2012 Seventh International Conference on Availability, Reliability and Security

A Scan-Based Side Channel Attack on the NTRUEncrypt Cryptosystem

Abdel Alim Kamal
Electrical and Computer Engineering Department
Concordia University
Montreal, Qc, Canada
a_kamala@ece.concordia.ca

Amr M. Youssef
Concordia Institute for Information Systems Engineering
Concordia University
Montreal, Qc, Canada
youssef@ciise.concordia.ca

***Abstract*—Scan-based Design-for-Test (DFT) is a widely deployed technique for testing hardware chips. Using this approach, all flip-flops in the design under test are connected to a scan chain where their states can be scanned out through this chain during the testing phase. Scan-based side channel attacks exploit the information obtained by analyzing the scanned data in order to retrieve secret information from cryptographic hardware devices that are designed with this testability feature.**

The NTRU encryption algorithm (NTRUEncrypt) is a parameterized family of lattice-based public key cryptosystems which has recently been accepted to the IEEE P1363 standards under the specifications for lattice-based public-key cryptography. In this paper, we present a scan-based side channel attack on NTRUEncrypt hardware implementations that employ scan based DFT techniques. Our attack determines the scan chain structure of the polynomial multiplication circuits used in the decryption algorithm which allows the cryptanalyst to efficiently retrieve the secret key.

***Keywords*-NTRU; public key cryptography; scan-based attacks; side channel attacks.**

I. Introduction

Scan-based Design-For-Test (DFT) [1] is a popular technique for validating the functionality of integrated circuits at fabrication time and providing on-chip debugging capabilities in the field. When using this approach, all flip-flops in the design under test are tied in one or more scan chains through which the states of theses flip-flops can be scanned out during the testing phase through the Joint Test Action Group (JTAG) boundary scan interface [2]. A JTAG interface is a special four/five-pin serial interface with the following pins: TDI (Test Data In), TDO (Test Data Out), TCK (Test Clock), TMS (Test Mode Select), and optional TRST (Test Reset). TMS selects between normal mode and test mode. TRST is the reset signal for the test controller. During testing mode, test vectors can be scanned in via the TDI pin and internal flip-flops can be scanned out via the TDO pin. As shown in Figure 1, a scan flip-flop is a flip-flop with a MUX at the input. In normal mode, it works like a normal flip-flop. In test mode, as shown in Figure 2, all scanned flip-flops are disconnected from the combinational circuit and connected to each other in a scan chain where their contents can be scanned in and out.

While scan-based DFT improves the quality of testing, it also opens a powerful side channel against hardware implementations of cryptographic devices that utilize this technique. Despite the fact that the internal structure of the scan chain is usually not known to attackers, exploiting the information obtained from analyzing the scanned data allows cryptanalysts to ascertain this structure and retrieve the secret key from the cryptographic hardware devices implementing various cryptographic algorithms such as DES [3], AES [4], [5], RSA [6], ECC [7], and stream ciphers [8], [9].

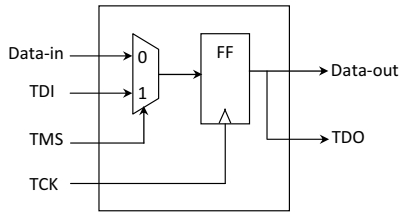

Figure 1. An illustration for a scan flip-flop (SFF)

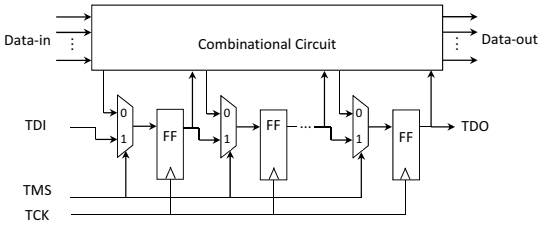

Figure 2. An illustration for a scan chain

The NTRU encryption algorithm (NTRUEncrypt) [10], [11] is a parameterized family of lattice-based public key cryptosystems. Both the encryption and decryption operations in NTRUEncrypt are based on simple polynomial multiplication which makes it very fast and compact compared to other public key alternatives such as RSA, and elliptic-curve-based systems [12]. The NTRU system has been accepted to the IEEE P1363 standards under the specifications for lattice-based public-key cryptography (IEEE P1363.1). In the past few years, the security of NTRUEncrypt against mathematical attacks had been analyzed by many researchers (e.g., [13]-[14]). Different classes of side channel attacks

978-0-7695-4775-6/12 $26.00 © 2012 IEEE
DOI 10.1109/ARES.2012.14

against NTRUEncrypt and their countermeasures were also considered [15], [16], [17], [18].

In this paper, we present a scan-based side channel attack against hardware implementations of NTRUEncrypt. In general, all scan-based side channel attacks can be viewed as a kind of differential cryptanalysis where attackers take advantages of the scan chains to observe the bit changes between pairs of chosen plaintexts/ciphertexts so as to identify the secret keys. More precisely, scan-based attacks can be classified into two classes [19]: constant based attacks (CBAs) and fixed-Hamming-distance-based attacks (FHDAs). Constant-based attacks take advantages of the fact that in the encryption/decryption process, the contents of some special registers are independent of the input. By using several different inputs and scanning out the contents at different times of the cryptographic operation, the internal registers of the cryptographic device can be easily identified. Then by setting these registers to specific states through scan operations, the complexity of secret key recovery can be reduced. In FHDAs, several pairs of relevant plaintexts/ciphertexts are applied and then, for each pair, the number of different bits in the output are counted so as to recover the secret key. Our attack can be seen as a type of FHDAs where we focus on determining the scan chain structure of the polynomial multiplication circuit in the decryption algorithm and then utilize the Hamming weight information of some particular registers, which can be obtained via the JTAG interface, to efficiently retrieve all the coefficients of the secret key polynomial.

The rest of the paper is organized as follows. Description of NTRUEncrypt, its hardware implementation options and some of the ideas behind our attack are briefly described in the next section. The details of the proposed attack is provided in section III and the simulation results are given in section IV. Finally, our conclusion is presented in section V.

II. Preliminaries

A. Description of the NTRUEncrypt

NTRUEncrypt is a lattice-based public key cryptosystem that is parameterized by three integers: (N, p, q), where N is prime, gcd(p, q) = 1 and $p << q$. Let R, R_p, and R_q be the polynomial rings

$$R = \frac{\mathbb{Z}[x]}{x^N - 1}, R_p = \frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^N - 1}, R_q = \frac{\mathbb{Z}/q\mathbb{Z}[x]}{x^N - 1}.$$

The product of two polynomials

$$\begin{aligned} a(x) &= a_0 + a_1 x + a_2 x^2 + ... + a_{N-1} x^{N-1} \in R, \\ b(x) &= b_0 + b_1 x + b_2 x^2 + ... + b_{N-1} x^{N-1} \in R \end{aligned}$$

is given by

$$c(x) = a(x) \star b(x),$$

where

$$c_k = \sum_{i+j=k \pmod N} a_i b_{k-i}.$$

For any positive integers d_1 and d_2, let $\tau(d_1, d_2)$ denote the set of ternary polynomials given by

$$\left\{ a(x) \in R : \begin{array}{l} \text{a(x) has } d_1 \text{ coefficients equal to 1,} \\ \text{a(x) has } d_2 \text{ coefficients equal to -1,} \\ \text{all other coefficients equal to 0.} \end{array} \right\}$$

In what follows, we briefly describe the key generation, encryption and decryption operations in NTRUEncrypt [10].

1) Key Generation:

- Choose a private $f(x) \in \tau(d_f, d_f - 1)$ that is invertible in R_q and R_p.
- Choose a private $g(x) \in \tau(d_g, d_g)$.
- Compute $F_q(x) = f^{-1}(x)$ in R_q and $F_p(x) = f^{-1}(x)$ in R_p.
- Compute $h(x) = f_q(x) \star g(x)$ in R_q.

The polynomial $h(x)$ is the user's public key. The corresponding private key is the pair $(f(x), F_p(x))$. The following steps denote the encryption operations for plaintext $m(x) \in R_p$.

2) Encryption:

- Choose a random ephemeral key $r(x) \in \tau(d_r, d_r)$.
- Compute the ciphertext $e(x) = pr(x) \star h(x) + m(x)$ mod q.

3) Decryption:

- Compute $a(x) = f(x) \star e(x)$ mod q.
- Compute $b(x)$ = Centerlift $(a(x))$ such that its coefficients lie in the interval $(-q/2, q/2]$.
- Compute $m = F_p(x) \star b(x)$ mod p.

B. Hardware Implementation options for NTRUEncrypt

Throughout the rest of the paper, we focus on NTRUEncrypt with the widely used parameters: $p = 3$ and q in the form of 2^n [11]. We also assume that the attacker has access to the high level timing diagram of the target hardware implementation and that the secret key is stored in secure memory that cannot be accessed through the scan chain. It is also assumed that the attacker has direct access to the scan chains via the JTAG port or by breaking open the package and directly probing the buried JTAG ports [1].

From the description of the encryption and decryption operations above, it is clear that the most time/area consuming step in both operations is the convolution multiplication required to compute $r(x) \star h(x)$ and $f(x) \star e(x)$ during the encryption and decryption operations, respectively.

While there are several hardware implementations for NTRUEncrypt (e.g., see [16], [17], [20], [21], [22], and [23]), the majority of these implementations focus on optimizing either the time or the area required by theses convolution operations. In what follows, we focus on the decryption process because it is the only relevant part for our

cryptanalysis since the encryption module does not contain any secret information.

For low area and low power implementations, which are viable for resource constrained applications such as RFIDs and sensor nodes, the convolution multiplication, $a = f \star e \bmod q$, is usually performed in approximately N^2 clock cycles [20], [21] as shown in Figure 3. In this case, the computation for the coefficients of the polynomial a are done sequentially where the coefficient a_i, of size $\log_2(q)$, is obtained after $(i+1) \times N$ clock cycles, $0 \leq i < N$. For example, a_0 is calculated by accumulating (mod q) the results of serially multiplying $e_0 \times f_0, e_{N-1} \times f_1, e_{N-2} \times f_2, \cdots, e_1 f_{N-1}$ into a register initialized to zero where each multiply-and-add step is performed in one clock cycle.

For this kind of implementations, scan-based side channel attack can be used to recover the secret information in a straightforward way as follows: First, the attacker determines the locations of the flip flops of the ciphertext register e in the scan chain output by analyzing the bit difference between the scan chain output corresponding to the all zeroes ciphertext and the $N \times log_2(q)$ scan chain outputs corresponding to the ciphertext blocks with a (bit) Hamming weight equal to one. This step is performed right after the ciphertext loading operation.

In the next step of the attack, the attacker loads a ciphertext e in which all coefficients are set to 1. Since the value of a_0 is computed sequentially, then the intermediate values of a_0 after the t^{th} clock cycle of the convolution multiplication step are given by

$$\begin{aligned} a_0^{(t)} &= \sum_{j=0}^{t} f_j \times e_{(N-j) \bmod N} \bmod q \\ &= \sum_{j=0}^{t} f_j \bmod q, 0 \leq t < N. \end{aligned}$$

Consequently, the attacker can recover the value of the secret key by scanning out the intermediate values of a_0. Note that the attacker is able to determine the locations of the $log_2(q)$ bits corresponding to a_0 since the location of the bits corresponding to e are already determined in the previous step and the only bits of the scan chain output that change in this step would correspond to the a_0 bits since the bits corresponding to $a_i, 0 < i < N$ remain zeroes throughout the computation of a_0.

In hardware implementations of NTRUEncrypt that targets higher speed applications (e.g., see [16], [17], [22]), instead of storing all the coefficients of f, the locations of the nonzero $(2 \times d_f - 1)$ coefficients are stored. This allows the convolution multiplication to be performed, as shown in Algorithm 1 [23], in $(2 \times d_f - 1) \times N$ steps instead of N^2 steps (see Figure 4). Since the number of nonzero coefficients, $2 \times d_f - 1$, is typically much smaller than N, this implementation leads to a faster decryption speed.

Determining the corresponding locations of the register e in the scan chain output can be performed using the same approach above, i.e., by loading different ciphertexts with a Hamming weight equal to one and analyzing the

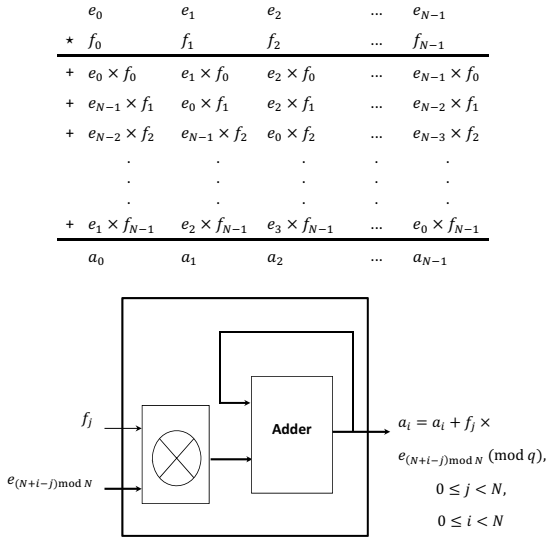

Figure 3. A typical low area implementation of the convolution multiplication $(f(x) \star e(x))$ in N^2 clock cycles. For each $0 \leq i < N$, j varies from 0 to $N-1$.

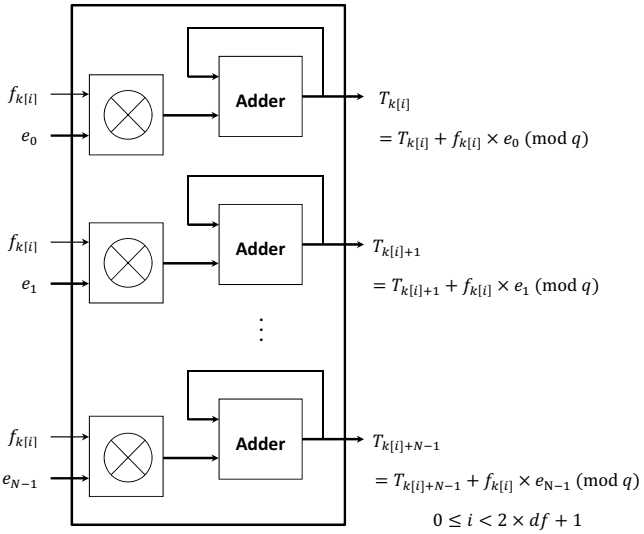

Figure 4. The convolution multiplication between the polynomials $f(x)$ and $e(x)$ in N clock cycles

output differences in the scan chain output corresponding to these ciphertexts and the one corresponding to the all zero ciphertext. On the other hand, determining the coefficients of the secret key polynomial, f, requires some deeper analysis which will be developed in the remaining sections of this paper.

III. The Proposed Scan-Based Attack

As mentioned above, when the convolution multiplication is implemented as shown in Figure 4, a straightforward application of scan-based attacks and trying to recover the

Algorithm 1 Fast Convolution Algorithm [23]

INPUT: An array k of d_f+1 locations for +1 and d_f locations for -1 representing f; e; N: the size of f and e
OUTPUT: $T = f \star e \bmod q$.
for $0 \leq l < 2N$ **do**
  $T_l \longleftarrow 0$
end for
for $1 \leq t \leq d_f + 1$ **do**
  for $0 \leq l < N$ **do**
    $T_{l+k[t]} \longleftarrow T_{l+k[t]} + e_l$
  end for
end for
for $d_f + 2 \leq t \leq 2d_f + 1$ **do**
  for $0 \leq l < N$ **do**
    $T_{l+k[t]} \longleftarrow T_{l+k[t]} - e_l$
  end for
end for
for $0 \leq l < N$ **do**
  $T_l \longleftarrow (T_l + T_{l+N}) \bmod q$
end for

full scan chain structure will not work because of the large number of flip-flops connected in the scan chain. Instead, in our attack, we focus on determining the relevant flip-flops in the scan chain structure of the polynomial multiplication circuit. The main idea of our attack is to distinguish the register T into two parts: T_R and T_L (the relevance of both parts will be explained below.) Then by single stepping through the $2d_f + 1$ clock cycles of the convolution multiplication step, and by recording the Hamming weight of T_R and T_L in each clock, the attacker can construct a system of linear equations, with the estimated positions of the non-zero elements of the secret keys as unknowns. This set of equations can be solved to obtain the set of possible keys. Then, the correct key can be determined by verifying the correctness of the decryption operation for a known plaintext using the keys in this set.

A. Summary of the attack

Conceptually, the steps of the attack can be summarized as follows:

1) Reset the chip and run it in normal decryption mode to load an all zero ciphertext into the register e. Resetting the circuit allows the attacker to reset all flip-flops (including the ones that belong to the control circuit) into the same initial state before each attack step. This is necessary in order to allow the attacker to calculate the difference in the Hamming weight of the target registers before and after each attack step.
2) Switch to test mode and scan out the bit stream pattern.
3) Repeat steps 1 and 2 using all the $N \times log_2(q)$ ciphertexts with a (bit) Hamming weight equal to one and compare the output differences in the scan chain output corresponding to these ciphertexts with the one corresponding to the all zeroes ciphertext.
   At the end of this step, the attacker is able to determine the locations corresponding to the ciphertext register, e, in the scan chain.
4) Load the chip with a ciphertext of all 1's in normal mode and clock the system one time to evaluate $T = f \times e$ as shown in Algorithm 1.
5) Switch to test mode and scan out the bit stream pattern. In this case, the register T can be distinguished into two parts: T_R which contains all 1's and T_L which contains 0's. Note that while the attacker can identify the group of bits that belong to each one of these two registers, the attacker cannot determine the exact location of these bits within these registers.
6) Clock the system in normal mode
7) Switch to test mode and scan out the bit stream to calculate the Hamming weight of the registers T_R and T_L.
8) Repeat steps 6-7 for $(2 \times d_f)$ times and record the Hamming weights of the registers T_R and T_L.
9) Use the Hamming weights obtained above as an input to Algorithms 2 and 3 to form a set of linear equations which can be solved to obtain the set of possible keys.
10) Determine the unique correct key by verifying the correctness of the decryption operation (using any arbitrary known plaintext-ciphertext pair obtained using the public key encryption process) for each one of the keys obtained in step 9 above.

B. Recovering the secret key

For $A = [a_0, a_1, ..., a_{N-1}]$ and $B = [b_0, b_1, ..., b_{N-1}]$, $a_i, b_i \in Z_q$, $0 \leq i < N$, the following notation will be used throughout the rest of the paper.

- $A_{i..j}$ denotes the vector $[a_i, a_{i+1}, ..., a_j]$ of length $j - i + 1$, $0 \leq i < j$.
- $\bar{A}_{i..j}$ denotes the vector $[(a_i + (q-1))\bmod q, (a_{i+1} + (q-1))\bmod q, ..., (a_j + (q-1))\bmod q] = [(a_i - 1)\bmod q, (a_{i+1} - 1)\bmod q, ..., (a_j - 1)\bmod q]$, $0 \leq i < j$.
- $\dot{A}_{i..j}$ denotes the vector $[(a_i - (q-1))\bmod q, (a_{i+1} - (q-1))\bmod q, ..., (a_j - (q-1))\bmod q] = [(a_i + 1)\bmod q, (a_{i+1} + 1)\bmod q, ..., (a_j + 1)\bmod q]$, $0 \leq i < j$.
  To illustrate the above notation, consider the following example where $A = [5, 0, 4, 2, 7]$ and $q = 0xF$ (in hexadecimal). Then $\bar{A}_{1..3} = [0 + (q-1)\bmod q, 4 + (q-1)\bmod q, 2 + (q-1)\bmod q] = [(0-1)\bmod q, (4-1)\bmod q)], (2 - 1)\bmod q = [0xE, 3, 1]$. Similarly, $\dot{A}_{0..2} = [5 - (q-1)\bmod q, 0 - (q-1)\bmod q, 4 - (q-1)\bmod q] = [5 + 1, 0 + 1, 4 + 1] = [6, 1, 5]$.
- $A^{(t)}$ denotes the value of the register A at time t.

- HW$(\cdot)$ denotes the Hamming weight of the enclosed argument.
- $A|B$ denotes the vector obtained from the concatenation of A and B.

1) Recovering the locations of the +1 elements in the secret key: As mentioned above, after determining the corresponding locations of the flip-flops corresponding to e in the scan chain, the attacker divides the flip-flops corresponding to the register T into two parts: T_L and T_R which contain, after the first step of convolution multiplication, all zeroes and all ones, respectively. Algorithm 2 is then used to determine the locations of the +1 elements in f relative to the location of the first +1 element. More precisely, Algorithm 2 outputs an array whose t^{th} element is equal to $(k[t]-k[0])$, $1 \leq t < d_f + 1$. As shown in the algorithm, A and B, are used to simulate the intermediate values of T_L and T_R, respectively, during the computation of $T = f \star e \bmod q$. By examining the Hamming weight information of T_L and T_R obtained from the scan chain output bit stream observed via the JTAG port, one can derive information about $k[t]-k[0]$ by going through all valid guesses for $k[t]$ (see the variable j in Algorithm 2) and choosing the value for which the Hamming weight of A and B match the corresponding one for T_L and T_R, respectively, at the corresponding time.

The steps of Algorithm 2 can be explained as follows. At $t = 1$, a ciphertext of all ones is circularly shifted by $k[0]$ elements and loaded into T (note that all ones at the bit level corresponds to $e_i = q - 1$.) This can be simulated by initializing $A_i = 0$ and $B_i = q - 1$, $0 \leq i < N$ (lines 3-6). At $t = 2$, the ciphetext polynomial is circularly shifted by $k[1]$ elements and added to T. Thus $(k[1]-k[0])$ elements of A will change from 0 to $q-1$ and $(N-(k[1]-k[0]))$ elements of B will change from $q-1$ to $= ((q-1)+(q-1)) \bmod q = q-2)$. The notation $\bar{A}_{i..j}$ and $\bar{B}_{i..j}$ (lines 14-15) is used to reflect these updates in A and B. Similarly, depending on the value of $k[2]-k[0]$, at $t = 3$, some elements of A change from $q-1$ to $q-2$ and from 0 to $q-1$ while some elements of B change from $q-2$ to $q-3$. This process continues in a similar way until $t = d_f + 1$. As shown in lines 8-17, we update $A_{0..N}$ and $B_{0..N}$ to $A_{0..N-j-1}|\bar{A}_{N-j..N-1}$, and $\bar{B}_{0..N-j-1}|B_{N-j..N-1}$, respectively, $1 \leq j < N$ to simulate the above process. For each j, we calculate $\mathsf{HW}(A_{0..N-j-1}|\bar{A}_{N-j..N-1}) - \mathsf{HW}(A)$. This step is repeated after incrementing j until the difference between these Hamming weights matches the value of $\mathsf{HW}({T_L}^{(t)}) - \mathsf{HW}({T_L}^{(t-1)})$, $2 \leq t \leq d_f + 1$. Then we set $k[t] - k[0] = j$, $1 \leq t \leq d_f + 1$. Note that while observing the changes in the Hamming weight of A is enough to allow Algorithm 2 to calculate the elements of S_1, we still update B since it is needed by Algorithm 3 which is used to determine all possible valid locations of the -1's in f.

Algorithm 2 Recovery of the locations of the +1's in the private key polynomial f

1: INPUT: $\mathsf{HW}({T_L}^{(t)})$, $1 \leq t \leq d_f + 1$.
2: OUTPUT: An array S_1 where $S_1[t] = k[t] - k[0]$ and $k[t]$ denotes the location of the t^{th} +1 elements in f, $1 \leq t < d_f + 1$.
3: **for** $0 \leq i < N$ **do**
4: $\quad A_i \longleftarrow 0$
5: $\quad B_i \longleftarrow q - 1$
6: **end for**
7: $j \longleftarrow 0$
8: **for** $2 \leq t \leq d_f + 1$ **do**
9: $\quad \mathsf{diff} \longleftarrow 0$
10: $\quad$ **while** $(\mathsf{diff} \neq \mathsf{HW}({T_L}^{(t)}) - \mathsf{HW}({T_L}^{(t-1)}))$ **do**
11: $\quad\quad j \longleftarrow j + 1$
12: $\quad\quad \mathsf{diff} \longleftarrow \mathsf{HW}(A_{0..N-j-1}|\bar{A}_{N-j..N-1}) - \mathsf{HW}(A)$
13: $\quad$ **end while**
14: $\quad A \longleftarrow A_{0..N-j-1}|\bar{A}_{N-j..N-1}$
15: $\quad B \longleftarrow \bar{B}_{0..N-j-1}|B_{N-j..N-1}$
16: $\quad S_1[t-1] \longleftarrow j$
17: **end for**
18: **return** S_1, A, B

2) Recovering the locations of the -1 elements in the secret key: Algorithm 3 receives A, B, the set of +1 locations, S_1, evaluated by Algorithm 2 and the Hamming weights $\mathsf{HW}({T_L}^{(t)})$ and $\mathsf{HW}({T_R}^{(t)})$ obtained by analyzing the scan out bit stream, $d_f + 2 \leq t \leq 2 \times d_f + 1$. The algorithm operates in a way similar to Algorithm 2 except that in this case, each element in S_2 represents a list of possible valid locations for the -1 elements as opposed to a single element for the case of S_1. Also, in this case, the content of register e is subtracted from register T, instead of addition in Algorithm 2 (see lines 11-15 in Algorithm 1.)

Lines 7-12 are used to initialize j to the starting values for our guesses for the location of the t -1 element which correspond to $k[t]$, $d_f + 2 \leq t \leq 2d_f + 1$. These steps can be explained by noting that S_2 represents the list of valid locations for the -1 elements and by the fact that $k[t] > k[t-1]$ for $d_f+2 \leq t \leq 2d_f+1$ (in other words, the location of the t^{th} -1 element has to be greater than the location of the $(t-1)^{th}$ -1 element in f). At $t = d_f + 2$, j starts from 0 since at this stage, the attacker cannot yet determine the exact value of $k[0]$ (Also, the j=k[0] step is skipped since two keys cannot be assigned to the same location.) More precisely, according to the attacker's knowledge at this step, $0 \leq k[0] < N - S_1[d_f]$. In lines 13-36, the updates of A and B can follow two different pathes depending on whether $k[d_f + 2]$ is less than $k[0]$ or greater than it (i.e., whether the first non-zero element in f is +1 or -1). Lines 26-31 are used to determine values of j that represent valid guesses

for the location of the -1's to be appended to the list S_2. This is performed by comparing the Hamming weight of the simulated registers A and B with the Hamming weight of T_L and T_R which can be calculated by observing the scan out data. In particular, in lines 29-30, we add j to the list $S_2[t-d_f-1]$, $d_f+2 \leq t \leq 2d_f+1$, as a possible solution and update the values of A and B and append it to list $L[t-d_f-1]$.

Simple analysis of Algorithm 2 and Algorithm 3 reveal that their run time complexity is given by $\mathcal{O}(d_f \times N)$ and $\mathcal{O}(d_f \times N^3)$, respectively.

Example 1. Consider a toy version of NTRUEncrypt cryptosystem with parameters (N,p,q,d_f)=(7,3,16,2) and with a private key

$$f = [0, -1, -1, 1, 0, 1, 1].$$

Thus the locations of the nonzero coefficients of f are $k = [3, 5, 6, 1, 2]$ where the first $d_f + 1$ values denote the locations of +1's in the key and the last d_f values denote the locations of -1's.

As explained above, the T register is initialized with all 0's before starting the convolution computation of $f \star e \bmod q$. At $t = 1$, T is loaded with a copy of the ciphertext after being circularly shifted by $k[0] = 3$ coefficients according to Algorithm 1. Recall that the attacker does not know $k[0]$. By scanning out the bit stream pattern via JTAG port, the attacker can distinguish T into T_L and T_R. Assume that the attacker observed the Hamming weights of of T_L and T_R as shown in Table I (Obviously, at $t = 1$, these Hamming weights are always going to be 0 and $N \times log_2(q)$, respectively.) At $t = 2$, the register T_L is changed by two coefficients which implies that $k[1] - k[0] = 2$. While the attacker cannot associate the bits corresponding to T in the scan chain output with the individual coefficients in T_L and T_R, the attacker can still calculate the value of $k[1] - k[0]$ by using Algorithm 2 which simulates the content of $T = T_L|T_R$ (using A and B) for different possible values of $k([1] - k[0])$ (the variable j in lines 7-17) until the change in the Hamming weight of A, i.e., $HW(A_{0..N-j-1}|\bar{A}_{N-j..N-1}) - HW(A)$ in the simulation, matches $\mathsf{HW}(T_L{}^{(2)}) - \mathsf{HW}(T_L{}^{(1)})$ obtained from the scan out bit stream pattern. Following the same strategy, the attacker recovers the distances between $k[t]$ and $k[0]$ for $2 \leq t < d_f + 1$. In this example, the attacker recovers the set $S_1 = \{k[1] - k[0] = 2, k[2] - k[0] = 3\}$ which defines the distances between the locations of the +1's in the key.

The top table in Figure 5 shows the corresponding input, output and intermediate computational results of Algorithm 2. To recover the locations of -1's in f, the attacker continues scanning out the bit stream pattern and, using Algorithm 3, calculates $\mathsf{HW}(T_L{}^{(t)})$ and $\mathsf{HW}(T_R{}^{(t)})$. Then the attacker calculates $\mathsf{HW}(T_L{}^{(t)}) - \mathsf{HW}(T_L{}^{(t-1)})$ and $\mathsf{HW}(T_R{}^{(t)}) - \mathsf{HW}(T_R{}^{(t-1)})$ for $d_f + 2 \leq t \leq 2 \times d_f + 1$. In this case

Table I
THE HAMMING WEIGHT OF T_L AND T_R AS OBTAINED FROM JTAG SCAN CHAIN OUTPUT IN EXAMPLE 1

t	$HW(T_L^{(t)})$	$HW(T_R^{(t)})$
1	0	28
2	8	23
3=d_f+1	10	23
4	12	16
5=2×d_f+1	12	16

and according to the obtained Hamming weights, different possibilities for these locations, at each t, can be recovered. The attacker appends all these possible locations of the -1's in a set of lists, S_2. In particular, for this example, the attacker evaluates $S_2 = \{k[3] = [1], k[4] = [2]\}$. Then, the values in S_1 represents the distances between the locations of +1's in the key in the form ($k[t]-k[0]$, $1 \leq t < d_f+1$) and the values in each element in S_2 represents a list of possible locations of the -1's in the key. Enumerating all possible value for $k[0]$, $0 \leq k[0] < N - S_1[d_f]$ (in this example, $0 \leq k[0] < 4$), the attacker is able to uniquely determine the correct key locations $\{k[0] = 3, k[1] = 5, k[2] = 6, k[3] = 1, k[4] = 2\}$. Figures 5 shows the corresponding input, output and intermediate computational results of Algorithm 2. Detailed calculations corresponding to Algorithm 3 are omitted due to the space limitations.

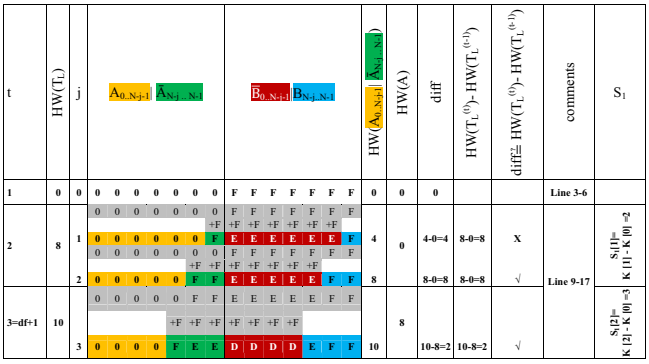

Figure 5. The computation steps corresponding to running Algorithm 2 with the parameters in Example 1

IV. EXPERIMENTAL RESULTS

In order to verify the correctness of the proposed attack, we implemented the NTRUEncrypt decryption system with the convolution circuit depicted in Figure 4 using the Synopsys Design Compiler and inserted a scan chain using Synopsys Test Compiler. Using this implementation, we confirmed our ability to determine the scan chain structure and the Hamming weight of T_R and T_L. The Hamming weights obtained from the ModelSim simulation were then

used as input to Algorithm 2 and Algorithm 3 which were implemented using Python programming language. Table II shows our simulation results for Algorithm 2 and Algorithm 3 with 100 randomly selected keys for NTRUEncrypt with parameters $(N, p, q, d_f, d_g, d_r) = (167, 3, 128, 61, 20, 18)$, $(263, 3, 128, 50, 24, 16)$, and $(503, 3, 256, 216, 72, 55)$ which correspond to the moderate, high, and highest security parameters in [24]. As shown in the table, the average size of the list of keys, returned by Algorithms 2 and 3, is given by $\approx 2^{18}, 2^{24}$ and 2^{64} for these three set of parameters while the exhaustive search key security is given by $\frac{1}{d_g!}\sqrt{\frac{N!}{(N-2d_g)!}} \approx 2^{83}$, 2^{111} and 2^{285}, respectively [24]. It should be noted that this relatively large values for the average were dominated by some few cases where the size of the resulting key list were too large compared to the other cases. As mentioned above, the unique correct key can be determined by going through this list and verifying the correctness of the decryption operation for a known plaintext. This off-line step does not require physical access to the cryptographic device. It also does not require large memory space since the size of the sets S_1 and S_2 (outputs from Algorithm 2 and 3) is limited to $O(d_f N)$. Also, since there is no dependency between the different search paths, then this exhaustive search step can be easily parallelized. It should be noted that the overall complexity of the attack is dominated by the complexity of Algorithm 3 and the complexity of going through the list of keys calculated by Algorithms 2 and 3 since the number of steps required by the scan-in and scan-out operations is negligible compared to these two steps.

Table II
AVERAGE SIZE OF THE LIST OF SUGGESTED KEYS

	$N = 167$	$N = 263$	$N = 503$
Average $\approx$	$2^{17.98}$	$2^{23.28}$	$2^{63.52}$

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a scan-based side channel attack to recover the NTRUEncrypt secret key. By analyzing the Hamming weight of the scan chain output at carefully chosen clock cycles, the attacker is able to efficiently recover the locations of the +1's and -1's of the secret key polynomial. The presented attack clearly shows the need to utilize secure scan chains [25] for hardware implementations of NTRUEncrypt with the scan-based DFT feature.

Other instantiations of NTRUEncrypt were proposed (e.g., see [26], [27], and [28]). While our attack can be applied in a straightforward way to [28], it is interesting to investigate how it can be applied to [26] and [27] where $p = x + 2$, q is a prime number and f can be expressed in the form $1 + p \star F$ where F has d_F coefficients equal to 1.

REFERENCES

[1] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing,* Kluwer Academic Publishers, Boston, 2000.

[2] *IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture*, IEEE Std 1149.7-2009, February 2010.

[3] B. Yang, K. Wu, and R. Karri, *Scan based side channel attack on dedicated hardware implementations of data encryption standard,* Proc. of International Test Conference, ITC'04, pp. 339-344, 2004.

[4] B. Yang, K. Wu, and R. Karri, *Secure scan: a design-for-test architecture for crypto chips,* Proc. of the 42^{nd} Annual Conference on Design Automation, DAC'05, pp. 135-140, 2005.

[5] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, A scan-based attack based on discriminators for AES cryptosystems, *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A, no. 12, pp. 3229-3237, 2009.

[6] R. Nara, N. Togawa, M. Yanagisawa, and T. Ohtsuki, Scan-based side-channel attack against RSA cryptosystems using scan signatures, *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E93-A, no. 12, pp. 2481-2489, 2010.

[7] R. Nara, N. Togawa, M. Yanagisawa, T. Ohtsuki, *Scan-Based Attack against Elliptic Curve Cryptosystems,* Proc. of Design Automation Conference (ASP-DAC'10), pp. 407-412, 2010.

[8] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay, *Scan Based Side Channel Attacks on Stream Ciphers and Their Counter-Measures,* Proc. of INDOCRYPT'08, LNCS 5365, pp. 226-238, Springer-Verlag, 2008.

[9] Y. Liu and K. Wu, Scan-Based Attacks on Linear Feedback Shift Register Based Stream Ciphers, *ACM Transactions on Design Automation of Electronic Systems*, vol. 16, issue 2, pp. 1-15, 2011.

[10] J. Hoffstein, J. Pipher and J. H. Silverman, *An Introduction to Mathematical Cryptography*, Undergraduate Texts in Mathematics, Springer, 2008.

[11] J. Hoffstein, J. Pipher and J. Silverman, *NTRU: a ring based public key cryptosystem,* Proc. of ANTS III, LNCS 1423, pp. 267-288, Springer-Verlag, June, 1998.

[12] B. Kaliski, Considerations for New Public-key Algorithms, *Network Security* vol. 2000, issue 9, pp. 9-10, September 2000.

[13] E. Jaulmes and A. Joux, *A chosen-ciphertext attack against NTRU*, Proc. of CRYPTO'00, LNCS 1880, pp. 20-35, Springer-Verlag, 2000.

[14] P. Nguyen and D. Pointcheval, *Analysis and Improvements of NTRU Encryption Paddings*, Proc. of CRYPTO'02, LNCS 2442, pp. 210-225, Springer-Verlag 2002.

[15] J. Silverman and W. Whyte, *Timing Attacks on NTRUEncrypt Via Variation in the Number of Hash Calls,* Proc. of CT-RSA'07, LNCS 4377, pp. 208-224, Springer-Verlag, 2007.

[16] A. Atici, L. Batina, and I. Verbauwhede, *Power Analysis on NTRU implementation for RFIDs: First results,* Proc. of RFIDSec'08, pp. 128-139, 2008.

[17] M. Lee, J. Song, D. Choi , and D. Han, Countermeasures against Power Analysis Attacks for the NTRU Public Key Cryptosystem, *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E93-A, issue 1, pp. 153-163, 2010.

[18] A. Kamal, A. Youssef, Fault Analysis of the NTRUEncrypt Cryptosystem, *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E94-A, issue 4, pp. 1156-1158, 2011.

[19] Y. Shi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, Robust Secure Scan Design Against Scan-Based Differential Cryptanalysis, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 1, pp. 176-181, January 2012.

[20] A. C. Atici, L. Batina, J. Fan, I. Verbauwhede, and S. B. Ors, *Low-cost Implementations of NTRU for pervasive security,* Proc. of 19^{th} IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), pp. 79-84, 2008.

[21] J. Kaps. *Cryptography for Ultra-Low Power Devices*, PhD thesis, Worcester Polytechnic Institute, USA, May 2006.

[22] M. Monteverde, *NTRU software implementation for constrained devices*, Msc. thesis, Faculty of Engineering, Katholike Universiteit Leuven, Belgium, May 2006.

[23] D. Bailey, D. Coffin, A. Elbirt, J. Silverman and A. Woodbury, *NTRU in Constrained Devices*, Proc. of CHES'01, LNCS 2162, Springer-Verlag, pp. 262-272, 2001.

[24] J. Hoffstein, D. Lieman, J. Pipher, J. H. Silverman, *NTRU: A Public Key Cryptosystem*, Submissions and Contributions to IEEE P1363.1, Presented at the August 1999 and October 1999 meetings. Avilable at http://grouper.ieee.org/groups/1363/lattPK/submissions/ntru.pdf

[25] D. Hely, F. Bancel, M. Flottes, *Secure scan techniques: a comparison*, Proc. of the 12th IEEE International On-Line Testing Symposium (IOLTS'06), 2006.

[26] Consortium for Efficient Embedded Security, *Efficient embedded security standards #1: Implementation aspects of NTRU and NSS*, 2001.

[27] Consortium for Efficient Embedded Security, *Efficient embedded security standards #1: Implementation aspects of NTRUEncrypt and NTRUSign*, 2002.

[28] N. Howgrave-Graham, J. Silverman, and W. Whyte, *Choosing parameter sets for NTRUEncrypt with NAEP and SVES-3,* Proc. of CT-RSA'05, LNCS 3376, pp. 118-135, Springer-Verlag, 2007.

Algorithm 3 Recovery of the -1's locations in the private key polynomial f

INPUT: A, B and $S_1[d_f]$ (from Algorithm 2), $\mathsf{HW}({T_L}^{(t)})$ and $\mathsf{HW}({T_R}^{(t)})$, $d_f + 2 \leq t \leq 2 \times d_f + 1$
OUTPUT: An array S_2 of lists where $S_2[t - d_f]$ is a list containing all estimated possible values for $k[t]$, $d_f + 1 \leq t < 2 \times d_f + 1$.
for $0 \leq k[0] < (N - S_1[d_f])$ **do**
  for $d_f + 2 \leq t \leq 2 \times d_f + 1$ **do**
    $i = 1$
    repeat
      if $(t = d_f + 2)$ **then**
        $j \longleftarrow 0$
      else
        $j \longleftarrow S_2[t - d_f - 2][i] + 1$
        $A|B \longleftarrow L[t - d_f - 2][i]$
      end if
      while $(j < N)$ **do**
        if $(j \neq k[0])$ **then**
          if $(j < k[0])$ **then**
            $\mathsf{Temp}_A \longleftarrow \overset{+}{A}_{0..k[0]-j-1}|A_{k[0]-j..N-1}$
            $\mathsf{Temp}_B \longleftarrow B_{0..k[0]-j-1}|\overset{+}{B}_{k[0]-j..N-1}$
            $\mathsf{diff}_1 \longleftarrow \mathsf{HW}(\mathsf{Temp}_A) - \mathsf{HW}(A)$
            $\mathsf{diff}_2 \longleftarrow \mathsf{HW}(\mathsf{Temp}_B) - \mathsf{HW}(B)$
          else
            $\mathsf{Temp}_A \longleftarrow A_{0..N-j+k[0]-1}|\overset{+}{A}_{N-j+k[0]..N-1}$
            $\mathsf{Temp}_B \longleftarrow \overset{+}{B}_{0..N-j+k[0]-1}|B_{N-j+k[0]..N-1}$
            $\mathsf{diff}_1 \longleftarrow \mathsf{HW}(\mathsf{Temp}_A) - \mathsf{HW}(A)$
            $\mathsf{diff}_2 \longleftarrow \mathsf{HW}(\mathsf{Temp}_B) - \mathsf{HW}(B)$
          end if
          if $(\mathsf{diff}_1 = \mathsf{HW}({T_L}^{(t)}) - \mathsf{HW}({T_L}^{(t-1)})$ and $\mathsf{diff}_2 = \mathsf{HW}({T_R}^{(t)}) - \mathsf{HW}({T_R}^{(t-1)}))$ **then**
            $A \longleftarrow \mathsf{Temp}_A$
            $B \longleftarrow \mathsf{Temp}_B$
            append $A|B$ to $L[t - d_f - 1]$
            append the location j to $S_2[t - d_f - 1]$
          end if
          $j \longleftarrow j + 1$
        else
          $j \longleftarrow j + 1$
        end if
      end while
      $i = i + 1$
    until $i >$ number of elements in the list $S_2[t - d_f - 1]$
  end for
end for
return S_2