IMPLEMENTATION AND ANALYSIS OF THE POST-QUANTUM ALGORITHM NTRU PRIME ON AN FPGA

JOSUE KALEB MARIN MOJICA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

IMPLEMENTATION AND ANALYSIS OF THE POST-QUANTUM ALGORITHM NTRU PRIME ON AN FPGA

JOSUE KALEB MARIN MOJICA

A dissertation submitted in partial fulfillment of the requirements for the degree of electronic engineer

Advisor:
WILLIAM ALEXANDER SALAMANCA BECERRA
INGENIERO ELECTRÓNICO. PhD.
Co-Advisor:
CARLOS AUGUSTO FAJARDO ARIZA
INGENIERO ELECTRÓNICO. PhD.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y DE TELECOMUNICACIONES
BUCARAMANGA
2023

ACKNOWLEDGEMENTS

I would like to thank Adrian Marotzke, one of the authors of the 2021 implementation, for providing insights and guidance, addressing my queries about his work and the overarching algorithm.

I'm also deeply appreciative of my parents, whose unwavering support and care have been a constant source of strength. Additionally, I'd like to express my gratitude to my project director William Salamanca for always being more positive than me, God knows I needed that. His consistent assistance has been invaluable during my journey.

-Josue Kaleb Marin Mojica

Dedicated to my families and friends.

TABLE OF CONTENTS

Page.

LIST OF FIGURES

LIST OF TABLES

Page.

LIST OF APPENDICES

RESUMEN

TÍTULO: IMPLEMENTACIÓN Y ANÁLISIS DE DESEMPEÑO DEL ALGORITMO POST CUÁNTICO NTRU PRIME EN FPGA. [*]

AUTORES: JOSUE KALEB MARIN MOJICA [**]

PALABRAS CLAVE: POST-CUÁNTICO, ALGORITMO, NTRU PRIME, FPGA, IMPLEMENTACIÓN

DESCRIPCIÓN:

Así como la tecnología avanza a un paso rápido, también lo hace la necesidad de mantener comunicaciones seguras. Los computadores actuales tienen una defensa eficiente contra ataques cibernéticos gracias a los algoritmos criptográficos, pero, el riesgo a la seguridad de los datos es cada vez más fuerte, especialmente en la era de los computadores cuánticos los métodos de encriptación actuales están en riesgo de ser comprometidos. Por lo tanto, últimamente se ha visto la necesidad de revisar nuevos algoritmos capaces de soportar ataques de este estilo. A estos se les da el nombre de algoritmos post-cuánticos. Teniendo en cuenta lo anterior, debido a que los computadores cuánticos representan un posible problema de seguridad, los algoritmos post-cuánticos han ganado bastante interés en los últimos años, tanto así que National Institute of Standards and Technology (NIST) de Estados Unidos creó un concurso con el fin de estandarizarlos. Este concurso cuenta con tres fases ya realizadas, actualmente entrando a la cuarta, y entre los finalistas de la fase tres se encontró NTRU PRIME, siendo este, el algoritmo a estudiar e implementar en el marco de este proyecto.

Con lo anterior en mente, con la amenaza constante de la computación cuántica es más importante que nunca seguir investigando algoritmos post-cuanticos, la seguridad de nuestros datos es más importante en esta nueva era y la implementación de estos algoritmos es un paso en la dirección correcta para asegurarse de la seguridad y confiabilidad de nuestra información teniendo al frente al campo de la computación cuántica avanzando a paso rápido. Por esto en este proyecto se estudiará el comportamiento de NTRU PRIME en una FPGA y, a su vez, revisar que tan útil seria en una implementación de este estilo a comparación con otras ya realizadas anteriormente

---

[*] Tesis de Pregrado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: William Alexander Salamnca Becerra. PhD.

ABSTRACT

TITLE: IMPLEMENTATION AND ANALYSIS OF THE POST-QUANTUM ALGORITHM NTRU PRIME ON AN FPGA [*]

AUTORES: JOSUE KALEB MARIN MOJICA [**]

KEYWORDS: NTRU PRIME, POST-QUANTUM CRYPTOGRAPHY, FPGA, STREAMLINED NTRU PRIME, IMPLEMENTATION.

DESCRIPTION:

Just as technology advances at a rapid pace, so does the need to maintain secure communications. Today's computers have an efficient defense against cybernetic attacks thanks to cryptographic algorithms like RSA, but the risk to data security is stronger everyday and, specially in the era of quantum computers, current encryption methods are at risk of being compromised. Therefore, lately there has been a need for new algorithms capable of withstanding this kind of attack. These are called post-quantum algorithms. These algorithms have gained so much attention that the national institute of standards and technology (NIST) created a competition in order to standardize post-quantum algorithms lasting from 2016 until 2022. The winner of the competition was an algorithm called CRYSTALS-Kyber, a lattice-based system, probably the most researched type right now because of its security possibilities. While the selected algorithm by NIST will not be the focus of this paper, we will study an algorithm that also uses a lattice-based system and was a finalist in the third round of the competition, NTRU Prime. NTRU Prime is a derivation from another post-quantum algorithm called NTRU. The main difference between the two is that NTRU Prime only uses prime numbers for most of the parameters. The algorithm was initially proposed in 2016 and was seen as a good alternative for NTRU due to the attack field reduction that it offered compared to its predecessor. This algorithm has two alternates depending on how the public key is generated, namely, Streamlined NTRU Prime and NTRU Lprime. For this work, an implementation of streamlined NTRU Prime in an FPGA will be presented as well as an analysis of the results.

---

[*] Undergraduate Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: William Alexander Salamnca Becerra. PhD.

INTRODUCTION

Just as technology advances at a rapid pace, so does the need to maintain secure communications. Today's computers have an efficient defense against cybernetic attacks thanks to cryptographic algorithms like RSA, but the risk to data security is stronger everyday and, specially in the era of quantum computers, current encryption methods are at risk of being compromised. Therefore, lately there has been a need for new algorithms capable of withstanding this kind of attack. These are called post-quantum algorithms.

These algorithms have gained so much attention that the national institute of standards and technology (NIST) [1] created a competition in order to standardize post-quantum algorithms lasting from 2016 until 2022. The winner of the competition was an algorithm called CRYSTALS-Kyber, a lattice-based system, probably the most researched type right now because of its security possibilities. While the selected algorithm by NIST [2] will not be the focus of this paper, we will study an algorithm that also uses a lattice-based system and was a finalist in the third round of the competition, NTRU Prime.

NTRU Prime is a derivation from another post-quantum algorithm called NTRU. The main difference between the two is that NTRU Prime only uses prime numbers for most of the parameters. The algorithm was initially proposed in 2016 [3] and was seen

---

[1] National Institute OF STANDARDS and TECHNOLOGY. *Post-Quantum Cryp- tography standirization*. 2017. URL: https://csrc.nist.gov/projects/post-%20quantum-cryptography/post-quantum-cryptography-standardization.

[2] National Institute OF STANDARDS and TECHNOLOGY. *Selected Algorithms*. 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[3] Lange BERNSTEIN Chuengsatiansup and VREDENDAAL. "NTRU Prime: reducing attack surface at a low cost". In: *Selected Areas in Cryptography 2017* 1.1 (2016), pp. 230–260.

as a good alternative for NTRU due to the attack field reduction that it offered compared to its predecessor.

This algorithm has two alternates depending on how the public key is generated, namely, Streamlined NTRU Prime and NTRU Lprime. For this paper, an implementation of streamlined NTRU Prime in an FPGA will be presented as well as an analysis of the results. In the following chapters II and III, a review of mathematical background and previous works is given. In chapter IV, a CPU implementation is introduced, in chapter V and VI the proposed FPGA design is explained and the results of the implementation are shown and compared against other similar implementations. Lastly, a final analysis of the results is given in section VII.

1. DEFINITIONS AND BACKGROUNDS

1.1. MATHEMATICAL BASICS

Before starting with the actual definitions of the algorithm, there are some mathematical aspects that must be mentioned:

- Lattices: In the context of quantum cryptography, a lattice serves as a fundamental mathematical construct, representing a collection of points distributed across a multidimensional space. Lattices play an important role in cryptography, whether it's for algorithmic design, such as key generation, or for ensuring the security of cryptographic systems. Prominent cryptographic algorithms like NTRU Prime, CRYSTALS-Kyber, and NTRU harness the properties of lattices.

  In traditional lattice cryptography, these points are often defined by integer basis vectors. However, in NTRU Prime, lattice points find their definition within the realm of polynomial rings.

- Rings: A ring is a set of elements that have 3 arithmetic operations: addition, subtraction and multiplication. In abstract algebra, the area that focuses on groups, rings and fields, the terminology changes, instead of subtraction there are additive inverses which means that in rings and fields, when we refer to addition, substraction is included.

  For every operation the set is closed, which means that if two elements are added the result is another element in the set, and if you multiply two elements the result will also be another element in the set. A ring used by NTRU prime has the following notation: $(Z/3)[x]/(x^p - x - 1)$, which represent the polynomials with degree less than p and coefficients modulo 3.

- Multiplicative inverse: The Multiplicative inverse of an element x is the one that multiplied by x yields the multiplicative identity 1.

- Fields: Fields work the same way as a ring, the main difference is that while rings are a set of numbers that is closed for multiplication and addition, fields are also closed for division, meaning, that the elements in the field must have multiplicative inverses. In essence, fields are rings with multiplicative inverses.

- Modulo: Modulo operations are the remainder of a division between two elements x and n, also denoted as $x \, mod \, n$. In this paper, these two elements, x and n, will take two forms, integers and polynomials. This duality comes from the fact that NTRU Prime operates within the context of rings and fields defined over polynomials with coefficients subject to modulo prime numbers.

1.2. BASICS OF NTRU PRIME

NTRU Prime, as well as NTRU, is a lattice based algorithm but, while NTRU is based on a ring denoted by $(Z/q)[x]/(x^p - 1)$ with p being a prime number and q being a power of 2, NTRU Prime is defined over the field $Rq = (Z/q)[x]/(x^p - x - 1)$ where both, p and q, are primes.

The change of the polynomial in NTRU Prime is a response to attacks that exploit special structures of the rings employed within the NTRU cryptosystem. The addition of another term allows NTRU Prime to increase its resistance to attacks, making it a less favorable target for certain vulnerabilities when compared to NTRU.

While the notation is very similar to that of the ring in NTRU, what makes Rq a field is the fact that it's defined by a prime number, which means that every element in Rq has a multiplicative inverse [4] . This is why NTRU Prime is considered to have a smaller

---

[4] David FORNEY. "chapter 7: Introduction to finite fields". In: *Lecture notes course 6.451 ("Principals of digital communication II") MIT* (2005).

attack field compared to NTRU.

1.3. MODULAR ARITHMETIC

Arithmetic operations, like polynomial multiplication and addition, follow modular arithmetic inside the rings and fields NTRU Prime uses. Both addition and multiplication use the basics of polynomial addition and multiplication. The main difference is that after the operation is done, modulo operations must be performed on the resulting polynomials. This means that, being $m(x)$ and $n(x)$ both polynomials in Rq with degree p-1, then for addition

$$c(x) = \sum_{i=1}^{p-1} (m_i + n_i)x^i \, mod \, q$$

where every coefficient in $c(x)$ is mod q, i= 0,1,2, ... p-1.
On the other hand, for multiplication, using the same $m(x)$ and $n(x)$ the operation changes, first a polynomial multiplication is performed:

$$d(x) = \sum_{0 \leq i,j \leq p-1} m_i n_j x^{i+j}$$

After the multiplication, the resulting polynomial $d(x)$ is divided by the polynomial that defines the ring, in this case, $(x^p - x - 1)$ obtaining the remainder in which every coefficient is then reduced mod q.
To explain this topic further, let p=3 and q=7, also $m(x) = 2x^2 + 3x + 1$ and $n(x) = 4x^2 + 6x + 5$ be two elements of the field Rq. Then the result of adding these two polynomials would be:

$$\begin{aligned} m(x) + n(x) &= (6x^2 + 9x + 6) \, mod \, 7 \\ &= 6x^2 + 2x + 6 \end{aligned}$$

This result can also be denoted, by using modular congruence, as:

$$m(x) + n(x) \equiv 6x^2 + 2x + 6 \pmod 7$$

And for multiplication:

$$m(x) * n(x) = 8x^4 + 24x^3 + 32x^2 + 21x + 5 \, mod \, (x^3 - x - 1) \, mod \, 7$$

$$m(x) * n(x) = 5x^2 + 4x + 6$$

1.4. INVERSION USING THE EXTENDED EUCLIDEAN ALGORITHM

Multiplicative inverses can be very challenging to find depending on the field, for example, in the field of $Z/5$, which are the set of integers mod 5, consider the operation 3*2. This multiplication yields 6, but in the field the modulo 5 operation is applied to find the result: $2 * 3 = 6 \, mod \, 5 = 1$ and since 1 is the multiplicative identity (just like in regular algebra, where dividing a number by itself equals 1) 2 would be the multiplicative inverse of 3. On the other hand, for a field like Rq finding a multiplicative inverse of a polynomial can be rather complicated and time consuming, this is the point where the extended euclidean algorithm comes in.
The extended euclidean method, is based on a variation of the Bézout's identity equation, which states that for any two elements a and n within a ring or field, there exist elements h and k such that:

$$a * h + k * n = 1$$

In the equation, a is the element for which we want to find the inverse, n is the modulus and h is the multiplicative inverse of a. The parameter k is also a result given by the EGCD but for the calculation of the multiplicative inverse is not important. This means

that for our context the Bézout's identity can be written as:

$$a * h \equiv 1 \pmod{n}$$

Now, for a general polynomial m(x) that belongs to the field Rq:

$$m(x) * h(x) + k(x) * (x^p - x - 1) = 1$$

In order to find h, the algorithm is as follows:

1. Assign 4 polynomials the following values, $x[0] = 0$, $y[0] = 1$, $u[0] = 1$, $v[0] = 0$, the rest coefficients are 0.
2. Assigning $a = (x^p - x - 1)$ and $b = m(x)$ divide a by b; after the operation, assign b to a and the remainder of the division to b and perform the division again.
3. After every division: $m = x - u * Q$ and $n = y - v * Q$, Q being the quotient of every division. After this assign u to x, v to y, m to u and n to v.
4. Steps 2 and 3 are done until b, or the remainder of the division between a and b is 0.
5. In this case, the output values would include the greatest common divisor (GCD), represented as b, as well as the values of x and y.

For our example, the multiplicative inverse of $m(x)$ would be the resulting $x \, mod \, q$.

2. STREAMLINED NTRU PRIME

As mentioned before, out of the two alternates NTRU Prime has, Streamlined NTRU Prime will be implemented. With this in mind, in order to define the algorithm, 2 additional rings are presented: R, denoting the ring $Z[x]/(x^p - x - 1)$, and $R/3$, representing the ring $(Z/3)[x]/(x^p - x - 1)$ [3].

In the context of the algorithm, a small element in R is a polynomial that has coefficients limited to the set $\{-1, 0, 1\}$. On the other hand, when referring to a t-small element in R is one where the coefficients remain within $\{-1, 0, 1\}$ as well but 2t of these coefficients must be nonzero.

There are 3 parameters in Streamlined NTRU Prime: p, q and t. All of the parameters are integers but p and q are also prime and can be chosen as long as $t \geq 1$, $p \geq 3t$ and $q \geq 32t + 1$.

Streamlined NTRU Prime can be divided in 3 parts, key generation, encryption and decryption. Before starting key generation the parameters must be selected, this can be considered the initial step.

For better understanding of the algorithm, every step, including parameter selection, will be explained in this section.

1. Parameter selection: Some parameters that are considered secure according to the original paper [3] are:

   - $p = 467$, $q = 3911$, $t = 122$
   - $p = 569$, $q = 1579$, $t = 49$
   - $p = 619$, $q = 9397$, $t = 206$
   - $p = 677$, $q = 3251$, $t = 101$
   - $p = 757$, $q = 3727$, $t = 116$

- $p = 827$, $q = 19081$, $t = 275$
- $p = 919$, $q = 14771$, $t = 306$

2. Key generation: NTRU Prime is an asymmetric cryptographic algorithm, meaning it has 2 different keys: The public key, used for encrypting the message. As well as the private key, used for decryption.
    - First: Create a random small polynomial g in R, g must be invertible in $R/3$, if not, repeat step.
    - Second: Create a random t-small polynomial f in R.
    - Third: Compute $h = g/3f$ in R/q.
    - Fourth: Encode h, the public key is $\underline{h}$.
    - Fifth: Private key is f in R and $1/g$ in $R/3$. Encode f, $1/g$.
3. Encapsulation: The encryption of the message occurs in this stage, using the public key.
    - First: Decode the public key and private key.
    - Second: Create a random t-small polynomial r in R.
    - Third: Compute $h * r$ in R/q.
    - Fourth: Obtain c by rounding all coefficients of the resulting polynomial to the nearest multiple of 3.
    - Fifth: Encode c and r to obtain $\underline{c}$ and $\underline{r}$.
    - Sixth: Hash r, obtaining left half C and right half K (session key).
    - Seventh: Ciphertext is the concatenation of C and $\underline{c}$.
4. Decapsulation: In order to decrypt the message, the public key is used, in case any other key is used, the algorithm will not return the original message.

- First: Decode c.
- Second: Multiply by 3f in R/q.
- Third: compute $e = 3f * c \mod 3$ to obtain a polynomial e in $R/3$.
- Fourth: Compute $e * 1/g$ in $R/3$ and lift to R, obtaining r'.
- Fifth: Repeat encapsulation for r' to get c', C' and K'.
- Sixth: if r' is t-small, $c == c'$, $C = C'$, output K'. Otherwise output false.

Note that in both encapsulation and decapsulation an algorithm for encoding and decoding is used [5], it is specific to streamlined NTRU Prime to transform polynomials in $R/3$ and R/q to and from byte strings.
It's important to emphasize that, for the purposes of this paper, the encoding and decoding algorithms will not be implemented. This decision is made in recognition of the fact that these operations are not the primary focus of this implementation, and implementing them would demand substantial resources and time.

[5] Marotzke CHEN Peng. "Streamlined NTRU Prime on FPGA". in: *Journal of Cryptographic Engineering* (2022).

3. REVIEW OF PREVIOUS WORKS

While there exist multiple implementations of NTRU Prime on CPU, both streamlined NTRU and NTRU Lprime, to our best knowledge, as of October 2023, there are three streamlined NTRU Prime implementations on FPGA.
While our focus for this project is not CPU implementation, it is important to mention that the first NTRU Prime implementation [3] was a valuable reference for our work. Therefore, a review on that paper will be presented later in this section as well as on two of the above mentioned FPGA implementations, the last of them being published in 2021.

3.1. NTRU PRIME BY BERNSTEIN, CHUENGSATIANSUP, LANGE, VAN VREDENDAAL

In this article, NTRU Prime was proposed based on an existing post-quantum algorithm called NTRU, which was proposed in 1996. Both algorithms are based on algebraic groups and rings for encryption but, lately, there have been some new attacks that have shown some faults in data protection of these algorithms that exploit special structures in algebraic rings.
With this in mind, the authors designed NTRU prime so that it still uses rings but, without the structures that affected the security of algorithms like NTRU. In the same article, the authors propose ”Streamlined NTRU Prime”, which, from an implementation standpoint makes the algorithm more efficient. It’s important to mention that this implementation was done on CPU.
This paper explains how the algorithm works, its parts, the parameters it uses and the steps of ”streamlined NTRU Prime”, all of which were very important for this project because it helped clear some doubts about the implementation of this algorithm.

It is also mentioned in the article that the main objective of post-quantum cryptography is data encryption so that a quantum computer could not decrypt it, so in order to fulfill the goal it is necessary to study the algorithms, implement them and attack them to test their resistance.

3.2. EFFICIENT MULTIPLIER AND FPGA IMPLEMENTATION FOR NTRU PRIME BY HUAPENG, GAO

Originally the topic of this article was presented as the doctorate dissertation by Xi Gao being presented as the first NTRU prime implementation in FPGA. In the article, and the dissertation, the authors mentioned the fact that NTRU prime was a finalist in the third round of the NIST competition, this coupled with it not having been implemented in FPGA before, as one of the reason why they chose it for the dissertation.

One of the main operations that are needed in order to implement the algorithm is polynomial multiplication, specially since this is the one that tends to cause the most bottlenecks for algorithms. This is why the authors start the article by explaining how polynomial arithmetic with abstract algebra works, including not only multiplication but addition as well.

Xi Gao y Huapeng Wu, decided to implement the ”streamlined NTRU Prime” alternate but they used a different multiplier designed and proposed by themselves in order to make the implementation more efficient. The multiplier proposed exploits the FPGA capacities, specially the linear feed-back shift registers (LSFR) which are shift registers in which the feedback is a linear function of the last state.

Thanks to this paper the abstract algebra arithmetic operations were better understood as well as the main operations to focus when designing the implementation. The paper also mentions the results of their implementations, which are a good reference for this project’s own results.

3.3. STREAMLINED NTRU PRIME ON FPGA BY PEN, MAROTZKE, TSAI, YANG, CHEN

As mentioned before, in 2016, Bernstein, Chuengsatiansup, Lange and van Vredendaal proposed an implementation of NTRU prime called ”Streamlined NTRU prime”. In the 2016 article [3] the implementation was done on CPU. With this in mind, Pen, Marotzke, Tsai, Yang and Chen, the authors of the article, decided to implement the same algorithm but on an FPGA. Using new methods to improve the performance.
Normally, the focus is on the efficiency of the multiplication and polynomial inversion, because this are normally the areas where the implementations tend to slow done. Taking the former into account, the authors proposed, along with other methods not directly related to multiplication and inversion, two techniques, each one centered around each operation.
The first method used is called “Schoolbook polynomial multiplier”, it is not new since it had already been used before for NTRU Prime as well as for other cryptography algorithms. In this implementation the authors used this technique in order to reduce the quantity of bit operations needed which leads to a small footprint on the FPGA.
The second method mentioned in the paper is called ”batch inversion” and, as the name suggests it is used for polynomial inversion, in this technique instead of doing n inversions in a ring, only one is done along with 3n-3 multiplications, this number is given by the montgomery trick, which is the basis of the method. This method can’t use the “Schoolbook polynomial multiplier” for the multiplications because this algorithm is designed for polynomials not in the same ring and the method multiplies polynomials inside the same rings. So, multiplications are done with the discrete Fourier transform (DFT) but, applied to polynomials is called Number theoretic transform (NTT). The NTT allows the multiplication of polynomials that are in the same ring, which makes it a very useful tool for the acceleration of the inversion process.
This article is very important for this project because it provides a good basis on the

implementation of this algorithm with an FPGA, the mathematical explanations needed to understand the algorithm better as well as what aspects to take into account when designing the NTRU prime implementation of this project.

4. CPU IMPLEMENTATION

As reference, an implementation was written in python in order to compare the results with those given by the FPGA implementation, the software sagemath [6] was also used for this propose.

From the original authors, and collaborators, there are 2 separate CPU implementations in sagemath [3] [7], both submitted to the NIST competition. This implementations were also used as reference for the design on the FPGA, specially for the rounding and lift algorithms.

For our implementation, the algorithms for multiplication, division and inversion were taken from [8]. While this implementation is not efficient for big numbers, it is useful for testing our own hardware design of multiplication, division and inversion, as well as the result of Key generation, Encapsulation and Decapsulation.

---

[6] William STEIN. 2005.

[7] NTRU Prime SOFTWARE. *Reference implementation in Sage*. 2022. URL: https://ntruprime.cr.yp.to/software.html.

[8] William J BUCHANAN. *Inverse Polynomial (mod p)*. 2023. URL: https://asecuritysite.com/pqc/inv%5C_poly (visited on 2023).

5. PROPOSED ARCHITECTURE FOR NTRU PRIME

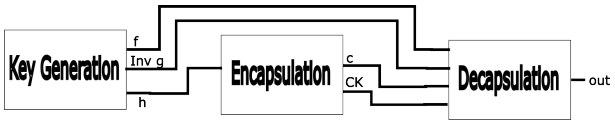

Figure 1. General architecture for streamlined NTRU Prime

As seen on figure 1, NTRU Prime is divided in key generation, Encapsulation (encryption) and Decapsulation (Decryption). The focus of the design was different for each part, but in general the main goal behind each architecture was functionality.

5.1. KEY GENERATION

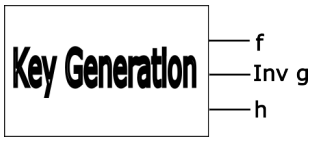

Figure 2. Key generation output

The key generation, as depicted in figure 2, produces 3 outputs, which are divided into two keys: the private key, consisting of f and 1/g, and the public key represented by the polynomial h.

The generation of keys is the aspect of NTRU prime that takes the most time. This is mainly due to that fact that the generation of two random polynomials [9] (f and g) and two inverse operations, using the extended euclidean algorithm (EGCD), are done inside the module in order to calculate the private key (f and 1/g) and the pubic key (h).

---

[9] Jacob HAMMOND. “FPGA Random Number Generator”. In: *arXiv, Paper arXiv:2209.04423, 2022* (2022).

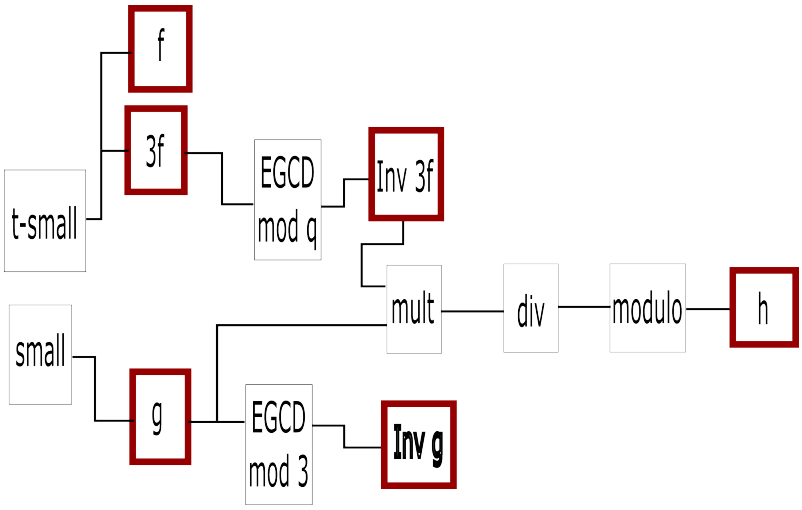

Figure 3. Key generation architecture

The public key h is particularly time-expensive because it is the multiplication of $g * 1/3f$ in R/q. This means that first the inverse of 3f needs to be calculated and then multiplied by g. Since this operation outputs a result that is not inside the ring R/q, a modulo reduction must be performed. This reduction involves two main stages shown in figure 3: first a division by the polynomial $(x^p - x - 1)$, which outputs a remainder that has coefficients that are not obligatorily modulo q. Subsequently, another modulo reduction is done on the coefficients of the polynomial.

Another aspect to take into account are the memories, for the design all the memories used for Key generation are distributed RAM with 13 data bits and 11 direction bits, except for the output of the multiplication, which is of 26 data bits. The dimension of the memories is a constant throughout the entire algorithm.

5.2. ENCAPSULATION

The encapsulation module takes the public key generated by the key Gen module as an input, and outputs the ciphertext: the polynomial c and the hash CK. This is depicted in Figure 4.

Figure 4. Encapsulation inputs and outputs

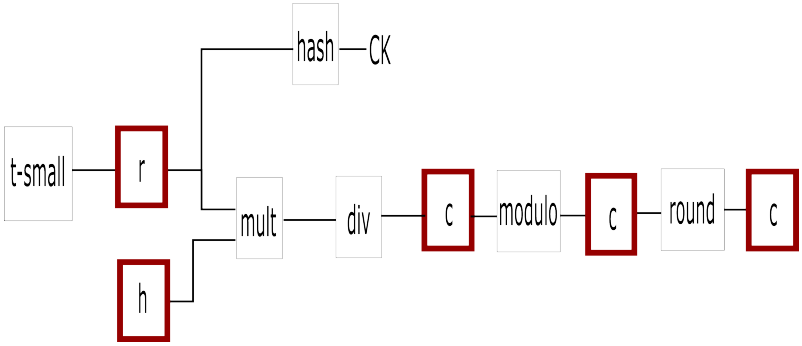

Figure 5. Encapsulation architecture

Figure 5 shows there are two main operations performed in encapsulation, the first one is the multiplication in R/q of the public key, h, and a generated t-small polynomial r. As in key generation, modulo reduction is also needed in order to perform the multiplication in R/q.

The second one is the hash operation on r, which is essential for this algorithm since the output hash, CK, will be used for checking if the results given in the next section, decapsulation, are correct. The output CK is a hash of 256 bits, it will be divided in 2, the confirmation key C and the session key K.

The output of the multiplication is also rounded to the closest multiple of 3, which gives us the output polynomial c.

5.3. DECAPSULATION

The decapsulation module involves four inputs, as illustrated in Figure 6. The core components of this module are the polynomial 'c' and the hash 'C', which together constitute the ciphertext. On the flip side, we have 'f' and the inverse of 'g,' forming the

Figure 6. Decapsulation inputs and outputs

private key.

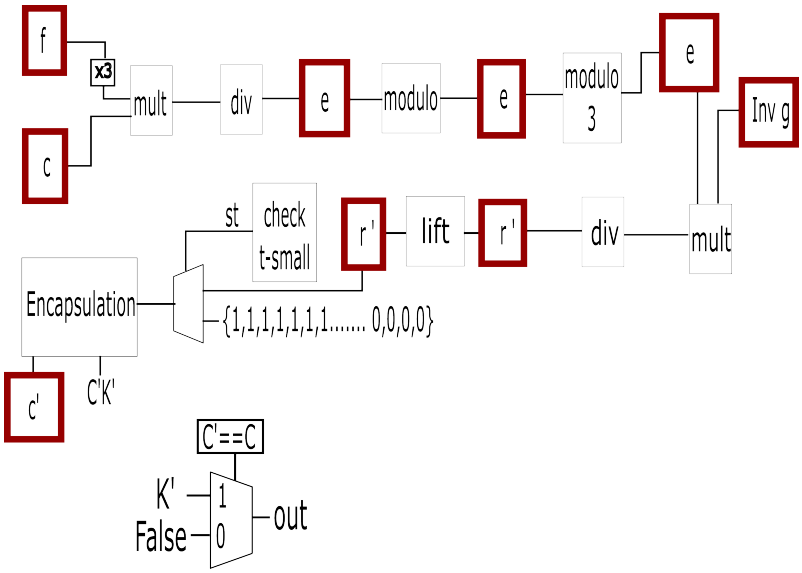

Figure 7. Decapsulation architecture

There are 2 multiplications performed inside the module, one in R/q and another one in $R/3$, which means 2 different modulo reductions.

The first multiplication takes the input polynomials c and 3f and multiplies them in R/q. After the resulting polynomial is already in R/q, another modulo reduction is carried out but this time in $R/3$. This is done to prepare the polynomial for the next multiplication.

The second multiplication takes the result of the previous multiplication and the input 1/g and multiplies them in $R/3$. After the multiplication and the following modulo reduction, the polynomial undergoes a process known as lifting, which involves converting the polynomial from the $R/3$ ring to the R ring. Resulting in the polynomial denoted as r'.

After, we need to check whether the polynomial is t-small or not, if it is, r' enters the

encapsulation again. This should output the same C obtained in the original encapsulation and if it does, the k obtained in the encapsulation of r' is the output of the decapsulation.
On the other hand, if r' is not t-small, a polynomial with the first 2*t coefficients as 1 and the rest as 0, enters the encapsulation. This is done in order to prevent an attacker from learning anything about the private key.
Having covered the key generation, encapsulation, and decapsulation modules, it's essential to delve into the foundational core modules of the NTRU Prime implementation. These core modules include: multiplication, division, Inversion in R/q and $R/3$, modulo reduction and SHA-512.

5.4. POLYNOMIAL MULTIPLICATION

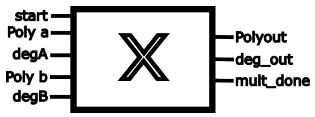

Figure 8. Polynomial multiplication module inputs and outputs

As seen on figure 8, multiplication has 2 polynomial inputs and 2 integer inputs. The integers represent the degrees of each polynomial.
As mentioned before, multiplication can be a bottleneck for the algorithm, with this in mind we used the same method that was also used previously [5] [10] , but unlike the previous implementation, no parallelism was done for the multiplications, instead the schoolbook multiplier was implemented just as is.
For this paper, the main focus was to design a functional multiplication algorithm following the method, as shown in figure 9, not focusing on the timing.

---

[10] M. B. İLTER and M. CENK. *Efficient Big Integer Multiplication in Cryptography*. 2017. URL: https://hdl.handle.net/11511/75405 (visited on 2023).

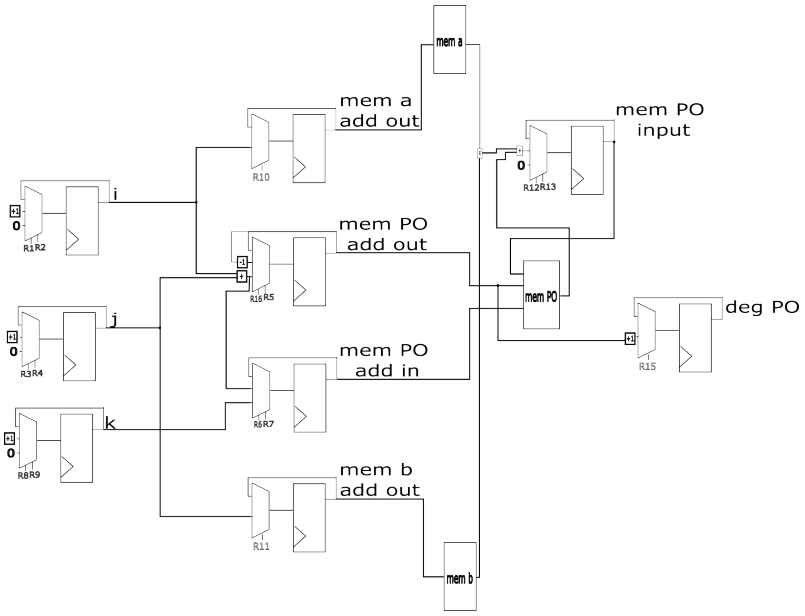

Figure 9. Polynomial multiplication architecture

The multiplication of 2 polynomials with our design will be performed as:

$$m(x) * n(x) = m_0 n_0 + x(m_0 n_1 + n_0 m_1) + x^2 m_1 n_1$$

The module outputs the polynomial and a register with the degree of the polynomial, as shown in figure 8, which is important for division and the EGCD.

5.5. POLYNOMIAL DIVISION

Figure 10. Polynomial division module inputs and outputs

For the design of the division module, three aspects were taken into account: the modulo of the division, memory efficiency and timing efficiency.
As with numerical division, the output of the division is a quotient and a remainder as

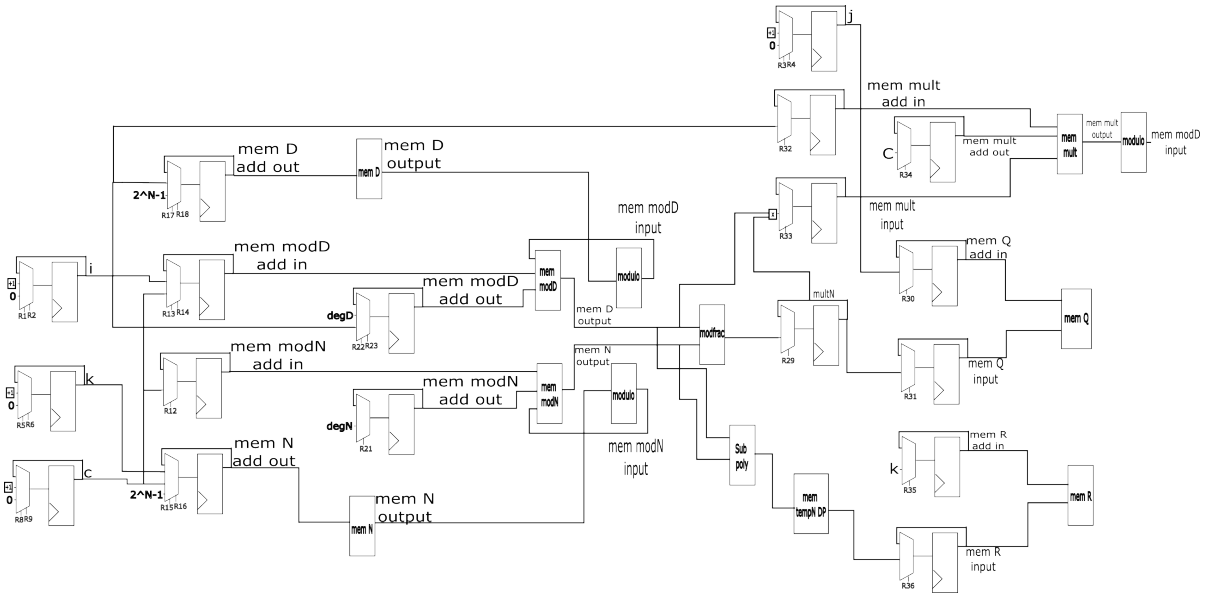

Figure 11. Division architecture

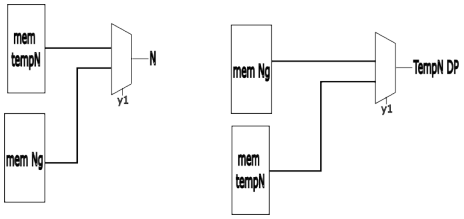

Figure 12. Memory arrangement architecture

shown in figure 10. The output quotient is either in R/q or $R/3$, depending on the modulo.

In order to perform a polynomial division, the coefficient of the highest degree of the dividend is divided by the coefficient of the highest degree of the divisor. The result is the coefficient of the highest degree of the quotient. This in itself does not present a problem, except when used in the EGCD, since the quotient will be multiplied by other polynomials that will also have fractions as coefficients. This means that by the end of the EGCD, the inverse will have enormous fractions as coefficients, not to mention that fraction multiplications are more complicated to perform than integer multiplications. Since modulo reductions must be performed on the inverse in order for the polynomial

to be inside the ring, the task will be both resource and time consuming. In section 2, the operations of the EGCD are explained further.
So, in order to bypass the problem fractions present, the modulo reduction is performed along with the division. Thus two modulo reductions on polynomials will be performed as well as one modulo operation on the fraction coefficient of the quotient. This can be seen in the polynomial division architecture in figure 11
As mentioned before the other aspect also taken into account was the memory efficiency. Since polynomial division is basically subtracting the dividend by the product of the divisor and the previously obtained term of the quotient. After the operation is performed, if the degree of the result is not lower than that of the divisor, the entire division process is reiterated. This time, however, the result of the previous subtraction becomes the new dividend. This is better explained with an example:

- Being one polynomial $P(x) = 2x^3 + 3x^2 + 5$ and $Q(x) = x^2 + 2x + 1$, the first term of the quotient is $2x$.
- $Q(x) = x^2 + 2x + 1$ multiplied by the quotient $2x$ produces $2x^3 + 4x^2 + 2x$.
- The result of the multiplication is then subtracted to the dividend, in this case $P(x) = 2x^3 + 3x^2 + 5$. Yielding the remainder: $-x^2 - 2x + 5$.
- Since the degree of the remainder $-x^2 - 2x + 5$ is not less than that of the divisor (Q(x)), we need to reperform the division. This time, the new dividend is the remainder.
- We continue this process iteratively until the degree of the remainder becomes less than that of the divisor.

In a hardware implementation every polynomial is a memory, so copying the result to the dividend memory used initially would be time consuming, and a memory specifically used only as dividend would need to be added. So we decided to, instead, rotate two

memories between the dividend and the result of the subtraction as shown in figure 12, being N, the dividend and Temp N the result of the subtraction.

5.6. POLYNOMIAL INVERSION

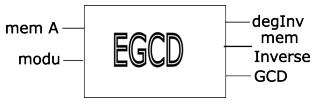

Figure 13. EGCD module inputs and outputs

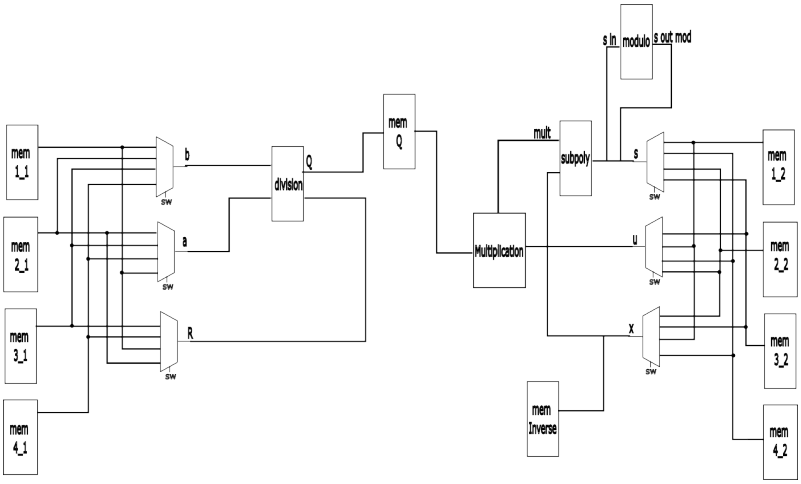

Figure 14. EGCD architecture

In figure 13, it is shown that the EGCD only has one memory input, this is because the memory that would be b, is the polynomial over which all the rings and fields used in the algorithm are defined: $x^p - x - 1$. Additionally, while the EGCD outputs the GCD, x, which is the inverse, and y. We only need the Inverse and the GCD so the EGCD is adapted to calculate and output only those two results.

Since the EGCD uses outside modules for all of its operations, except modulo reduction of S, our main focus when designing the architecture of the EGCD was handling the memories in the most efficient way possible. As mentioned in section 2, a, b, r, x, u and S, which is the equivalent of m, all interchange values.

Since copying memories would not be time efficient, rotating the memories is actually a better alternative. The arranged memories are shown in figure 14, along with the operations connected to each. The order in which they change follows the order mentioned in section two and can be seen in table 1, for a, b and r, and table 2, for S, u, x.

a	b	r	-	sw
2_1	1_1	3_1	4_1	00
3_1	2_1	4_1	1_1	01
4_1	3_1	1_1	2_1	10
1_1	4_1	2_1	3_1	11

Table 1. Changing memories for division in EGCD

S	x	u	-	sw
1_2	2_2	3_2	4_2	00
4_2	3_2	1_2	2_2	01
2_2	1_2	4_2	3_2	10
3_2	4_2	2_2	1_2	11

Table 2. Changing memories for the Inverse

The hyphen symbol in the table means those memories are resetting, this is needed since r and S need to be blank because they are the result of division and subtraction, respectively.

5.7. MODULO REDUCTION

There are 2 modulo reduction modules that are used for this algorithm, one is for integers, used in the modulo reduction of all the polynomials, and the other one is for fractions, only used in the division module.

The modulo for integers uses the following architecture:

In this module, the two inputs are the modulo M and the number N, as depicted in figure 15. Normally these two numbers size in bits will be different, since N will be bigger.

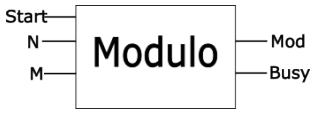

Figure 15. Integer modulo module inputs and outputs

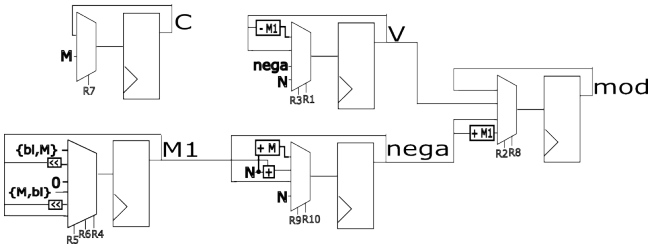

Figure 16. Integer modulo architecture

Since using the built in modulo function in verilog would be more time and resource consuming. For modulo reduction, figure 16 shows the method used is based on initially assigning N to V and aligning the bits of the modulo value M with the input number N, effectively padding M on the right. If M is smaller than V then a subtraction is performed by the modified modulo, referred to as M1. In case it is not, M1 is right-shifted iteratively until it becomes smaller than V. Afterwards the subtraction operation is performed. This process is done until the result of the subtraction is less than the original M.

As an example, let $N = 10000010$ (130) and $M = 11$ (3):

1. align the bits of the modulo, getting $M1 = 11000000$ (192).
2. Shift M1 to the right until it is smaller than V. This operation gives $M1 = 01100000$ (96).
3. Subtracting M1 from V, obtaining $V = 00100010$ (34).
4. Shift M1 to the right until it is smaller than V. This operation gives $M1 = 00011000$ (24).
5. Subtract M1 from V, obtaining $V = 00001010$ (10).

6. Since V is not smaller than M, M1 is shifted again, obtaining $M1 = 00000110$ (6).
7. Subtract M1 from V, obtaining $V = 00000100$ (4).
8. Since V is not smaller than M, M1 is shifted again, obtaining $M1 = 00000011$ (3).
9. Subtract M1 from V, obtaining $V = 00000001$ (1).
10. Since V is smaller than M: $130 \, mod \, 3 = 1$

For negative numbers, initially the logic changes, instead of padding M on the right and shifting it to the right, the operations are done in the opposite direction. In this case, N is assigned to nega instead of V. Subsequently, the process largely parallels the method used for positive numbers, with the exception that nega is subtracted from M1 without checking which if it is smaller. This continues until the result becomes positive. If it is also less than the modulo then it outputs that number. However, if it exceeds the modulo, the process resets, and it proceeds with the calculation of the modulo for positive numbers, following the same logic as described earlier.
The modulo for fractions used in division uses the following architecture:

Figure 17. Fractions modulo module inputs and outputs

For this case, the module takes three inputs, as seen in figure 17, the numerator and denominator of the fraction, and the value of the modulo.The process begins by computing the inverse of the denominator modulo the input modu. Subsequently, the inverse is multiplied by the numerator. The outcome of the multiplication undergoes a modulo reduction and the result becomes the output Modfrac.
In order to calculate the inverse of the denominator, the same algorithm used for calculating the polynomial inverse, the EGCD, is also used but this time applied to integers.

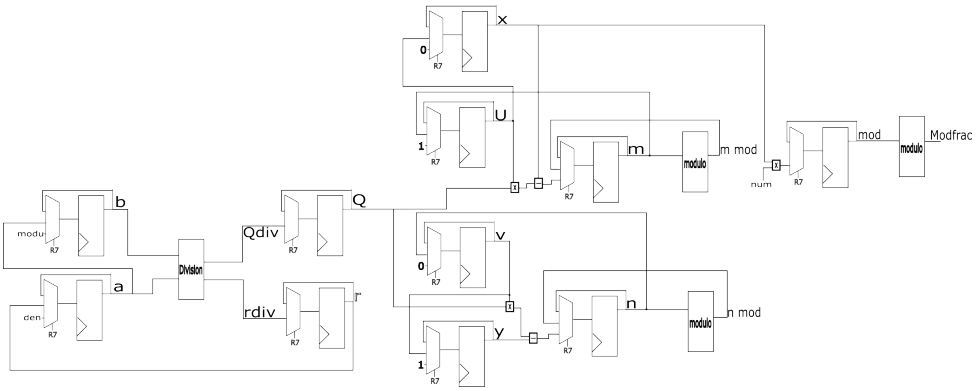

Figure 18. Fractions modulo architecture

The architecture detailed in Figure 18 adheres to the same principles and steps outlined in Section 2 for this method. It is worth noting that the division module used is not an original creation but is based on an open-source division module [11].

5.8. SHA-512 HASH

Hashing functions are normally used in cryptography due to its ability to take a variable-length input and convert it into a fixed-length message. This helps ensure the authenticity of a piece of data since if there is a single change in the original message, the output hash will be completely different.

Streamlined NTRU Prime uses the SHA-512 hash function in order to generate the hash of the t-small polynomial r, generated in encapsulation. Half of the resulting hash will be used in the cyphertext, along with the polynomial c, as the confirmation hash C, the other half is used as the session key. It is also important to mention that while the SHA-512 function outputs a hash of 512 bits, only 256 bits will be used which means C, the confirmation hash, and the session key will each be of 128 bits.

The hash implementation used for this project is not original, instead, an open source

---

[11] Jan SUMMER. *Verilog Iterative division*. 2020. URL: `https://github.com/jasommer/verilog_iterative_division/blob/master/division_core.v` (visited on 2020).

implementation [12] is used with some modifications in order to adapt it to our design. This core has been used before in other FPGA designs.

[12] Kindgren STRÖMBERGSON Nordmark. *SHA512*. 2021. URL: https://github.com/secworks/sha512 (visited on 2021).

6. FPGA IMPLEMENTATION

Due to time and resource constraints, by the time this paper is submitted we could not implement it in an actual FPGA, but the design was simulated, synthesized and implemented with the vivado software and we were able to get the following results with two different set of parameters.

6.1. SIMULATION AND SYNTHESIS SETTINGS

- Verilog HDL is used as the design language.
- Artix 7 xc7a200tfbg484-3 was chosen as the target device.
- The frequency is set at 200 MHz.

6.2. SIMULATION AND SYNTHESIS RESULTS

Parameters	Module	slices	LUT	Mem LUT	DSP	Time (mS)
p=677	Key Gen	6221	19304	28800	21	475.4
q=3251	Encap	10949	10959	5760	5	108.5
t=101	Decap	13760	18629	14976	15	295.4
p=757	Key Gen	6221	19420	28800	21	565.5
q=3727	Encap	12999	10949	5760	5	136.6
t=116	Decap	15810	18598	14976	15	362.9

Table 3. Our design syntethized on an Xilinx Artix 7 FPGA with the simulations results

The results in table 3 show that our implementation takes more than 50% of the total time used by the algorithm for Key Generation on both parameter sets. This makes sense since this is the module tasked with the generation of the polynomials small and t-small as well as calculating the polynomial inverses.

Operation	Key gen p=757	Encap p=757	Decap p=757
Inversion in R/q	427.238 mS	-	-
Inversion in R/3	100 mS	-	-
Multiplication	11.5 mS	11.4 mS	23 mS
Division	124.5 mS	124.6 mS	328.04 mS

Table 4. Time taken by the main operations in each module

Another aspect to mention, as evidenced by the data presented in Table 4, is that in this implementation, the operation that takes the most time, after the inverses, is the modulo reduction of polynomials after a multiplication. This process includes division and modulo, mainly division, that takes more than 90% of the entire time the algorithm spends on encapsulation.

In decapsulation is where the most time is spend on division. It is essential to note that the time values provided in the table account for not just one but three division operations. This is the principal reason behind the considerably greater time requirements for this operation in decapsulation when compared with the encapsulation and key generation processes.

6.3. PERFORMANCE AGAINST PREVIOUS IMPLEMENTATIONS

In order to compare the results to previous implementations also in FPGA, the parameters p=757, q=3727 and t=116 will be taken since this set is closer to the one used in previous implementations.

The results will be compared with those obtained by the streamlined NTRU Implementations SNTRUP, HS [5] and SNTRUP[13], as well as an implementation of the original

[13] MAROTZKE. “A constant time full hardware implementation of Streamlined NTRU Prime”. In: *Smart Card Research and Advanced Applications—19th International Conference, CARDIS 2020* (2020).

NTRU [14]. SNTRUP [13] and NTRU [14] were both implemented in on a Xilinx Zynq Ultrascale+ FPGA. On the other hand, SNTRUP, HS [5] was implemented on the same target device as this implementation.

Module	Design	slices	LUT	BRAM	DSP	Freq (MHz)	Time (mS)
Key Gen	SNTRUP (proposed)	6,221	48,104	-	21	200	565.5
	SNTRUP, HS	10,827	39,200	33.5	23	143	0.4477
	SNTRUP	1,068	5,935	11,5	12	271	4.748
	NTRU	10,127	50,347	6,5	45	250	0.2686
Encap	SNTRUP (proposed)	12,999	16,700	-	5	200	136.6
	SNTRUP, HS	11,218	40,879	4.5	6	144	0.0348
	SNTRUP	844	4,570	7.5	8	271	0.439
	NTRU	7,370	33,698	5.5	0	250	0.0183
Decap	SNTRUP (proposed)	15,810	33,574	-	15	200	362.9
	SNTRUP,HS	10,169	36,789	3.5	9	137	0.0802
	SNTRUP	902	5,117	7	8	271	0.958
	NTRU	7,785	38,642	2.5	45	300	0.183

Table 5. A comparison of different streamlined NTRU Prime implementations. The implementation of this paper is marked as proposed

As can be seen in table 4, the implementation proposed in this paper takes longer than previous implementations. The time differences can be attributed to 3 main operations: multiplication, modulo reduction, specifically division, and the inverses. This is because the algorithms we proposed for these operations are relatively straightforward in comparison to those used by other implementations that leverage more advanced techniques, which significantly enhance operational efficiency.

---

[14] Viet Ba DANG, Kamyar MOHAJERANI, and Kris GAJ. "High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber". In: *Cryptology ePrint Archive, Paper 2021/1508* (2021).

For instance, implementation SNTRUP, HS [5] uses a technique for multiplication involving linear shiftback register (LFSR), which enables them to circumvent the need for division during modulo reduction. This is achieved by conducting all operations within the ring or field, requiring only the modulo reduction mod q to be performed. On the other hand, our implementation does the entire modulo reduction process. A step that, as indicated by our simulation results, consumes a substantial amount of time.

Furthermore, it is worth noting that another operation where our efficiencies differ significantly is the inversion. Even though the EGCD is used in implementation SNTRUP [13] as well, they use a streamlined constant time variant of Euclid's algorithm [15], which makes it significantly more efficient.

Meanwhile, on the resource utilization of the FPGA, the proposed implementation tends to compare more favorably specially in the LUT and Block RAM usage, this last one is mainly due to the fact that our design does not use any block RAM for memory, only distributed RAM.

The presented design uses 50.4% less LUTs as logic in Key gen than implementation SNTRUP, HS [5]. Moreover, in the encapsulation phase, this design utilizes 50.4% fewer LUTs, encompassing both memories and logic, in contrast to the approach detailed in NTRU [14]. Additionally, in encapsulation, 8.73% less LUTs are used than SNTRUP, HS [5]. Meaning that, while it is not faster it compares rather favorably when it comes to the logic utilization of the FPGA.

---

[15] BERNSTEIN and YANG. "Fast constant-time gcd computation and modular inversion". In: *Cryptology ePrint Archive, Paper 2019/266, 2019.* (2019).

7. CONCLUSIONS

In this paper an architecture for an implementation of streamlined NTRU Prime was proposed. When comparing the results to previous implementations the proposed design does not compare favorably in terms of timing taking almost 12000% more than the slower previous implementation SNTRUP [13] just on key generation.

While we did take some steps to make the algorithm more efficient. Specifically we took deliberate steps in memory arrangements, especially in the division process and the Extended Greatest Common Divisor (EGCD) calculation. Additionally, we have introduced parallel execution for both inverse operations. These efforts, although beneficial, were not enough to match the results obtained by previous implementations.

On the other hand, while the implementation does lag in terms of timing, it performs relatively well on resource utilization of the FPGA specially against SNTRUP, HS [5] and NTRU [14]. This aspect can be also crucial since the resource utilization of an implementation can impact the practicality and feasibility of using this kinds of cryptographic algorithms.

Furthermore, there are aspects that could improve in the future particularly in optimizing the multiplication, avoiding the division operation in modulo reduction, which currently represents a significant time overhead in our FPGA implementation, after inversion in R/q.

BIBLIOGRAPHY

BERNSTEIN Chuengsatiansup, Lange and VREDENDAAL. “NTRU Prime: reducing attack surface at a low cost”. In: *Selected Areas in Cryptography 2017* 1.1 (2016), pp. 230–260 (cit. on pp. 12, 19, 22, 24, 26).

BERNSTEIN and YANG. “Fast constant-time gcd computation and modular inversion”. In: *Cryptology ePrint Archive, Paper 2019/266, 2019.* (2019) (cit. on p. 44).

BUCHANAN, William J. *Inverse Polynomial (mod p)*. 2023. URL: https://asecuritysite.com/pqc/inv%5C_poly (visited on 2023) (cit. on p. 26).

CHEN Peng, Marotzke. “Streamlined NTRU Prime on FPGA”. In: *Journal of Cryptographic Engineering* (2022) (cit. on pp. 21, 31, 42–45).

DANG, Viet Ba, Kamyar MOHAJERANI, and Kris GAJ. “High-Speed Hardware Architectures and FPGA Benchmarking of CRYSTALS-Kyber, NTRU, and Saber”. In: *Cryptology ePrint Archive, Paper 2021/1508* (2021) (cit. on pp. 43–45).

FORNEY, David. “chapter 7: Introduction to finite fields”. In: *Lecture notes course 6.451 (”Principals of digital communication II”) MIT* (2005) (cit. on p. 15).

HAMMOND, Jacob. “FPGA Random Number Generator”. In: *arXiv, Paper arXiv:2209.04423, 2022* (2022) (cit. on p. 27).

İLTER, M. B. and M. CENK. *Efficient Big Integer Multiplication in Cryptography*. 2017. URL: https://hdl.handle.net/11511/75405 (visited on 2023) (cit. on p. 31).

MAROTZKE. “A constant time full hardware implementation of Streamlined NTRU Prime”. In: *Smart Card Research and Advanced Applications—19th International Conference, CARDIS 2020* (2020) (cit. on pp. 42–45).

SOFTWARE, NTRU Prime. *Reference implementation in Sage*. 2022. URL: https://ntruprime.cr.yp.to/software.html (cit. on p. 26).

STANDARDS, National Institute OF and TECHNOLOGY. *Post-Quantum Cryp- tography standirization*. 2017. URL: https://csrc.nist.gov/projects/post-%20quantum-cryptography/post-quantum-cryptography-standardization (cit. on p. 12).

— *Selected Algorithms*. 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022 (cit. on p. 12).

STEIN, William. 2005 (cit. on p. 26).

STRÖMBERGSON Nordmark, Kindgren. *SHA512*. 2021. URL: https://github.com/secworks/sha512 (visited on 2021) (cit. on p. 40).

SUMMER, Jan. *Verilog Iterative division*. 2020. URL: https://github.com/jasommer/verilog_iterative_division/blob/master/division_core.v (visited on 2020) (cit. on p. 39).