operated. Note that, the proposed analysis depends on the operation of the processor. Thus, if the processor uses another method to represent negative value, the proposed analysis should consider such circumstances.

The next step, the relative orders of the coefficient 0 are known from 5 to 13 steps which are the polynomial multiplication of cipher-text e and private key f. The power consumption when calculating the coefficient of the cipher-text and 0 will be lower than other calculation processes. This portion where the power consumption is low is referred to as low peak. Therefore, after finding the relative position from $0, 1$ to -1, and combining this result with the information of 0s then F is completed recovered. Finally, we can get f by computing $pF + 1$.

3.2.2. Experiment

Figure 4a is a full trace of the NTRUEncrypt porting on the KLA-SCARF AVR and is captured with a Lecroy HDO6104A oscilloscope at a 250 M sampling rate [12,31]. The parameters for the experiment are $N = 49$, $p = 3$, $q = 2048$, and a private key is as follows.

$$f = \{1, 1, 1, 1, 1, 0, 1, 0, -1, -1, 1, 0, 0, 1, 1, -1, -1, 1, 0, 0, 0, -1, 0, -1, -1, 1, -1, 1, 0, 0, 0, -1, 0, 0, -1, 0, 1, 0, 1, -1, 0, 0, -1, -1, 1, 1\}$$

The p and q follow the proposed parameter but N is smaller than the standard because of the experimental environment.

Figure 4c depicts the power consumption of steps 1 to 3 in Algorithm 3. As mentioned above, the high peaks represent the moment when p is multiplied by -1. Also, in the Figure 4c, there are the low peaks related to the coefficient 0 and 1. Thus the relative orders of -1 and others can be recovered by analyzing Figure 4c.

The following process is to recover the coefficients 0. For each coefficient of the cipher-text, there are N multiplications with the private key. During the N operations, the operation of the private key 0 appears in the same order, so the low peaks appear regularly on the whole power trace (Figure 4a). To recover the degree, we should classify a set of multiplications by SPA among the trace. The multiplication between cipher-text and private key occurs after computing pF, and the total recovered number of multiplications is N^2. To reduce the noise, one can average multiple power consumption trace. Figure 5 illustrates the average of 10 traces. Figure 4b is an enlarged plot of four low peaks to deduce that peaks are identified. Lastly, with the three coefficients recovered from the analysis, the private key f is obtained.

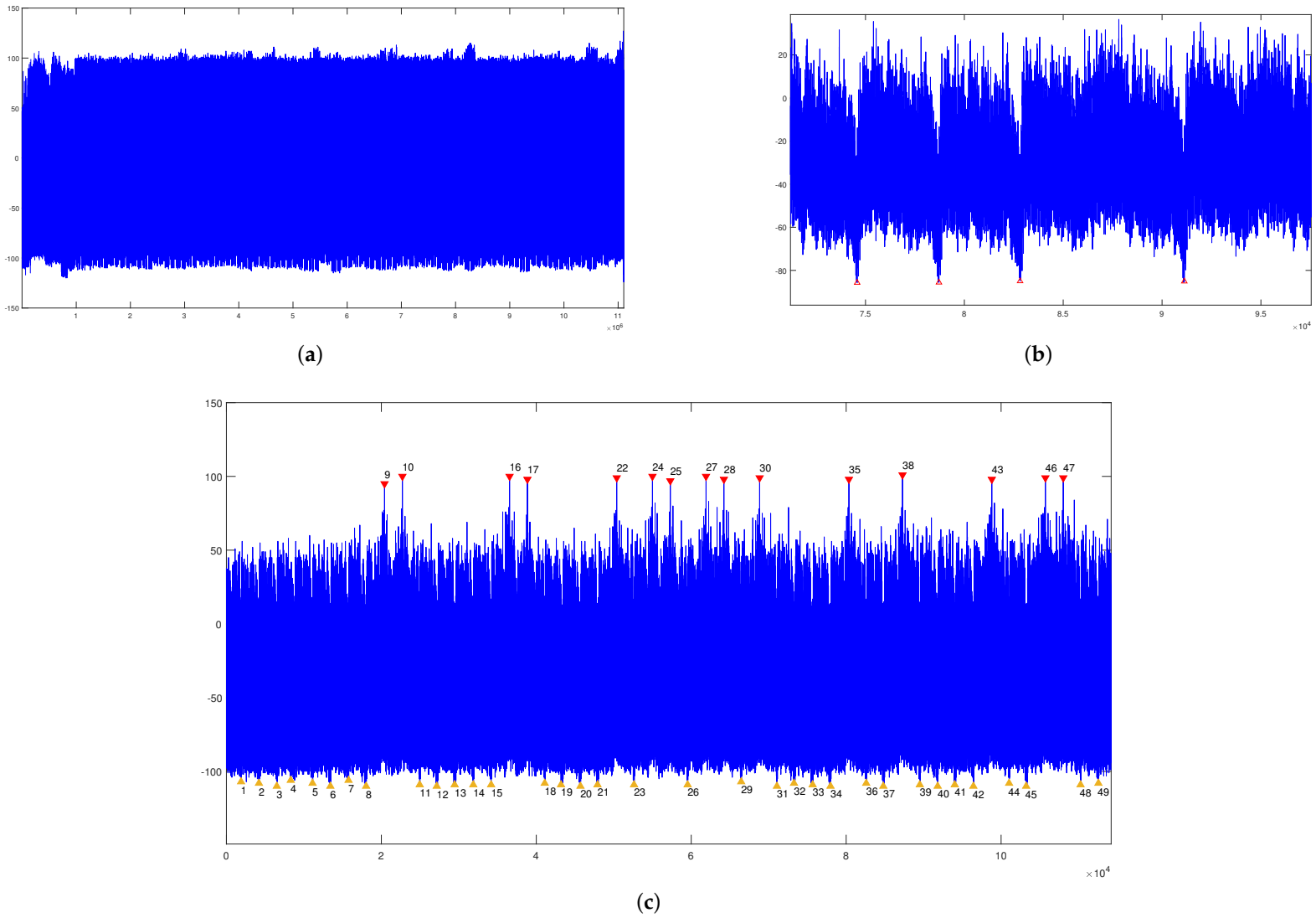


Figure 4. (**a**) Full trace of NTRUEncrypt; (**b**) Enlarged Low Peaks; (**c**) Result of SPA against $f = pF + 1$.

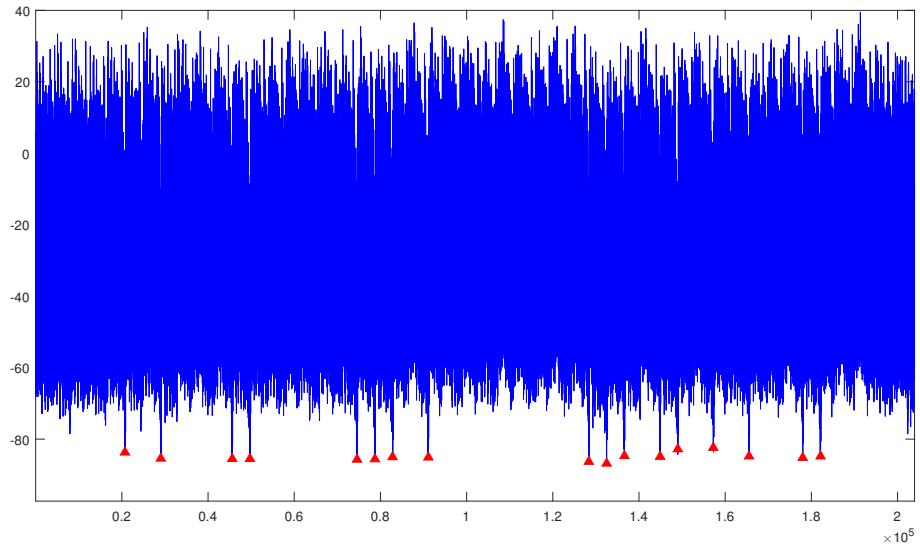


Figure 5. The Average of 10 Power Consumption Traces of Polynomial Multiplication using Grace School Multiplication Method.

4. Countermeasure

In this section, we propose a countermeasure for each of the two implementations. The proposed analysis on NTRU Open Source and NTRUEncrypt does not depend on the data information. Since we used a single power consumption trace, countermeasures to prevent DPA such as adding dummy operation and shuffling cannot prevent our attack.

4.1. Countermeasure against NTRU Open Source Implementation

Since the advantage of the original implementation is that computes both modular reduction $(x^N - 1)$ and polynomial multiplication, simultaneously, the countermeasure we propose also process both of the operation at the same time. Furthermore, the modified implementation has the same number of polynomial coefficient addition as the original implementation. The Algorithm 4 is a countermeasure for the polynomial multiplications described in Algorithm 2.

The Algorithm 4 is a method that precomputes the index i, where the cipher-text polynomial e_i will be added. For example, let 9 be the degree of the polynomial and let 2 be a coefficient of degree 0, then the original cipher-text polynomial coefficient addition performs as in Figure 6a. During the original iteration, the additions when $i = 0$ to 6 operate with different loop when $i = 7$ to 8. However, the addition in the proposed method operates in the same loop so there is no leakage for side channel analysis. The proposed method first finds the index of cipher-text which is added to the middle index of the result array, then the addition operates simultaneously as in Figure 6b.

Algorithm 4 Countermeasure of NTRU Open Source Proeject

Require: cipher-text polynomial $e \in R$ and coefficient location indices of private key b
Ensure: $H = F \cdot e \pmod{q}$

for $i = 0; i < N; i$++ **do**
  $t_i \leftarrow r$ ▷ r is a random value
end for
for $j = d_f + 1; j < 2d_f + 1; j$++ **do**
  $k \leftarrow b_j$
  $x \leftarrow \frac{N-1}{2} - k, y \leftarrow N - k$
  $t_{\frac{N-1}{2}} \leftarrow t_{\frac{N-1}{2}} + e_x$
  $x \leftarrow x + 1$
  for $i = 0; i < \frac{N-1}{2}; i$++, x++,y++ **do**
    $t_{\frac{N}{2}+i+1} \leftarrow t_{\frac{N}{2}+i+1} + e_x$
    $t_i \leftarrow t_i + e_y$
  end for
end for
for $i = 0; i < N; i$++ **do**
  $t_i \leftarrow -t_i$
end for
for $j = 0; j < d_F + 1; j$++ **do**
  $k \leftarrow b_j$
  $x \leftarrow \frac{N-1}{2} - k, y \leftarrow N - k$
  $t_{\frac{N-1}{2}} \leftarrow t_{\frac{N-1}{2}} + e_x$
  $x \leftarrow x + 1$
  for $i = 0; i < \frac{N-1}{2}; i$++, x++,y++ **do**
    $t_{\frac{N}{2}+i+1} \leftarrow t_{\frac{N}{2}+i+1} + e_x$
    $t_i \leftarrow t_i + e_y$
  end for
end for
for $i = 0; i < N; i$++ **do**
  $H_i \leftarrow t_i - r \pmod{q}$
end for
return H

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
+	+	+	+	+	+	+	+	+
e_7	e_8	e_0	e_1	e_2	e_3	e_4	e_5	e_6
$i = 7$	$i = 8$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$

(**a**)Original Implementation Iteration

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
+	+	+	+	+	+	+	+	+
e_7	e_8	e_0	e_1	e_2	e_3	e_4	e_5	e_6
$i = 0$	$i = 1$	$i = 2$	$i = 3$	$step\ 7$	$i = 0$	$i = 1$	$i = 2$	$i = 3$

(**b**)Proposed Implementation Iteration

Figure 6. Iteration of Addition.

4.2. Countermeasure against NTRUEncrypt Implementation

The proposed countermeasure in this paper uses three tables initialized with a random number. The countermeasure prevents not only proposed attack in this paper but also the other previous attacks with a decreased number of computations compared to the submitted implementation. The NTRUEncrypt first calculates the private key $f(= pF + 1)$ then multiplies $f \cdot e$. However, the proposed countermeasure uses a trinary polynomial F, and computes $p \times F \cdot e$ and adds e to decrypt the cipher-text. This is expressed in Equation (8).

$$m = f \cdot e = (pF + 1) \cdot e = pe \cdot F + e \tag{8}$$

To compute $f \times e$, the proposed countermeasure first computes $p \times e$ and temporarily save the value. After that, it updates the output of computation in the three tables according to $-1, 0, 1$ from the trinary polynomial F. Then the table of the coefficient -1 is subtracted from the table of the coefficient 1. During the accumulation, a start index is sought by $(i + j) \pmod N$ to process $\pmod{x^N - 1}$ at the same time for increased efficiency.

For the side channel resistance, by encoding the trinary polynomial of the private key $F' = enc(F)$ at the storage step, it relieves the difference in power consumption coming from loading -1 and $0, 1$. The encoding function enc is chosen by considering the physical property. The proposed countermeasure uses an encoding function as $enc(-1) = 1, enc(0) = 2$, and $enc(1) = 4$, to have the same hamming weight. Then the trinary polynomial would be represented with $1, 2$, and 4 at the proposed algorithm. Algorithm 5 describes the above procedure.

Algorithm 5 Countermeasure Applied Decryption of NTRUEncrypt

Require: Trinary polynomial F' an encoding of $F \in \mathcal{L}_f$, cipher-text $e \in \mathcal{R}$
Ensure: message $m = f \cdot e \pmod q$

1: **for** $0 \leq i < N$ **do**
2: $\quad PE_i \leftarrow p \times e_i$
3: $\quad T_i[1] \leftarrow r$ ▷ r is a random
4: $\quad T_i[2] \leftarrow r$
5: $\quad T_i[4] \leftarrow r$
6: **end for**
7: **for** $0 \leq i < N$ **do**
8: $\quad$ **for** $0 \leq j < N$ **do**
9: $\quad\quad T_{i+j \pmod N}[F'_i] \leftarrow T_{i+j \pmod N}[F'_i] + PE_j$
10: $\quad$ **end for**
11: **end for**
12: **for** $0 \leq i < N$ **do**
13: $\quad m_i \leftarrow (T_i[4] - T_i[1] + e_i) \pmod q$
14: **end for**
15: **return** m

At the final step 13, a subtraction of 1 and -1, an addition of e and $\pmod q$ could be processed at once. Since q is a power of 2 in the proposed parameters, $\pmod q$ can be operated as AND ($\wedge$). In Algorithm 5, different tables are accessed according to the coefficient of F. However, since the same operation is performed regardless of the coefficients, it can be considered that there is no difference in the power consumption which depends on $-1, 0,$ and 1. Furthermore, as the three tables are initialized with the same random number (steps 3 to 5), the algorithm also prevents SPA and CPA proposed in [7]. SPA can be prevented by initializing the array to hold the result with non-zero values and removing the operations that add zero and non-zero values. Also, choosing the non-zero value for initial as random, the algorithm is protected from CPA because the intermediate value cannot be guessed. When

three tables initialized with the same random non-zero value, the random values can be removed without additional computation, as in step 13.

4.3. Implementation of Countermeasure

4.3.1. Comparison of the NTRUEncrypt

The computational cost of the proposed countermeasure on NTRU Open Source is similar to the unprotected version, only with additional precomputations. Moreover, it is hard to compare the NTRU Open Source and NTRUEncrypt because of the different process of the private key multiplication. Thus, we only compare the computational cost and memory of the protected and unprotected version of NTRUEncrypt. Table 1 includes the number of initialization, addition, and multiplication costs when the degree is N. Since the computational cost of subtraction is similar to that of the addition, we include both the numbers of subtractions and the numbers of additions. The computation to find the start degree $(i + j) \pmod N$ is not included.

Table 1. Comparison of Operation between Unprotected and Protected NTRUEncrypt.

	Unprotected	**Protected**
Initial	N	$3N$
Add/Sub	N^2	$N^2 + 2N$
Mul	$N^2 + N$	N

As shown in Table 1, the total number of computational steps necessary to calculate $f \cdot e$ is reduced when applying our countermeasure. Moreover, the number of multiplications is reduced to square root of the original. Also, the comparison of the memory usage is presented in Table 2. Since NTRUEncrypt use 16 bit as a word size, Table 2 refers to the multiplication of a word size and the number of used arrays.

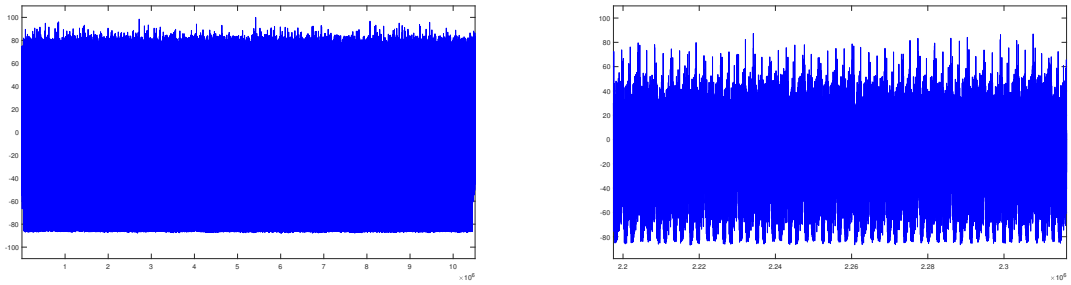
Table 2. Comparison of the Memory Usage between Unprotected and Protected NTRUEncrypt.

	Unprotected	**Protected**
RAM (bytes)	$6N$	$12N$

Although the number of used arrays is doubled compared to that used in the unprotected NTRUEncrypt, the total computational cost(number of computational steps) is reduced from $2N^2 + 2N$ to $N^2 + 6N$, where N is at least 443. Consequently, considering the computational costs and the memory size, Algorithm 5 is more efficient compared to Algorithm 3 and has side channel resistance.

4.3.2. Result of the Countermeasure Implementation

Figure 7a is the full trace of the Algorithm 5 implemented in the same environment as the analyzed traces. Since the countermeasure uses a table for all coefficients in the private key, the power consumption difference depending on the private key is not exposed and this is observed in Figure 7b.



(**a**) (**b**)

Figure 7. Iteration of Addition. (**a**) Full Trace of Countermeasure; (**b**) Enlarged Trace of Countermeasure.

5. Conclusions

Although the cryptosystem is proven to be secure, the security of its implementation must be considered as the devices may expose side channel leakages. Since PQC cryptosystems are implemented in classical computers, side channel must be considered. In this paper, we analyzed the two versions of NTRU implementation – NTRUEncrypt and NTRU Open Source. By using a single power consumption obtained in the decryption, we were able to recover the private key on both implementations. Our attack is practical and powerful since it can be applied without constraints of the environment. We also proposed countermeasures for our attack. Our countermeasures not only prevent our proposed attack but also prevents the previous attack. Moreover, our countermeasures do not degrade its performance. In addition, as the NIST standardization project is still in process, every algorithm including NTRU may provide an updated optimized implementation. The proposed analysis is based on the implementation up to now but more optimized version might appear in the future.

Acknowledgments: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2017R1C1B2004583).

Author Contributions: All authors have contributed to this work. Soojung An and Suhri Kim analyzed the algorithm and drafted and revised the manuscript. Soojung An and Sunghyun Jin performed the experiment and analyzed the result. Soojung An, Suhri Kim, Sunghyun Jin, and HanBit Kim devised the countermeasure and HeeSeok Kim verified the analytical methods and supervised this work.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
2. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
3. Koblitz, N. Elliptic curve cryptosystems. *Math. Comput.* **1987**, *48*, 203–209. [CrossRef]
4. Chen, L.; Chen, L.; Jordan, S.; Liu, Y.K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; US Department of Commerce, National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016.
5. Hoffstein, J.; Pipher, J.; Silverman, J.H. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory, Proceedings of the Third International Symposiun, ANTS-III, Portland, OR, USA, 21–25 June 1998*; Springer: Berlin, Germany, 1998; Volume 1423, pp. 267–288.
6. Kocher, P.C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO '96, Proceedings of the 16th Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 1996*; Springer: Berlin, Germany, 1996; Volume 1109, pp. 104–113.
7. Lee, M.K.; Song, J.E.; Choi, D.; Han, D.G. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2010**, *93*, 153–163. [CrossRef]

8. Zheng, X.; Wang, A.; Wei, W. First-order collision attack on protected NTRU cryptosystem. *Microprocess. Microsyst.* **2013**, *37*, 601–609. [CrossRef]
9. Song, J.E.; Han, D.G.; Lee, M.K.; Choi, D.H. Power analysis attacks against NTRU and their countermeasures. *J. Korea Inst. Inf. Secur. Cryptol.* **2009**, *19*, 11–21.
10. Whyte, W.; Etzel, M. NTRU Open Source. 2017. Available online: https://github.com/NTRUOpenSourceProject/ntru-crypto (accessed on 19 October 2018).
11. Zhang, Z.; Chen, C.; Hoffstein, J.; Whyte, W. NTRUEncrypt. 2017. Available online: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions (accessed on 19 October 2018).
12. Choi, Y.; Choi, D.; Ryou, J. Implementing Side Channel Analysis Evaluation Boards of KLA-SCARF system. *J. Korea Insti. Inf. Secur. Cryptol.* **2014**, *24*, 229–240. [CrossRef]
13. Hoffstein, J.; Silverman, J. Optimizations for NTRU. In *Public-Key Cryptography and Computational Number Theory*; Walter de Gruyter: Warsaw, Poland, 2001; pp. 77–88.
14. Guariglia, E. Entropy and fractal antennas. *Entropy* **2016**, *18*, 84. [CrossRef]
15. Zmeskal, O.; Dzik, P.; Vesely, M. Entropy of fractal systems. *Comput. Math. Appl.* **2013**, *66*, 135–146. [CrossRef]
16. Zanette, D.H. Generalized Kolmogorov entropy in the dynamics of multifractal generation. *Phys. Stat. Mech. Its Appl.* **1996**, *223*, 87–98. [CrossRef]
17. Guariglia, E. Harmonic Sierpinski Gasket and Applications. *Entropy* **2018**, *20*, 714. [CrossRef]
18. Berry, M.V.; Lewis, Z.; Nye, J.F. On the Weierstrass-Mandelbrot fractal function. *Proc. R. Soc. Lond. A* **1980**, *370*, 459–484. [CrossRef]
19. Guariglia, E. Spectral analysis of the Weierstrass-Mandelbrot function. In Proceedings of the 2017 2nd International Multidisciplinary Conference on Computer and Energy Science (SpliTech), Split, Croatia, 12–14 July 2017; pp. 1–6.
20. Brier, E.; Clavier, C.; Olivier, F. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems—CHES 2004, Proceedingds of the 6th International Workshop, Cambridge, MA, USA, 11–13 August 2004*; Springer: Berlin, Germany, 2004; Volume 3156, pp. 16–29.
21. Kocher, P.; Jaffe, J.; Jun, B. Differential power analysis. In *Advances in Cryptology—CRYPTO' 99, Proceedingds of the 19th Annual International Cryptology Conference, Santa Barbara, CA, USA, 15–19 August 1999*; Springer: Berlin, Germany, 1999; Volume 1666, pp. 388–397.
22. Yarom, Y.; Falkner, K. FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014; Volume 1, pp. 22–25.
23. Inci, M.S.; Gulmezoglu, B.; Irazoqui, G.; Eisenbarth, T.; Sunar, B. Cache attacks enable bulk key recovery on the cloud. In *Cryptographic Hardware and Embedded Systems—CHES 2016, Proceedings of the 18th International Conference, Santa Barbara, CA, USA, 17–19 August 2016*; Springer: Berlin, Germany, 2016; Volume 9813, pp. 368–388.
24. Martinasek, Z.; Zeman, V.; Trasy, K. Simple electromagnetic analysis in cryptography. *Int. J. Adv. Telecommun. Electrotech. Signals Syst.* **2012**, *1*, 13–19. [CrossRef]
25. Rooney, C.; Seeam, A.; Bellekens, X. Creation and Detection of Hardware Trojans Using Non-Invasive Off-The-Shelf Technologies. *Electronics* **2018**, *7*, 124. [CrossRef]
26. Coron, J.S.; Goubin, L. On boolean and arithmetic masking against differential power analysis. In *Cryptographic Hardware and Embedded Systems—CHES 2000, Proceedings of the Second International Workshop, Worcester, MA, USA, 17–18 August 2000*; Springer: Berlin, Germany, 2000; Volume 1965, pp. 231–237.
27. Yuan, J.S.; Lin, J.; Alasad, Q.; Taheri, S. Ultra-Low-Power Design and Hardware Security Using Emerging Technologies for Internet of Things. *Electronics* **2017**, *6*, 67. [CrossRef]
28. Cao, Y.; Zhao, X.; Ye, W.; Han, Q.; Pan, X. A Compact and Low Power RO PUF with High Resilience to the EM Side-Channel Attack and the SVM Modelling Attack of Wireless Sensor Networks. *Sensors* **2018**, *18*, 322. [CrossRef] [PubMed]
29. Silverman, J.H.; Whyte, W. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In *Topics in Cryptology—CT-RSA 2007, Proceedings of the Cryptographers' Track at the RSA Conference 2007, San Francisco, CA, USA, 5–9 February 2007*; Springer: Berlin, Germany, 2007; Volume 4377, pp. 208–224.

30. Howgrave-Graham, N.; Silverman, J.H.; Singer, A.; Whyte, W. NAEP: Provable Security in the Presence of Decryption Failures. *IACR Cryptol. ePrint Arch.* **2003**, *2003*, 172.
31. LeCroy, T. HDO6000A High Definition. Available online: http://cdn.teledynelecroy.com/files/pdf/hdo6000a-oscilloscopes-datasheet.pdf (accessed on 11 October 2018).

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).