# Secure and Compact Full NTRU Hardware Implementation

Konstantin Braun*, Tim Fritzmann*, Georg Maringer*, Thomas Schamberger*, Johanna Sepúlveda*
*Technical University of Munich, Munich, Germany
email: {konstantin.braun, tim.fritzmann, georg.maringer, t.schamberger, johanna.sepulveda}@tum.de

***Abstract*—The foreseeable breakthrough of quantum computers represents a risk for secure communications. In order to prepare for such an event, electronic systems must integrate secure quantum-computer-resistant (post-quantum) cryptography protected against implementation attacks. The NTRU cryptosystem is one of the main alternatives for practical implementations of post-quantum public-key cryptography. The standardized version of NTRU (IEEE 1363.1) provides security against chosen ciphertext attacks (CCA) through a padding scheme that limits ciphertext malleability, thus restricting a large range of attacks. So far, previous NTRU hardware implementations do not include the NTRU padding scheme. Moreover, a previously proposed NTRU optimization of the polynomial multiplication leads to a degradation of the security level. Therefore, previous works provide a wrong impression regarding the real implementation cost of NTRU. In this work, we present two contributions: i) the first complete and compact NTRU hardware implementation; and ii) the analysis of the security degradation due to the NTRU multiplication optimization proposed in previous works.**

## I. Introduction

Public-key cryptography (PKC) provides the basis for establishing secured communication channels between multiple parties. It allows confidentiality, authenticity and non-repudiation of electronic communications and data storage. Internet-of-Things (IoT) and Cloud computing are some of the technologies that use PKC to secure channels. The fast development of quantum computers represents a risk for many public-key cryptosystems. Almost all established approaches rely on the hardness of factoring large integers (RSA) or computing discrete logarithms (ECC). It is known that cryptographic systems based on these problems will be threatened when a large enough quantum computer is built. Shor's quantum algorithm will solve the problems on which PKC currently relies in polynomial time.

To ensure long term communication security quantum-resistant (also called post-quantum) cryptography must be adopted. Post-quantum cryptography relies on mathematical problems that remain hard to solve even with a quantum computer. In order to provide resistance against quantum attacks, the National Institute of Standards and Technology (NIST) announced the beginning of the transition towards post-quantum cryptography [1]. The NTRU lattice-based cryptosystem is one of the main alternatives for practical implementations of post-quantum PKC. NTRU is characterized by small key sizes (low memory footprint) and computational efficiency when compared to other post-quantum approaches [2], [3]. NTRU has been standardized in the IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices (IEEE-1363.1) [4].

Empowering electronic devices with strong security poses a challenging problem due to limited resources and tight performance requirements. Moreover, embedded implementations must be resistant to implementation attacks, such as Chosen-Ciphertext Attack (CCA) or Side-Channel Attacks (SCA). Adversaries can recover the secret key by gathering information obtained through the decryption of fabricated ciphertexts or by the physical information leakage during the cryptographic operation (power consumption, timing and electromagnetic radiation). While CCA can be avoided by adopting a padding scheme, inhibiting SCA requires a careful implementation of the cryptographic algorithm. The NTRU standard (IEEE-1363.1) defines the Short Vector Encryption Scheme (SVES) as the padding scheme to avoid CCA.

NTRU hardware implementations have been previously demonstrated in [5]–[10]. While current works about NTRU focus on efficient convolution techniques, a complete implementation of the standardized NTRU is still missing. Moreover, security aspects of the implementation are still largely unexplored. The works presented in [5]–[10] do not implement the SVES padding scheme. Furthermore, the NTRU optimization presented in [10] reduces the security of NTRU by leaking information regarding the secret key, as the execution time of this implementation depends on the value of the secret key. This makes an implementation impractical for real applications.

In this work, we present the first complete, compact and secure NTRU hardware implementation. In addition, we demonstrate the security reduction of a previous NTRU implementation. In summary, the contributions of the paper are:

- First complete NTRU hardware implementation which includes the SVES padding scheme;
- A compact NTRU implementation able to execute encryption and decryption operations;
- Performance and cost evaluation of our NTRU implementation;
- A security analysis of the previous NTRU implementation presented in [10] and demonstration of the security reduction.

The remainder of this article is organized as follows: Section II presents the previous work on NTRU hardware implementations. Section III describes the instantiation of NTRU with the SVES padding scheme. Section IV and Section V present our complete NTRU implementation and the security analysis of the optimized NTRU presented in [10]. The experimental results are presented in section VI. A conclusion is given in section VII.

## II. Related Works

The probably first NTRU encryption hardware implementation was proposed by Bailey *et al.* in 2001 [5]. To speed up the polynomial multiplication, which is usually the performance bottleneck of NTRU, the authors propose to scan the

978-1-5386-4756-1/18/$31.00 

coefficients of the blinding polynomial $r$. For each non-zero coefficient, the public key $h$ is added to a temporary result. Atici proposed the first encryption and decryption NTRU hardware implementation. The architecture includes power saving methods, such as clock gating and partially rotating registers [6]. The implementation of Kamal *et al.* uses the special structure of the public key, which has a large number of zero coefficients to optimize performance [7]. In [9], Liu *et al.* use the fact that the polynomial multiplication in the truncated polynomial ring of NTRU can be modeled with a linear feedback shift register (LFSR) to implement the polynomial multiplication. In [10], they speed up their implementation by skipping the multiplication operation when two consecutive zero coefficients in the ternary polynomial are detected. Thus, the multiplication time depends on the number of double zeros contained in the polynomial. This information decreases the NTRU security level as discussed in Section V.

Moreover, so far none of the existing works proposed a full hardware implementation of NTRU with the SVES padding scheme as defined in the IEEE-1363.1 standard. As the integration of SVES is mandatory to inhibit a CCA, these implementations show a misleading picture of the implementation cost. A commonly used tool for transforming cryptographic algorithms into CCA secured schemes is the NAEP [11] transformation. SVES is a concrete instantiation of the NAEP transform, which was specially designed for NTRU. The first iterations, SVES-1 and SVES-2, are vulnerable to attacks exploiting decryption errors [12]. The latest iteration, SVES-3, which is sometimes only referred to as SVES, does not show this vulnerability. It is standardized in IEEE-1363.1 [4]. In contrast to previous works, we present a CCA secure NTRU hardware implementation compliant with the standard.

## III. NTRU

### A. Notation

The main elements of NTRU are the polynomials in the following integer rings:

$$R_{N,p} = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^N - 1)}, \quad R_{N,q} = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^N - 1)} \ . \tag{1}$$

These rings define each polynomial to be at most of degree $N-1$ and to have integer coefficients. For $R_{N,p}$ and $R_{N,q}$ these coefficients are reduced modulo $p$ and modulo $q$, respectively. Unless otherwise noted, all polynomials are elements of the ring $R_{N,q}$.

For the standardized parameter sets of NTRU the modulus $p$ is fixed to a small prime $p = 3$. In this case, the elements of the ring $R_{N,p}$ are called ternary polynomials. A ternary polynomial $\mathcal{T}_N(d, e)$ has $d$ coefficients equal to one and $e$ coefficients equal to minus one, while the remaining coefficients are set to zero. NTRU ternary polynomials are sparse, that is, the majority of coefficients are set to zero. The values $d$ and $e$ are part of the parameter set and can be changed to achieve different security levels.

Additionally, the parameter $q$ is fixed to $q = 2048$, which simplifies the implementation of the algorithm. By choosing the modulus to be a power of two, the modulo operation can be performed without additional cost by reducing the result register size.

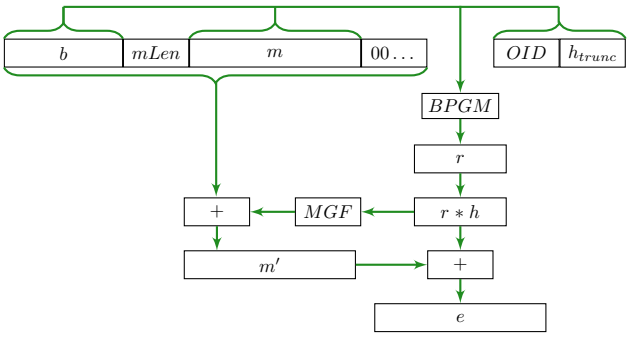


Fig. 1. NTRU encryption with SVES

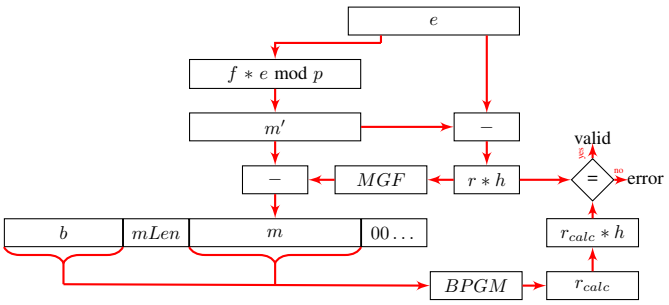


Fig. 2. NTRU decryption with SVES

### B. Short Vector Encryption Scheme (SVES)

Padding schemes prevent cryptanalysis by hiding the characteristics of the ciphertext. For instance, the length of the encrypted message may leak information regarding the original message $m$. The SVES pads the message by using a variable number of zeros and uses two auxiliary methods: i) Blinding Polynomial Generation Method (BPGM) and ii) Mask Generation Function (MGF).

*1) Blinding Polynomial Generation Method (BPGM):* This method generates a ephemeral blinding polynomial $r$ in a deterministic way with the use of a pseudo-random number generator (PRNG). This PRNG is based on a hash function $\mathbf{G}$ and is initialized by a seed consisting of four values:

$$BPGM(OID, b, m, h_{trunc}) \ . \tag{2}$$

The identifier $OID$ is an unique three-byte value for each parameter set. The parameter $b$ is a random number and $m$ the message to be encrypted. The last part of the seed $h_{trunc}$ consists of a defined number of bits of the public key $h$.

*2) Mask Generation Function (MGF):* Similar to BPGM, the MGF uses a hash function $\mathbf{G}$ to generate a mask. The input of $\mathbf{G}$ is the result of the polynomial multiplication of the ephemeral blinding polynomial and the public key $(r * h)$. The resulting mask is added to $m$.

### C. NTRU with SVES

The NTRU scheme instantiated with SVES consists of the three cryptographic operations: key generation, encryption and decryption.

The key generation step creates a key pair, consisting of the public key $h$ with its corresponding secret key $f$, through three steps. The first step generates two random ternary polynomials,

$F \in R_{N,p}$ and $g \in R_{N,p}$. The positions of the polynomial coefficients with value one and minus one are selected based on an uniform distribution. The second step calculates the private key $f$ as $f = 1 + pF$ together with its inverse $f^{-1}$ modulo $q$. Not all the polynomials have an inverse in the corresponding ring. Therefore, it is possible that the inverse $f^{-1}$ can not be found. In this case, the key generation is restarted until a key with a valid inverse is found. The third step computes the public key $h$ as $h = f^{-1} * g$.

The NTRU encryption is shown in Fig. 1. It transforms a message $m$ into a ciphertext $e$ through five steps. The first step formats $m$ into a ternary polynomial representation and concatenates this polynomial together with the random $b$, the identifier $OID$, $h_{trunc}$, $mLen$ and zeros for padding as $m_{pad} = (OID \,||\, b \,||\, m \,||\, h_{trunc} || \, mLen || \, 00 \dots)$. The second step uses $m_{pad}$ as the input of the BPGM to create the ephemeral polynomial $r$ as $r = BPGM(m_{pad})$. The third step multiplies $r$ with the public key polynomial $h$. The result is masked through the MGF in order to obtain $m_{Mask} = MFG(r * h)$. The fourth step adds the mask to the concatenated message to produce $m' = m + m_{Mask}$. The final step computes the ciphertext as $e = m' + r * h$.

The NTRU decryption is shown in Fig. 2. It retrieves the original message $m$ from the ciphertext $e$ through four steps. The first step retrieves $m'$ by multiplying the ciphertext $e$ with the private key $f$ as $m' = f*e \pmod p$. In the second step $r*h$ can be retrieved, by subtracting $m'$ from the ciphertext $e$, as the equation $r*h = e - m'$ holds. The third step uses the resulting product as an input to the MGF to retrieve the concatenated message as $m = m' - MGF(r * h)$. The fourth step checks the validity of the ciphertext by applying the BPGM to the corresponding elements of $m$ to produce the value $r_{calc}$. The multiplication of $r_{calc}*h$ is now compared with the polynomial $r * h$ from the second decryption step. If both polynomials are equal, the algorithm outputs the concatenated message $m$. Otherwise an invalid ciphertext is detected and the algorithm outputs an error message.

## IV. NTRU Architecture

Our proposed hardware NTRU architecture is illustrated in Fig. 3. The encryption and decryption flows are highlighted in green and red, respectively. To keep the area costs low, the encryption and decryption operation share common hardware modules. The resource sharing is managed by a small controller that sets the data selector values of all multiplexers. The NTRU architecture is composed of four main hardware modules: CONV, BPGM, MGF and MOD $p$. Their implementation is described in the following subsections.

### A. Convolution (CONV)

In this work, we adopted the convolution architecture of [9] because of its efficiency and simplicity. This architecture is able to multiply a ternary polynomial with a regular polynomial in $R_{N,q}$. However, in order to support the encryption and decryption operations, the following modifications are required: i) integration of the *control unit* to manage the convolution during encryption and decryption operations; and ii) support for the regular multiplication ($f*e$). The enhanced convolution circuit (*CONV*) is shown in Fig. 4.

*CONV* multiplies a ternary polynomial $A \in \mathcal{T}_N$ with coefficients $\{-1, 0, 1\}$ and a regular polynomial $B \in R_{N,q}$.

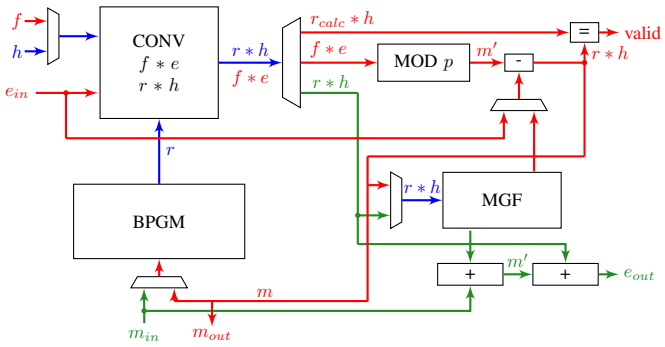


Fig. 3. NTRU architecture, green encryption, red decryption, blue shared

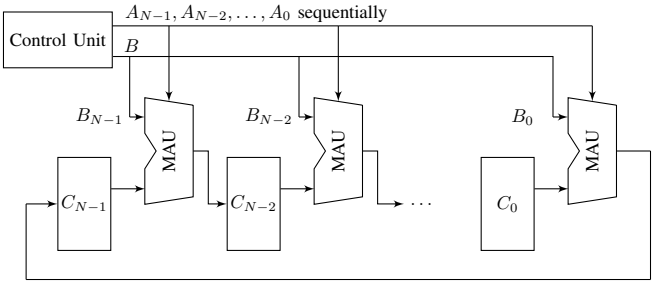


Fig. 4. Circular convolution model

The circularity of the convolution is realized by shifting the result values $C$ in an linear-feedback shift register (LSFR). Depending on the sequentially inputted coefficient of $A$, the modular arithmetic unit (*MAU*) either adds $B_i$ to $C_i$, subtracts $B_i$ from $C_i$ or keeps $C_i$ unchanged, where $i = 0, 1, \dots N-1$. The result of each *MAU* is forwarded to the next register.

During encryption, the ternary polynomial $r$ and a regular polynomial $h$ are multiplied through *CONV*, thus generating $r*h$. However, the decryption requires a multiplication of two regular polynomials $f$ and $e$. In order to use *CONV* for this multiplication, we use the definition of $f$ given in the IEEE-1363.1 standard, such that $f = 1 + pF$, where $F$ is a ternary polynomial. As a result, $f * e = (1 + pF) * e = e + pF * e$. To obtain $pF * e$, we repeat the convolution of $F * e$ two more times ($p = 3$) without resetting the registers after each round. At the end of this operation, the *control unit* inputs a ternary polynomial such that the value of the first coefficient is '1' and '0' otherwise. The second input will remain with the polynomial $e$. This procedure for calculating the addition of $pF*e$ with $e$ takes one round. To avoid this round, the registers can also be preloaded with $e$ at the beginning of the decryption process. In addition to the calculation of $f * e$, the decryption has to calculate $r_{calc} * h$, which requires one additional round. The proposed process increases the convolution processing time during the decryption operation by a factor of four. However, it avoids the integration of additional multipliers, thus decreasing the required area for the decryption.

### B. Blinding Polynomial Generation Method (BPGM)

Hash functions are the core of the *BPGM* and *MGF* modules. The IEEE-1363.1 standard suggests the use of SHA-1 or SHA-256, depending of the desired security level. SHA-1 and SHA-256 have a 512 bit input and 160 and 256 bit output,

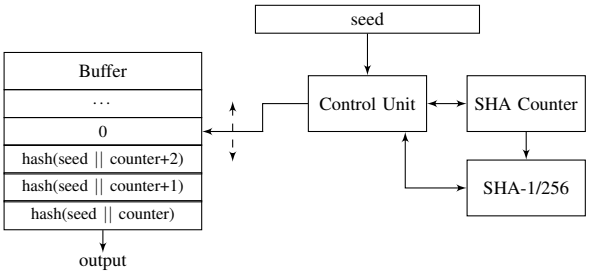


Fig. 5. Buffer generation

respectively. The NTRU parameter set defines the number of calls of the SHA function in order to generate the ephemeral blinding polynomial $r$. More specifically, the SHA function is executed $minCallsR$ times, whose value is defined in the standard. The seed varies in each hash call, by using the four values described in Subsection III-B ($OID, b, m, h_{trunc}$) concatenated with a *counter* value, which is increased after each hash call. Our NTRU architecture uses the *SHA-1/256* open core modules from [13]. The *control unit* manages the generation of $r$. As shown in Fig. 5, the generated hash values are stored in a buffer. These values are used for determining the indexes of ones and minus ones of $r$.

### C. Mask Generation Function (MGF)

The $MGF$ shares the *buffer generation module* presented in Fig. 5 with the $BPGM$. However, instead of using the buffer output for finding the value of the indexes of $r$, the $MGF$ transforms the output from a binary into a ternary representation. A look-up table is employed to perform this transformation.

### D. Modulo reduction (MOD p)

The coefficients of the polynomials can be reduced modulo $p$ by subtracting $p$ from each coefficient of the polynomial until the coefficient is smaller than $p$. However, as $p = 3$ is a Mersenne prime number, a faster method to calculate the modulo reduction can be employed, as shown in [14] and optimized in [3]. Algorithm 1 presents the modulo reduction employed in our NTRU architecture. To improve the NTRU performance, the *MOD* $p$ block can be instantiated multiple times.

**Algorithm 1:** Mersenne prime modulo division ($p = 3$)

**Input:** Integer $a$
**Result:** Integer $a$ mod 3
additional_reduction = $\{0, 1, 2, 0, 1, 2\}$
// reduce $a$
$a = (a \gg 8) + (a \ \& \ 0xFF)$
$a = (a \gg 4) + (a \ \& \ 0xF)$
$a = (a \gg 2) + (a \ \& \ 0x3)$
$a = (a \gg 2) + (a \ \& \ 0x3)$
// at this point $a < 6$
a = additional_reduction[$a$]

## V. Security Analysis

In [10], Liu *et al.* show an optimization of their implementation presented in [9]. It is based on scanning the coefficients of the ternary polynomial. When two consecutive zeros are detected, the multiplication can be skipped. In the following subsections, we describe the optimized architecture and present the security vulnerability caused by this optimization.

### A. Optimized Architecture

The optimized architecture is able to detect two consecutive zeros in the ternary input polynomial $A$ (Figure 4). The processing of a zero coefficient during the convolution can be seen as a single circular shift of the coefficients $C_i$. Therefore, two zeros can be substituted by a single shift of two places within one clock cycle. This results in a reduction of one cycle in the total multiplication time for each pair of consecutive zeros. The implementation of Liu *et al.* requires an additional multiplexer for the *MAU*, which is connected to the preceding register output of the result coefficient $C_i$. In comparison to their original and non-optimized implementation in [9], the authors report a reduction of 36.7 % of the execution time for the convolution with the parameter set *ees541ep1*.

### B. Vulnerability

The optimized implementation of [10] leaks information regarding the secret key through a timing side-channel because the convolution time depends on the structure of the secret key. More specifically, it depends on the amount of consecutive double zeros in the secret key polynomial $F$. This vulnerability cannot be fixed because the optimization is solely based on the amount of double-zeros. A time protected version will remove the performance gains and will behave equally to the unoptimized implementation. Exploiting the side-channel by observing the execution time of the algorithm reveals the amount of double-zeros in the private key polynomial $F$. This additional information limits the search space and therefore reduces the effective length of the private key polynomial and thus the security level.

**Theorem 1.** *The number of valid ternary polynomials for a given number of double-zeros $d_z$ is upper bounded by*

$$u_r(n, d_f, d_z) = \frac{(2d_f + d_z)!}{(d_f!)^2 d_z!} \binom{2d_f + d_z + 1}{n - 2d_f - 2d_z}, \quad (3)$$

*where $n$ is the number of coefficients of the polynomial and $d_f$ the number of ones and minus ones in the private key $F$.*

*Proof.* Valid keys consist of zeros, double zeros (two consecutive zeros), ones and minus ones. All possible patterns are constructed using these elements following the constraints imposed by the public parameters ($n$, $d_f$) as well as the number of double zeros $d_z$, gained from the timing side-channel. For simplicity, we discuss the two factors of Equation 3 independently.

The first factor represents the quantity of possible key patterns under the assumption that the locations of single zeros are fixed. It counts the number of all pattern changes that can be made by swapping double zeros, ones and minus ones. As a visualization, we denote the interchangeable elements with the symbol $*$ and the fixed single zero coefficients with 0.

$$*0**0**\cdots*0**$$

The number of $*$ symbols in the pattern is $2d_f + d_z$, which gives

$$\frac{(2d_f + d_z)!}{(d_f!)^2 d_z!} \tag{4}$$

possible combinations (using the multiset permutation formula).

In the next step, we take into account that the locations of single zeros are not fixed anymore and introduce the second part of Equation 3. This factor increases the amount of possible patterns as it takes the different possibilities of single zero locations into account. It is upper bounded to

$$\binom{n - d_z - l + 1}{l} \ , \tag{5}$$

where $l$ denotes the number of single zeros. The parameter $l$ is set to $n - 2d_f - 2d_z$, which can be included in Equation 5 to get the second factor of Equation 3.

The proof for Equation 5 is done by induction. First, we consider only two single zeros ($l = 2$) in the pattern and fix the rightmost zero at the last position.

$$**********\cdots * 0 * 0$$

In this case, the first zero has $(n - d_z - 2)$ possible positions in the pattern. Shifting the rightmost zero by one position to the left reduces the number of possibilities for the first zero by one, resulting in $(n - d_z - 3)$ possibilities. The summation of these quantities for all valid positions of the rightmost zero results in

$$\frac{(n - d_z - 1)(n - d_z - 2)}{2} = \binom{n - d_z - 1}{2} \ . \tag{6}$$

Next, the inductive step ($l \rightarrow l+1$) is shown. In the following, the last two elements, containing one of the single zeros, are separated from the other elements by the symbol $\|$.

$$*********\ldots 0 * 0 * \cdots * 0 * 0 \,\|\, * 0$$

All single zeros $l$ are located left of this border. Therefore, we can use the induction hypothesis to compute the amount of possible configurations for the single zeros given by

$$\binom{(n - d_z - 2) - l + 1}{l} \ . \tag{7}$$

Similar to the proof for $l = 2$, the rightmost zero is shifted step by step to the left. By adding up the resulting possibilities, it follows

$$\sum_{k=0}^{n-d_z-2l-1} \binom{l + k}{l} = \binom{n - d_z - l}{l + 1} \ , \tag{8}$$

by using the equation

$$\sum_{k=0}^{m} \binom{n + k}{n} = \binom{n + m + 1}{n + 1} \ . \tag{9}$$

□

Figure 6 illustrates the complexity reduction factor $\alpha$ for different parameter sets. This factor can be computed by

$$\alpha = \frac{u_r(n, d_f, d_z)}{K_c} \ , \text{ with } \ K_c = \frac{n!}{(d_f!)^2 (n - 2d_f)!} \ , \tag{10}$$

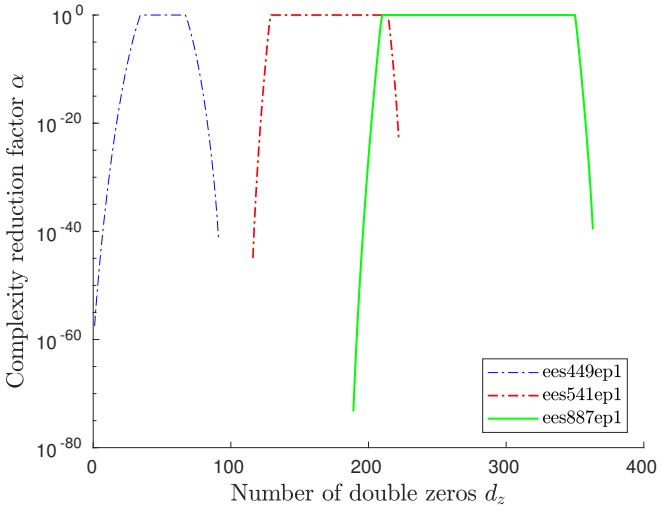


Fig. 6. Complexity reduction of the exhaustive search space of the private key $F$ for a known amount of double-zeros. For the parameter set *ess887ep1*, the secret key $F$ consists of $d_f = 81$ ones and minus ones, which results in a maximum of 362 double zeros. The complexity reduction factor is bounded by a minimal amount of double zeros as it is possible that the number of non-zero elements is not sufficient to separate the remaining zeros.

where $K_c$ denotes the cardinality of the key space. It shows the complexity reduction of an exhaustive search for $F$ given a known amount of double zeros.

## VI. Results

Our proposed NTRU hardware architecture was implemented on the Zedboard, which is equipped with a Xilinx Zynq-7000. The IEEE-1363.1 standard provides different parameter sets for different security levels and optimization goals. Table I summarizes the results of our proposed system. It contains the total number of LUTs and the required number of clock cycles for encryption and decryption. The results show that the number of LUTs scales with the parameter $n$, which determines the size of the polynomials.

Figure 7 provides a more detailed view of the required clock cycles for the encryption. The time required for the convolution depends directly on the value of $n$, as $n$ clock cycles are required for the circular shift within the LSFR. Results show that the impact of the padding scheme on the cost and performance of NTRU is not negligible. For some NTRU configurations, the convolution has a minor influence on the computation cost when compared to the padding scheme. For the parameter set $ees401ep1$, the padding scheme takes nearly 90 % of the encryption time.

The main bottleneck of the padding scheme is the hash function. The number of clock cycles spent by the BPGM depends on the parameter $d_r$, which determines the number of ones and minus ones of $r$ and thus the number of hash calls.

Figure 8 illustrates the computation costs for the decryption. Compared to the encryption, the number of clock cycles of the convolution during the decryption is four times higher. Both, the convolution and the modulo $p$ operation directly scale with $n$. For the modulo $p$ operation only one module was implemented. To decrease the computation cost, several instances of MOD $p$ modules can be used.

TABLE I
PARAMETER SET DEFINED IN IEEE-1363.1

| Security Level | Parameter Set | $n$ | LUTs | #CC Enc. | #CC Dec. |
|---|---|---|---|---|---|
| Low | $ees401ep1$ | 401 | 29545 | 3423 | 5430 |
| | $ees541ep1$ | 541 | 38240 | 2409 | 5116 |
| | $ees659ep1$ | 659 | 46124 | 2413 | 5711 |
| Middle | $ees449ep1$ | 449 | 32907 | 3642 | 5890 |
| | $ees613ep1$ | 613 | 44797 | 2675 | 5743 |
| | $ees761ep1$ | 761 | 53338 | 2799 | 6606 |
| High | $ees677ep1$ | 677 | 49786 | 4020 | 7407 |
| | $ees887ep1$ | 887 | 63861 | 3113 | 7551 |
| | $ees1087ep1$ | 1087 | 74280 | 3760 | 9197 |
| Highest | $ees1087ep2$ | 1087 | 74620 | 4723 | 10159 |
| | $ees1171ep1$ | 1171 | 83896 | 4345 | 10202 |
| | $ees1499ep1$ | 1499 | 99717 | 4715 | 12212 |

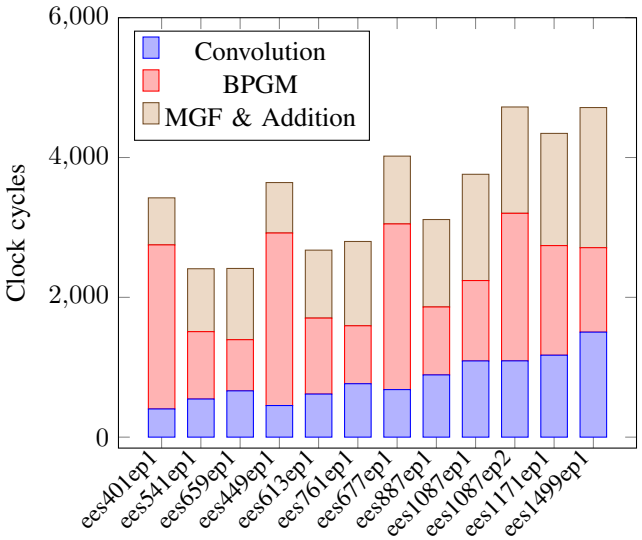


Fig. 7. Clock cycles for encryption

## VII. CONCLUSION

Efficient and secure post-quantum cryptography is mandatory to ensure long-term security. In this work, we proposed the first compact, complete and secure NTRU architecture. It is compact due to its shared resources for the encryption and decryption process. In contrast to previous works, our NTRU architecture is complete because we included the SVES padding scheme, defined in IEEE-1363.1. Our architecture is secure because it avoids CCA attacks by including the padding scheme, and SCA attacks by implementing a constant time convolution. We show that previous NTRU works may decrease the security level of NTRU. Moreover, we show that the integration of the SVES scheme requires, for some parameters, nearly 90 % of the encryption time and thus its implementation cost cannot be neglected.

**Acknowledgments.** This work was partly funded by the Fraunhofer High Performance Center for Secure Connected Systems of Munich.

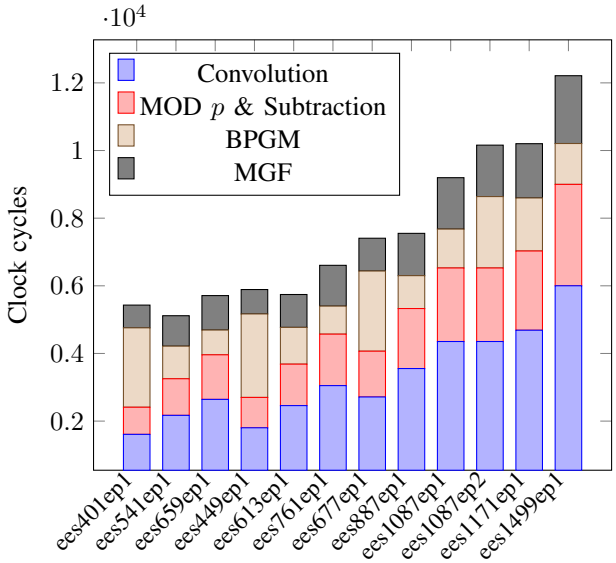


Fig. 8. Clock cycles for decryption

## REFERENCES

[1] National Institute of Standards and Technology, "Announcing request for nominations for public-key post-quantum cryptographic algorithms," 2016, https://csrc.nist.gov/news/2016/public-key-post-quantum-cryptographic-algorithms.

[2] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, W. Whyte, and Z. Zhang, "Choosing parameters for NTRUEncrypt," *IACR ePrint*, vol. 2015, p. 708, 2015. [Online]. Available: http://eprint.iacr.org/2015/708

[3] O. M. Guillen, T. Pöppelmann, J. M. B. Mera, E. F. Bongenaar, G. Sigl, and J. Sepulveda, "Towards post-quantum security for IoT endpoints with NTRU," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, March 2017, pp. 698–703.

[4] IEEE, "IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices," *IEEE Std 1363.1-2008*, pp. C1–69, March 2009.

[5] D. V. Bailey, D. Coffin, A. J. Elbirt, J. H. Silverman, and A. D. Woodbury, "NTRU in constrained devices," in *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, no. Generators, 2001, pp. 262–272.

[6] A. C. Atici, L. Batina, J. Fan, I. Verbauwhede, and S. B. Örs, "Low-cost implementations of NTRU for pervasive security," in *19th IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2008, July 2-4, 2008, Leuven, Belgium*, 2008, pp. 79–84.

[7] A. A. Kamal and A. M. Youssef, "An FPGA implementation of the NTRUEncrypt cryptosystem," in *Microelectronics (ICM), 2009 International Conference on*. IEEE, 2009, pp. 209–212.

[8] X. Zhan, R. Zhang, Z. Xiong, Z. Zheng, and Z. Liu, "Efficient Implementations of NTRU in Wireless Network," *Communications and Network*, vol. 5, no. 03, p. 485, 2013.

[9] B. Liu and H. Wu, "Efficient architecture and implementation for NTRUEncrypt system," in *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*. IEEE, 2015, pp. 1–4.

[10] ——, "Efficient multiplication architecture over truncated polynomial ring for NTRUEncrypt system," in *IEEE International Symposium on Circuits and Systems, ISCAS 2016, Montréal, QC, Canada, May 22-25, 2016*, 2016, pp. 1174–1177.

[11] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte, "NAEP: Provable security in the presence of decryption failures." *IACR Cryptology ePrint Archive*, vol. 2003, p. 172, 2003.

[12] N. Howgrave-Graham, P. Q. Nguyen, D. Pointcheval, J. Proos, J. H. Silverman, A. Singer, and W. Whyte, "The impact of decryption failures on the security of NTRU encryption," in *Annual International Cryptology Conference*. Springer, 2003, pp. 226–246.

[13] A. de la Piedra, "SHA-256 Core," 2013, https://opencores.org/project/sha256core/.

[14] D. W. Jones, "Modulus without division, a tutorial," 2001. [Online]. Available: http://homepage.cs.uiowa.edu/ jones/bcd/mod.shtml