Universidade do Minho
Escola de Engenharia
Departamento de Informática

Afonso Pires Fontes

Integrating post-quantum cryptography (NTRU) in the TLS protocol

December 2019

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Afonso Pires Fontes

Integrating post-quantum cryptography (NTRU) in the TLS protocol

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
José Manuel Esgalhado Valença

December 2019

STATEMENT OF INTEGRITY

ABSTRACT

We aim to integrate new “suites”, using post-quantum authentication and encryption techniques, in the TLS protocol. Namely, this project is dedicated to integrating algorithms belonging to the NTRU family of cryptossystems in the OpenSSL library and in the Python package “Cryptography”.

Even though all the algorithms included in this project have already been implemented as part of their submissions to the NIST Post-Quantum Standartization project, currently there doesn’t seem to exist a way to perform prototyping and testing of these cryptossystems in real-life use cases, and it would be interesting to create such tools.

We also aim to test if these algorithms could be further optimized for speed and efficiency by comparing the reference implementations (submited to NIST and publicly available) with our own implementations that perform some required mathematical operations in a very efficient manner (by using specialized number theory libraries).

Keywords: cryptography, post-quantum, NTRU

RESUMO

Pretende-se integrar novas “suites” no protocolo TLS que usem técnicas de autenticação e cifra na categoria de técnicas pós-quanticas. Nomeadamente, este projecto é dedicado à integração de algoritmos da família NTRU na biblioteca OPENSSL e na “package” Cryptography para o Python.

Apesar de todos os algoritmos contemplados neste projeto já terem sido implementados no âmbito da sua submissão ao NIST Post-Quantum Standartization project, actualmente não parece existir forma de testar e prototipar estes criptossistemas em casos de uso realistas, e seria interessante desenvolver ferramentas que o permitam.

Pretende-se também aferir se estes algoritmos podem ser optimizados em eficiência e velocidade de execução, comparando as implementações de referência (submetidas ao NIST e disponiveis publicamente) com as nossas implementações, que efectuam algumas operações matemáticas necessárias de forma muito eficiente (com recusro a bibliotecas de teoria de números especializadas).

Palavras-chave: criptografia, pós-quântica, NTRU

CONTENTS

1

INTRODUCTION

1.1 PROBLEM

All public key cryptossystems are based on the intractability of certain mathematical problems (so called hard problems). Namely, the public key cryptossystems used today in most applications are based in the intractability of one of three mathematical problems: Integer Factorization Problem (RSA cryptossystem, etc), Discrete Logarithm Problem over Finite Fields (Diffie-Hellman Key Exchange, DSA - Digital Signature Algorithm, etc) and Elliptic Curve Discrete Logarithm Problem (elliptic curve cryptography).

With the evolution of quantum computing, significant research effort is being put into leveraging the power of quantum computers to solve the hard problems used in today's cryptography, which will predictably impact the security of current (not post-quantum) public key cryptossystems in the not so distant future.

One of the techniques being studied to solve the hard problems used in today's public key cryptography is Shor's Algorithm. It was already demonstrated that, on a sufficiently powerful quantum computer, Shor's Algorithm can solve the three problems that act as the foundation of current public key cryptography with polynomial complexity, which is currently impossible on a classical computer (for example, the most efficient integer factorization algorithm currently known that can run entirely on a classical computer is General Number Field Sieve, which runs with sub-exponential complexity). The largest number known to have been factored using Shor's Algorithm today is 21 (factored into 3 x 7, see (2)), which is not nearly large enough to pose a security threat, but that is only due to the limitations of the current quantum computers, which will predictably be overcome in the future.

Due to all the mentioned facts, it becomes increasingly important to develop and implement new encryption and authentication techniques that are not based on the hard problems used in classical public key cryptography and are not vulnerable to quantum based attacks.

1.2 STATE OF THE ART

1.2.1 *Post-quantum hard problems and cryptossystems*

Several alternatives to the classical hard problems used in public key cryptography have been proposed. One of such alternatives is lattice-based cryptography. A lattice is a subgroup of $\mathbb{R}^n$ spanned by a set of linearly independent vectors with coefficients in $\mathbb{Z}$ (for simplicity, a lattice can be visualized as a mesh of points in n-dimensions). Lattices have proven an excellent source of problems that can be applied to post-quantum cryptography, namely, the Closest Vector Problem (CVP) (i.e. determining the vector $\mathbf{v}$ in a lattice that is closer to a given non-lattice vector), Shortest Vector Problem (SVP) (i.e. determining the shortest non-zero vector in a lattice), the Learning With Errors problem (LWE), among others.

While lattices seem to be one of the most important sources of problems suited to post-quantum cryptography, they are far from being the only one. Alternatives include code-based cryptography, based on the difficulty of decoding linear codes, supersingular elliptic curve isogeny, based on properties of supersingular isogeny graphs, among others. Unlike the classical problems, we believe public key cryptossystems based on these should not be vulnerable to quantum based attacks, i.e., having a quantum computer should not offer any advantage to an attacker in relation to only having access to classical computers.

Based on these hard problems, several alternatives to the classical public key cryptossystems have been proposed. The NTRU cryptossystem, originally consisting on two algorithms - NTRUEncrypt (a public key cipher) and NTRUSign (a signature scheme) - is one of such alternatives, and while the cryptossystem as a whole will probably not be at the forefront of post-quantum cryptography in the future (NTRUSign and its derivatives, for example, seem to have fallen out of favor with the cryptography community), NTRUEncrypt is still considered one of the most viable options for post-quantum public key encryption and key exchange, and most importantly, the mathematical structures used in the original NTRU cryptossystem served as “inspiration” to further advance the field of post-quantum cryptography and today there are several public key encryption and authentication mechanisms based on those structures. These subjects will be discussed with more detail in sections 1.2.2 and 1.2.3.

Other alternatives include NewHope (key exchange mechanism based on the Learning With Errors problem), the McEliece algorithm (a code-based public key cipher), SIKE (a key exchange protocol based on supersingular elliptic curve isogeny), etc.

1.2.2 *The NTRU cruptossystem*

The NTRU cryptossystem was first developed by the mathematician Jeffrey Hoffstein around 1995 and further developed by the mathematicians Jeffrey Hoffstein, Fill Pipher and Joseph H. Silverman in the following years. It was probably the first practical lattice-based cryptossystem to be developed. In 1996, the developers of NTRU founded the company *NTRU Cryptossystems Inc.* and were granted a patent on the cryptossystem. In 2009, *NTRU Cryptossystems Inc.* was acquired by the security company *Security Innovation*.

The NTRU cryptossystem originally consisted of two algorithms - NTRUEncrypt and NTRUSign. The original version of NTRUEncrypt algorithm is still considered secure. Some theoretical attacks have been proposed against it, but all of them proved to either be impractical or rely in bad implementations of the algorithm. For example, Kyungmi Chung, Hyang-Sook Lee and Seongan Limb proposed an attack based on lattice reduction using the Lenstra-Lenstra-Lovász (LLL) algorithm (3), but the attack is only possible if a small enough dimension parameter (N) is chosen and should not be practical against an NTRU implementation with properly chosen parameters. Another attack proposal by Nick Howgrave-Graham (see (4)) shows that even a hybrid attack combining lattice reduction with the LLL algorithm and meet-in-the-middle strategies is not feasible if the security parameters are properly chosen.

The NTRUSign algorithm however has been broken successfully and its original version is no longer considered secure. For example, Phong Q. Nguyen and Oded Regev successfully conducted an attack that recovered the NTRUSign private key from a list of as little as 400 signatures (for an NTRU dimension parameter of 251) by turning the problem into a multivariate optimization problem and solving it (see (5)). To counter these attacks, several improvements have been proposed to the original algorithm and several improved versions of it have been developed. These versions usually employ a combination of *message perturbation* techniques (slightly displacing the original message by a small random amount prior to signing it, to include some degree of randomness in the signature, see (6)) and *rejection sampling* (rejecting “bad” signatures, i.e., signatures that could potentially leak information about the private key (11)).

Being the one of the first practical cryptossystems resistant to quantum-based attacks to be developed, NTRU served as a basis for a considerable amount of research done in this field. Consequently, there are some alternative algorithms (both public key ciphers and signature schemes) based on the original NTRU that are worth mentioning, some of them being digital signature schemes that aim to correct the security vulnerabilities found in the original scheme. pqNTRUSign is one of such alternatives developed by researchers from *Security Innovation* and incorporates both *message perturbation* and *rejection sampling* in an attempt to close the security vulnerabilities of the original scheme. Another alternative based on the same techniques was proposed by Vadim Lyubashevsky and presented by Joseph H. Silverman (one of the developers of NTRU) in his talk “NTRU and Lattice-Based Crypto: Past, Present and Future” that took place in the DIMACS Workshop on The Mathematics of Post-Quantum Cryptography in January 2015 (11). Both of these alternatives are yet to be proven insecure.

Another alternative to the, now proven insecure, NTRUSign is FALCON (Fast-Fourrier Lattice-based Compact Signatures over NTRU). FALCON is a signature scheme developed by Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang, and while it is not part of the original NTRU cryptossystem, it is based on the same mathematical structures used in NTRU (the so-called NTRU Lattices). Therefore, it is being treated in this dissertation as part of the so-called NTRU family.

It is also worth mentioning *NTRU Prime*, a public key cipher developed by Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange and Christine van Vredendaal, published in 2016. This cipher is based heavily on NTRUEncrypt but it aims to strengthen some potential weaknesses in the basic algebraic structures used in NTRU by performing its operations in a slightly different polynomial ring (7).

In 2017, *Security Innovation* made all the patents pertaining NTRUEncrypt public domain, while still holding the patents to pqNTRUSign (*Security Innovation's* improved alternative to NTRUSign).

1.2.3 *Post-quantum cryptography - Implementations and standardization*

There is currently an ongoing project by NIST (National Institute of Standards and Technology) - Post-Quantum Cryptography Standardization - to standardize post-quantum cryptography, similary to other (past) NIST projects to standardize other sub-fields of cryptography (most notably, when AES was standardized by NIST in 2001). Several algorithms belonging to the *NTRU family* have been submitted, namely, a version of NTRUEncrypt and pqNTRUSign were submited by Security Inovation, while NTRU Prime and FALCON were submitted independently. At the time of writting this document, the list of candidate algorithms approved to be considered in round 2 of the competition was already published: NTRUEncypt, NTRU Prime and FALCON have been approved to be present in round 2, while pqNTRUSign wasn't.

There are currently some implementations of the NTRU cryptossystem, including the C implementation by *Security Innovation* submitted to NIST as part of the Post-Quantum Cryptography Standardization project, but none of them executes the necessary mathematical operations in the most efficient manner. For example, the implementation submitted to NIST as part of the Post-Quantum Cryptography Standardization, computes polynomial multiplication with $O(\frac{N^2}{2})$ complexity for a N degree polynomial, using a naive NTT algorithm (8), instead of using the Cooley-Turkey method, which wields the same result with $O(n\,log\,n)$ complexity and is specially advantageous for big polynomials, like the ones used in NTRU.

There is also an implementation in C and Java by Tim Buktu (9) that seems to focus on leveraging AVX2 and SSSE3 instructions whenever they are available to deliver improved performance, but these instructions are not available in most devices (for example, they are not available in current smart phones or smart cards). In addition, having part of the algorithm depend on manufacturer-specific implementations of certain instructions may raise security concerns with some users, while a full software implementation is unlikely to.

It is also important to note that the most recent version of the TLS protocol (TLS 1.3 whose last specification was published in August 2018) makes no mention to NTRU or any other post-quantum cryptography technique.

2 FUNDAMENTALS

All the public key cryptosystems currently in use are based, directly or indirectly, on the intractability of a small set of mathematical problems that, as already stated, cannot be considered secure in a world where quantum computers are a viable tool to attackers. In this chapter, we will present some basic concepts about lattices, lattice-based hard problems suitable to be used in cryptography and how can lattices be used to design public key cryptosystems that are secure in a quantum world.

2.1 LATTICES

In a given vector space V, any vector belonging to V can be added to another vector in V or multiplied by a real number (with the resulting vector still being a member of V). A lattice is a group similar to a vector space, except a vector in a lattice can only be multiplied by an integer (instead of any real) number. The resulting group can be visualized as a mesh of points in n dimensions (1).

A formal definition of a lattice can be stated as (according to (1)):

Definition 1. *Let $v_1, ..., v_n \in \mathbb{R}^m$ be a set of linearly independent vectors. The lattice L generated by $v_1, ..., v_n$ is the set of linear combinations of $v_1, ..., v_n$ with coefficients in $\mathbb{Z}$.*

$$L = a_1v_1 + ... + a_nv_n : a_1, ..., a_n \in \mathbb{Z}.$$

A basis for L is any set of independent vectors that generates L. Any two such sets have the same number of elements. The dimension of L is the number of vectors in a basis for L.

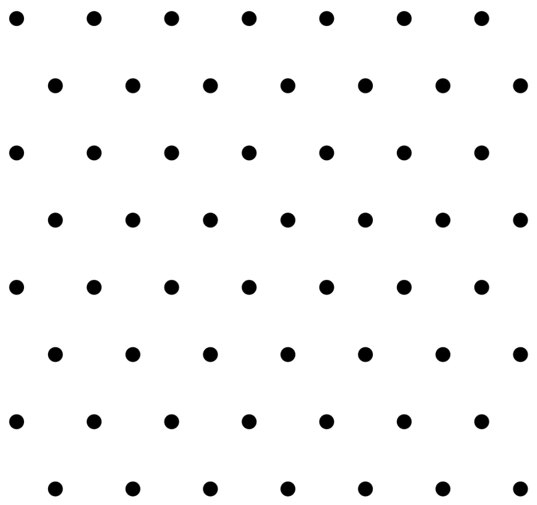
Figure 1.: Visual representation of a lattice in 2 dimensions

Lattices have several uses in both pure and applied mathematics. In cryptography, lattices have several uses. Namely, they are one of the main sources of hard problems in the design of post-quantum public key cryptosystems (for example, all the public key cryptosystems based on either NTRU or the Learning with Errors problem are, directly or indirectly, based on lattice problems). In addition to being used as a source of hard problems in the design of new cryptosystems, lattices have multiple other uses in cryptography. For example, the Lenstra–Lenstra–Lovász (LLL) lattice basis reduction algorithm has been successfully used in the cryptanalisys of some cryptossystems.

2.2 HARD PROBLEMS IN LATTICES

In order to design lattice-based public key cryptosystems, it's required to establish appropriate trapdoor functions (so-called hard problems) based on lattices and their properties. This section aims to briefly describe some of those hard problems, namely, the ones that are most relevant to the cryptosystems being studied in this dissertation.

2.2.1 *Closest Vector Problem and Shortest Vector Problem*

The *Closest Vector Problem* and the *Shortest Vector Problem* are probably the two most fundamental problems involving lattices. They can be defined as (1):

Definition 2. ***Closest Vector Problem (CVP):*** *Given a vector* $\mathbf{w} \in \mathbb{R}^m$ *that is not in the lattice* $\mathbf{L}$*, find a nonzero vector* $\mathbf{v} \in \mathbf{L}$ *that is closest to* $\mathbf{w}$*, i.e., find a vector* $\mathbf{v} \in \mathbf{L}$ *that* *minimizes the euclidean norm* $\|\mathbf{w} - \mathbf{v}\|$.

Definition 3. ***Shortest Vector Problem (SVP):*** *Find a shortest nonzero vector in a lattice* $\boldsymbol{L}$*, i.e., find a nonzero vector* $\mathbf{v} \in \mathbf{L}$ *that minimizes the euclidean norm* $\|\mathbf{v}\|$.

Note that both the CVP and the SVP may have more than one valid solution. This is why both ask for “a” shortest/closest vector.

Both the CVP and the SVP are relatively easy to solve given a so-called *good basis* for the lattice **L**, i.e., a basis consisting of relatively short vectors highly orthogonal amongst themselves, while being extremely hard to solve with a so-called *bad basis*, even if both bases generate the exact same lattice.

2.2.2 *Short Integer Solution Problem*

Definition

Definition 4. *Given a matrix* $\boldsymbol{A}$*, find a nonzero vector* $\boldsymbol{x}$ *with coefficients in* $\mathbb{Z}$ *that satisfies the conditions* $\boldsymbol{xA} = 0 \, mod \, q$ *and* $|\boldsymbol{x}| < \beta$ *for a given short* β.

While at first this problem doesn’t seem to be lattice related, Miklós Ajtai proved that the Short Integer Solution (SIS) problem is secure for the average case if the Shortest Vector Problem is considered secure for the worts case scenario (14).

3

NTRU AND NTRU-BASED CRYPTOSSYSTEMS

3.1 BACKGROUND

The basic algebraic operations employed by the original version of the NTRU cryptossystem can simply be described as polynomial multiplications in the following quotient rings:

$$\frac{\mathbb{Z}[X]}{X^N - 1} \qquad \frac{\mathbb{Z}/\mathbf{q}\mathbb{Z}[X]}{X^N - 1} \qquad \frac{\mathbb{Z}/\mathbf{p}\mathbb{Z}[X]}{X^N - 1}$$

For proof of security however its useful to see the underlying problems of NTRU as lattice problems, not as polynomial ones. Both the original NTRU paper (10) and Joseph H. Silverman's January 2015 DIMACS talk (11) describe how the basic NTRU problems can be formulated as lattice-based problems in the so-called NTRU lattice, i.e. the 2N dimension lattice spanned by the rows of the matrix L_h.

$$L_h = \begin{pmatrix} I & h \\ 0 & qI \end{pmatrix}$$

Where **I** is the dimension N identity matrix, **h** is the matrix consisting on N permutations of the public key, **0** is the N dimension square null matrix and **qI** is the dimension N identity matrix multiplied by q.

For example, for NTRUEncrypt, it was proven that recovering the private key knowing only the public key is equivalent to solving the Shortest Vector Problem in the NTRU lattice and recovering a plaintext message from the cyphertext and the public key is equivalent to solving the Closest Vector Problem. (1), (11).

This particular form of lattices has proven to be a great source of hard problems employed in post-quantum cryptography, and the algorithms belonging to the so-called NTRU family all have in comom the usage of such lattices (or closely related ones) in their underlying mathematical operations.

The sections that follow (2.2, 2.3 and 2.4) will present a more detailed and individualized description of the algorithms studied for this dissertation. Namely, NTRUEncrypt and NTRUPrime were the PKE/KEM algorithms choosen and FALCON was the only signature scheme studied. NTRUSign (and all its subsequent versions) was not selected because, in an attempt to keep this dissertation as relevant as possible, only algorithms selected to be part of NIST round 2 submissions were choosen.

3.2 NTRUENCRYPT

3.2.1 *The algorithm*

- Public Parameters:
  - **N** - Prime modulus (250 < N < 2500)
  - **q** - Big modulus (250 < q < 2500)
  - **p** - Small modulus (for example, p = 3) relatively prime to **q**
- Private Key:
  - **F, G** - Random $\in \{-1, 0, 1\}^N$
  - **f** = 1 + pF
  - **g** = pG
  - (pF, pG = F or G respectively, with each coefficient multiplied by p)
- Public Key:
  - **h** $\equiv f^{-1} * g \, (mod \, q)$
  - (f^{-1} = the inverse of polynomial f in the $X^N - 1$ polynomial ring)
- Encrypt:
  1. **m** - Clear text $\in \{-1, 0, 1\}^N$
  2. **r** - Random $\in \{-1, 0, 1\}^N$
  3. **e** $\equiv r * h + m \, (mod \, q)$
- Decrypt:
  1. **a** $\equiv f * e \, (mod \, q)$
  2. Lift **a** to $\mathbb{Z}^N$ with coefficients $|a_i| \leq \frac{1}{2}q$, that is, a_i becomes negative if a_i mod q > q/2
  3. **a mod p** equals **m**

3.2.2 *Considerations*

It is easy to see the algorithm works because:

$$\begin{aligned} a &\equiv f * e \,(mod\, q) && (1)\\ &\equiv f * (r * h + m)\,(mod\, q) && (2)\\ &\equiv f * (r * f^{-1} * g + m)\,(mod\, q) && (3)\\ &\equiv r * p + f * m\,(mod\, q) && (4) \end{aligned}$$

Since **r**, **p**, **f** and **m** all have small coefficients ($< \frac{q}{2}$), lifting **a** to $\mathbb{Z}^N$ yields an exact equality:

$$a = r * p + f * m \qquad (5)$$

And then, reducing modulo p returns the original message **m**:

$$\begin{aligned} a &\equiv r * g + f * m\,(mod\, p) && (6)\\ &\equiv r * pG + (1 + pF) * m\,(mod\, p) && (7)\\ &\equiv m\,(mod\, p) && (8) \end{aligned}$$

Notice that in (7), $(r * pG)$ and pF are both $\equiv\, 0\,(mod\, p)$.

3.3 NTRU PRIME

3.3.1 *The algorithm*

NTRU Prime is an algorithm of the NTRU family and one of the public key ciphers present in round 2 of NIST Post Quantum Standardization project.

The NTRU Prime submission to NIST contains two slightly different Key Exchange Mechanisms (KEM) based on NTRU Prime, named *Streamlined NTRU Prime* and *NTRU L Prime*. Both KEMs are based on the same algorithm and both should offer equivalent security guarantees, therefore, *NTRU L Prime* was selected to be studied in this project since, according to the performance metrics published by the NTRU Prime developers and to our preliminary testing, it should perform somewhat better for our use-case (TLS).

- Public Parameters:
  - $(p, q, \omega, \delta, I)$ such as:
    - p and q are prime numbers

    - ω, $\delta \, and \, I$ are positive integers
    - $2p \geq 3\omega$
    - I is a multiple of 8
    - $p \geq I$
    - $q \geq 16\omega + 2\delta + 3$
    - $x^p - x - 1$ is irreducible in the polynomial ring $\mathbb{R}$ / q
  - For NTRU L Prime, the following parameters are also required:
    - $\boldsymbol{\tau}$, a positive integer.
    - A deterministic function **Top**: $\mathbb{Z}/q \to \mathbb{Z}/\tau$
    - A deterministic function **Right**: $\mathbb{Z}/\tau \to \mathbb{Z}/q$ such as: Right(Top(C)) - C $\in$ Z/ q is in {0,1,...,δ} for any C in $\mathbb{Z}/q$.

- Key pair:
  - **G** - Random $\in \mathbb{R}$ / q
  - **a** - random short vector
  - **Public key** = (G, A = Round(aG))
  - **Private key** = a
- Encrypt:
  1. **r** = $(r_0, r_1, ..., r_{n-1})$ = message
  2. **(G, A)** = Public key
  3. Compute bG $\in \mathbb{R}$ / q, where **b** is a short random vector
  4. Compute the bottom I coefficients of bA $\in \mathbb{R}$ (coefficients $(bA)_0, (bA)_1, ..., (bA)_{I-1}$)
  5. Compute **T** = $(T_0, T_1, ..., T_{I-1}) \in (\mathbb{R}/\tau)^I$,such as: $T_j = (Top(bA_j) + r_j(q-1)/2)$
  6. **Ciphertext** = (B = Round(bG), T)
- Decrypt:
  1. Compute the bottom I coefficients of aB in R/ q
  2. Compute the clear text message **r** such as: View $Right(T_j) - (aB)_j + 4\omega + 1 \in$ Z/ q as an integer between -(q-1)/2 and (q-1)/2 and define the bit r_j as the integers sign bit (1 if negative, 0 if positive).

3.3.2 *Consideration*

As can be seen, NTRU Prime is a relatively simple algorithm based on the same high-level mathematical principles as the original NTRU. However, NTRU Prime contains some differences, including a slightly different polynomial ring ($x^p - x - 1$ instead of $x^p - 1$ which is the ring used in the original NTRU). These differences aim to protect the cryptossystem agains some known attacks in relation to the original NTRU as well as prevent some difficulties with the original NTRU like decryption failures (13).

3.4 FALCON

3.4.1 *The algorithm*

FALCON stands for *Fast-Fourrier Lattice-based Compact Signatures over NTRU* and it's a signature scheme whose underlying mathematical operations are performed over NTRU lattices. It is worth mentioning that the authors of FALCON define reducing communication complexity (as a function of the public key size + signature size) as a main goal of the algorithm (more so than speed). In fact, the *C* in FALCON stands for *Compact* and the algorithm is optimized to reduce, as much as possible, the size of the signature and public key (the size of the private key is not taken into account because, evidently, in a typical use case, the private key is not sent over any communication channel and therefore does not affect the communication complexity).

A simplified version of the algorithm could be presented as:

- Public Parameters:
  - The polynomial $\phi = x^N + 1$, for a given security parameter N.
  - An integer modulo **q**
  - A real β
- Private Key:
  - The basis of a lattice, defined by the matrix $B = \begin{pmatrix} g & -f \\ G & -F \end{pmatrix}$, with f, g, F and G choosen to satisfy the equation $fG - gF = 0 \, mod \, \phi$
  - The matrix $\widehat{B} = \begin{pmatrix} FFT(g) & -FFT(f) \\ FFT(G) & -FFT(F) \end{pmatrix}$, with FFT refering to the Fast Fourrier Transform.
  - The so-called FALCON tree T - a normalized LDL decomposition of $\widehat{B} \times \widehat{B}^*$
- Public Key:
  - $\mathbf{h} \equiv g \times f^{-1} \, mod \, q$

- Sign:
  1. **c** = HashToPoint(r||m), with r = random bits and m = message
  2. **t** = $(FFT(c), FFT(0)).\widehat{B}^{-1}$
  3. **e** $\equiv r * h + m \, (mod \, q)$
  4. **do**
  5. **z** = ffSampling(t, T)
  6. **s** = $(t - z).\widehat{B}$
  7. **while** $\|s\| > \beta$
  8. (s_1, s_2) = invFFT(s)
  9. **s** = Compress(s_2)
  10. **sig** = (r, s)
- Verify:
  1. **c** = HashToPoint(r||m,q,n)
  2. s_2 = Decompress(s)
  3. $s_1 = c - s_2 h \, mod \, q$
  4. Accept if and only if $\|(s_1, s_2)\| \leq \beta$

3.4.2 *Considerations*

The FALCON algorithm can be seen as relatively simple when seen at a high abstraction level, although it is considerably more complex when all the operations performed are analyzed in detail.

The most significant operations performed could be described as:

- **HashToPoint** - To hash a given bit stream into a polynomial in $\mathbb{Z}_q[x]/\phi$.
- **FFT** - Fast Fourrier Transform.
- **ffSampling** - A function that takes a matrix A, a trapdoor function T and a target c and outputs a vector z such as, zA = c mod q. FALCON uses the Ducas and Prest sampler, co-developed by Thomas Prest, one of the inventors of FALCON.
- **Compress/Decompress** - Functions that aim to reduce the communication complexity by optimizing the encoding of the signature for compactness. Note that Decompress(Compress(s)) = s.

Also note that FALCON's security is based on the *Short Integer Solution Problem*, (described in section 2.2.2) given by the **ffSampling** function.

4

IMPLEMENTATION AND INTEGRATION

4.1 INTEGRATION ON OPENSSL

The integration of the algorithms on OpenSSL was done by integrating them first in Open Quantum Safe's LibOQS and then integrating LibOQS on the Open Quantum Safe's branch of OpenSSL.

LibOQS is an open-source C post-quantum cryptography library that allows us to integrate and test new key exchange mechanisms and signature schemes with the TLS protocol. The version of LibOQS used in this project already had *Security Innovation's* NTRU implementation properly integrated, so that implementation was used. NTRU Prime on the other hand was not only not integrated on LibOQS but it was not possible to find any work on how to integrate it into LibOQS so a clean integration had to be done. Falcon was also integrated, with the integration work being based on previous (publicly available on the repositories) work from the Open Quantum Safe's team.

With all the desired algorithms being properly integrated, LibOQS can then be built into a static library that can easily be linked when building Open Quantum Safe's OpenSSL branch. This process allowed us to achieve a fully functional OpenSSL build with the desired algorithms being available as key exchange mechanisms or signature schemes, without having to change OpenSSL's code which is extremely complex and would probably not be a realistic task for one person in the time available to complete this dissertation.

The OpenSSL branch being used directly supports using NTRU (which was already integrated into LibOQS) as a KEM. It does not however directly support using NTRU Prime as a KEM or Falcon as a signature scheme. Fortunately, the version of LibOQS being used allows us to set a default KEM and Signature Scheme that can be used in OpenSSL even without OpenSSL specifically supporting it. NTRU Prime and Falcon were then set as the default KEM and signature scheme respectively, which allowed us to use them in OpenSSL alongside NTRU.

A more detailed guide on how to integrate an algorithm into LibOQS can be found in appendix A.

4.2 INTEGRATION ON PYTHON PACKAGE CRYPTOGRAPHY

The three algorithms under test were also integrated into the Python package Cryptography. Cryptography is a python package that, amongst other uses, exposes cryptographic primitives to the user (Python programmer) to be used directly (i.e, it allows users to directly encrypt or sign messages without having to directly use the standards and protocols commonly used in the industry, like the TLS protocol). The aim is to integrate the algorithms under study allowing them to be used directly by a relatively low level programming interface.

The algorithms were integrated on Cryptography by creating three new classes (one for each algorithm) in Cryptography's *hazmat* layer (the layer where all the wrappers to all cryptographic primitives are located).

The created classes do not directly contain any logic pertaining to the algorithms under study. Instead, they simply call Python bindings to the same algorithms being used in LibOQS allowing the programmer to work with the exact same implementations used in LibOQS/OpenSSL.

More detailed information about the integration on Cryptography can be found in appendix B.

4.3 NTL-BASED NTRU IMPLEMENTATION

In addition to using the reference implementations of the studied algorithms, a C++ implementation of the NTRU (NTRUEncrypt) algorithm based mostly on the NTL library was also implemented.

NTL is a C++ number theory library developed by the mathematician Vitor Shoup. In the context of this project and the NTRU algorithm, the NTL library is being used to perform the required mathematical operations in the most efficient manner possible. For example, for polynomial multiplications, by far most significant operation performance wise performed by the NTRU algorithm and the one we have a bigger interest in optimizing, the NTL library heuristically chooses the multiplication method to used from four different options (classical algorithm, Karatsuba method, FFT using small primes and FFT using the Schoenhagge-Strassen approach) based on the coefficient domain.

Using the NTL library allows us not only to perform all the required mathematical operations in a very efficient manner, but it also accelerates and facilitates the coding process by abstracting some mathematical structures and operations. Namely, it allows us to instantiate a class (ZZ_p) representing integer numbers modulo p and a class (ZZ_pX) representing polynomials with coefficients in ZZ_p. The modular inverse of a ZZ_pX polynomial is also a good example of a seemingly complex mathematical operation performed at a high abstraction level by the NTL library. In addition to the potential gains in speed of execution, this approach also greatly enhances code maintainability and readability.

Unfortunately, the usage of external libraries and the fact that the code had to be written in C++ (rather than “pure” C) makes it impossible (or at least, extremely complex and time consuming) to integrate this implementation in OpenSSL or LibOQS. So instead of putting the implementation through the standard battery of tests (explained in the next chapter), this implementation was subject to a different testing scheme and compared to the reference implementation of NTRU (NTRUEncrypt).

This implementation was based on the algorithm presented by Joseph H. Silverman (one of the inventors of the NTRU cryptossystem) (11) in 2015.

5

TESTS

5.1 TLS TESTS

5.1.1 *Description*

In order to assess the performance of the studied algorithms, a test was performed to measure their speed while establishing TLS connections. Signature schemes were also tested for speed in generating certificates and size of the generated certificates. A list of the algorithms used in the tests can be seen in table 1 (for ECDH and ECDSA, the name of the elliptic curve used is mentioned instead of the algorithm).

To properly compare the algorithms, their estimated bit security and the security level achieved on the NIST cybersecurity framework (levels 1, 3 and 5 indicate a bit security of approximately 128, 192 and 256 bits respectively) were used. Since it is not always trivial to compute the bit security of algorithms, and for post-quantum algorithms in particular, there are several conflicting proposals describing how to perform the computation, we opted to omit the estimated bit security value for post-quantum algorithms. Instead, since the rules of the NIST Post-quantum Standartization Project state that submitters must submit versions of the their algorithm satisfying security levels 1, 3 and 5, we assume that all the submissions were correctly made and compare their performance against equivalent pre-quantum algorithms.

In regard to Falcon, the original submission to NIST included three possible values for the security parameter: 512, 768 and 1024, satisfying security levels 1, 3 and 5 respectively. However, on August 2nd 2019 the authors of the algorithm released an updated C implementation of Falcon not allowing the option to set the security parameter at 768, leaving only the 512 and 1024 options. The documentation accompanying the new implementation does not offer any explanation for this decision, but since it was impossible to establish if the security parameter of 512 satisfies the security level 3 in the NIST cybersecurity framework, the security parameter of 1024 was used in all the tests pertaining level 3 security (as well as level 5).

Also, in order to perform tests at the highest security level, we used RSA with a modulus of 8192 bits. This achieves a security level somewhat higher than level 3, which requires a 7680 bit modulus, but doesn't reach the value required for level 5/256 bit security which is 15360 bit modulus. This was done because 15360 bit RSA key pairs proved extremely slow to generate. This not only would significantly complicate our tests, but also make the algorithm virtually unusable at establishing TLS connections at such high security levels.

Algorithm/Curve	Bit Security (est.)	NIST level
RSA 3072	128	1
Curve Prime256v1	128	1
NTRU Prime 653	NA	1
NTRU 509	NA	1
Falcon 512	NA	1
RSA 7680	192	3
Curve Secp384r1	192	3
NTRU Prime 761	NA	3
NTRU 677	NA	3
Falcon 1024	NA	5
RSA 8192	192+	3+
Curve Secp521r1	256	5
NTRU Prime 857	NA	5
NTRU 821	NA	5

Table 1.: Bit security (est.) and NIST Security level achieved by the algorithms under test

All the tests were performed by establishing TLS connections using the OpenSSL command line interface. A Linux shell script was used to establish 1000 TLS connections via command line and output the time required to perform each connection (with a millisecond precision). The average value (time) and standard deviation where computed for all the algorithms. The size (in bytes) of a certificate signed with each of the signature schemes under test was also measured and all the results are presented in the following three sections of the document.

All the tests were performed on a i7-6700HQ CPU and a fully updated Linux Mint 19.1 installation. Other than essential system processes, nothing else was running in the test machine during the tests.

5.1.2 *Results - Level 1*

The results pertaining to the Security Level 1 can be seen in figures 2 and 3 and table 2. For this security level, RSA was used with a 3072 bit modulus, ECDH and ECDSA used Curve Prime256v1, and NTRU Prime, NTRU and FALCON used security parameters of 653, 509 and 512 respectively.

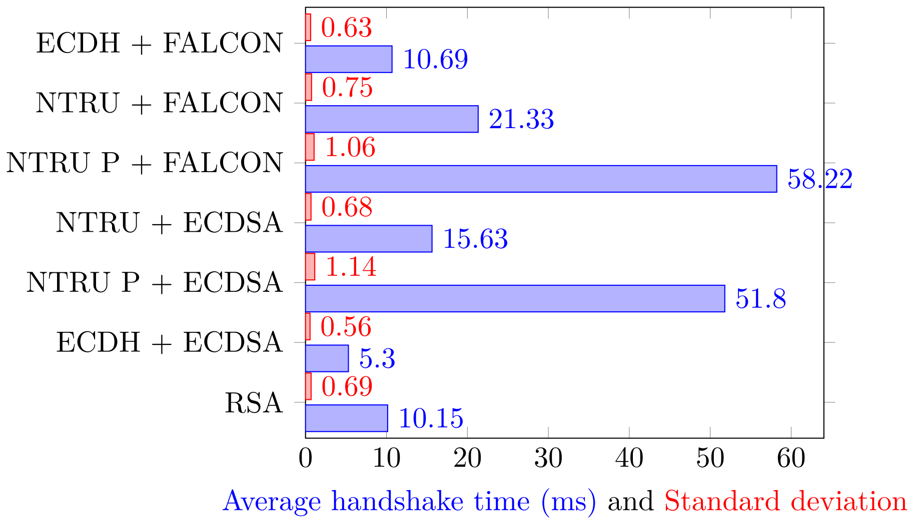

Figure 2.: Level 1 results (TLS handshake)

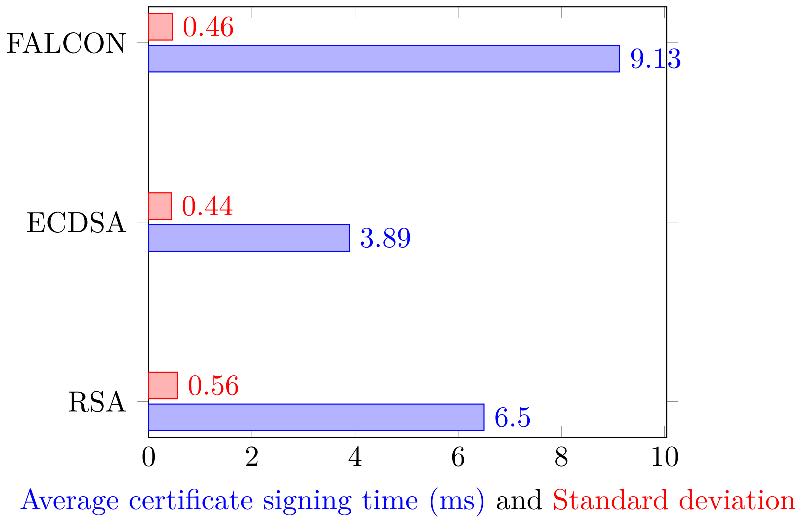

Figure 3.: Level 1 results (certificate signing)

Algorithm/Curve	Size (bytes)
RSA 3072	1505
Curve Prime256v1	570
Falcon 512	2484

Table 2.: Certificate size for the three signature schemes used

5.1.3 *Results - Level 3*

The results pertaining to the Security Level 3 can be seen in figure 4 and 5 and table 3. For this security level, RSA was used with a 7680 bit modulus, ECDH and ECDSA used Curve Secp384r1, and NTRU Prime, NTRU and FALCON used security parameters of 761, 677 and 1024 respectively.

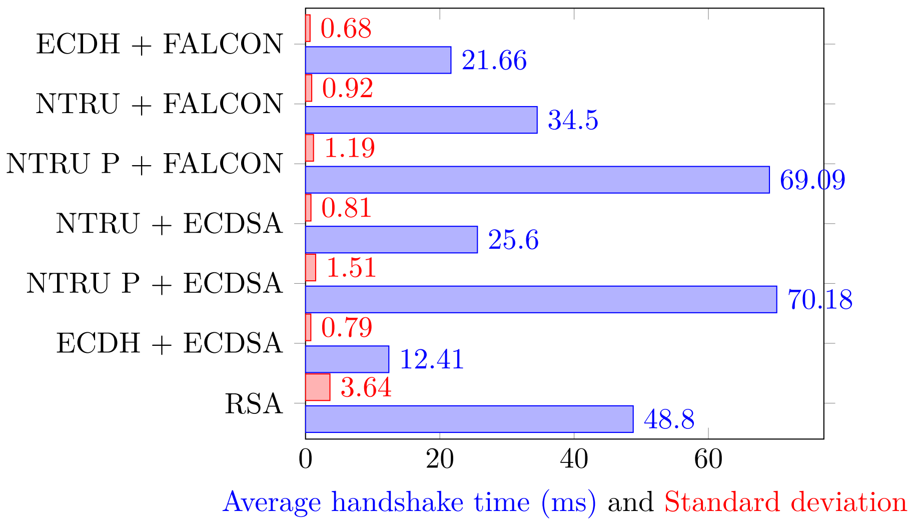

Figure 4.: Level 3 results (TLS handshake)

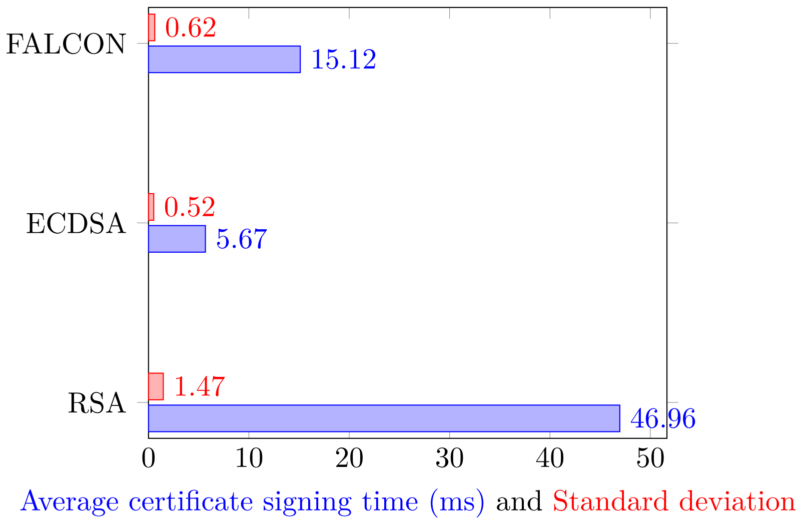

Figure 5.: Level 3 results (certificate signing)

Algorithm/Curve	Size (bytes)
RSA 7680	3024
Curve Secp384r1	656
Falcon 1024	4519

Table 3.: Certificate size for the three signature schemes used

5.1.4 *Results - Level 5*

The results pertaining to the Security Level 5 can be seen in figure 6 and 7 and table 4. For this security level, RSA was used with a 8192 bit modulus, ECDH and ECDSA used Curve Secp521r1, and NTRU Prime, NTRU and FALCON used security parameters of 857, 821 and 1024 respectively.

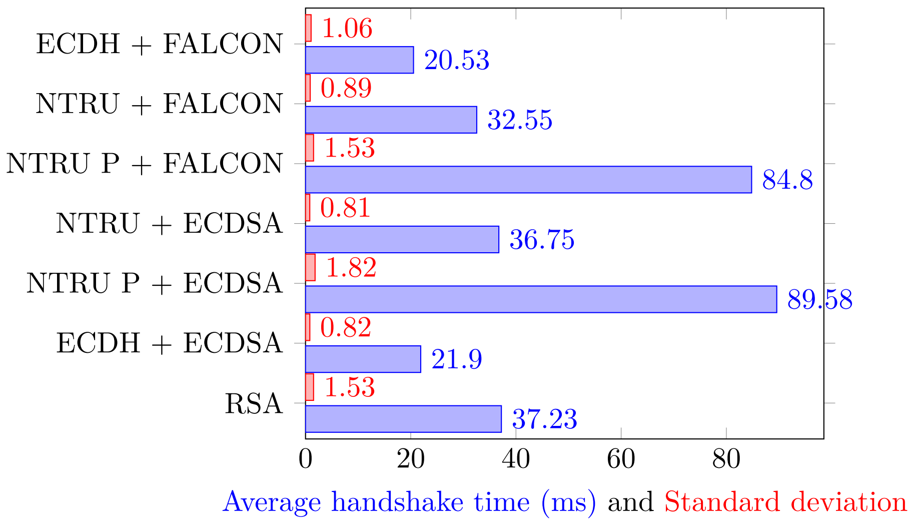

Figure 6.: Level 5 results (TLS handshake

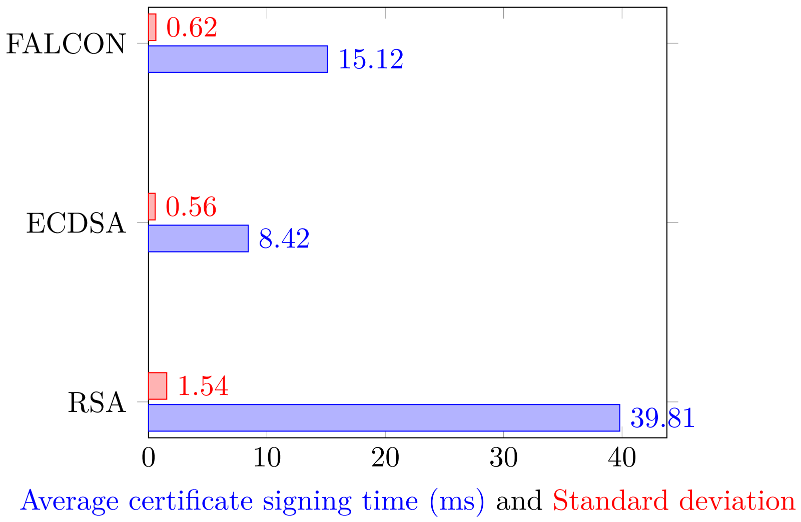

Figure 7.: Level 1 results (certificate signing)

Algorithm/Curve	Size (bytes)
RSA 8192	3195
Curve Secp521r1	757
Falcon 1024	4519

Table 4.: Certificate size for the three signature schemes used

5.2 NTL IMPLEMENTATION TESTS

5.2.1 *Description*

Since it proved impossible to integrate a C++ implementation with several external dependencies in OpenSSL, a different testing scheme had to be used to test the NTL-based implementation of NTRU. Instead of measuring the speed of the implementation in the context of TLS sessions, the implementation was tested by computing the key exchanges locally.

Three tests were performed, corresponding to the three security levels also used in the TLS tests. In the three tests, the NTRU class was instantiated with the exact same security parameter (N) as the reference NTRU implementation and 1000 key exchanges were locally computed for each security level. For comparisson, the exact same test was also run for the reference NTRU implementation (the same one we used with LibOQS) and the results can be seen in the following sections.

5.2.2 *Results - Level 1*

For level 5 security, the security parameters are:

- N = 509
- q = 2053
- p = 3

The results can be observed in figure 8.

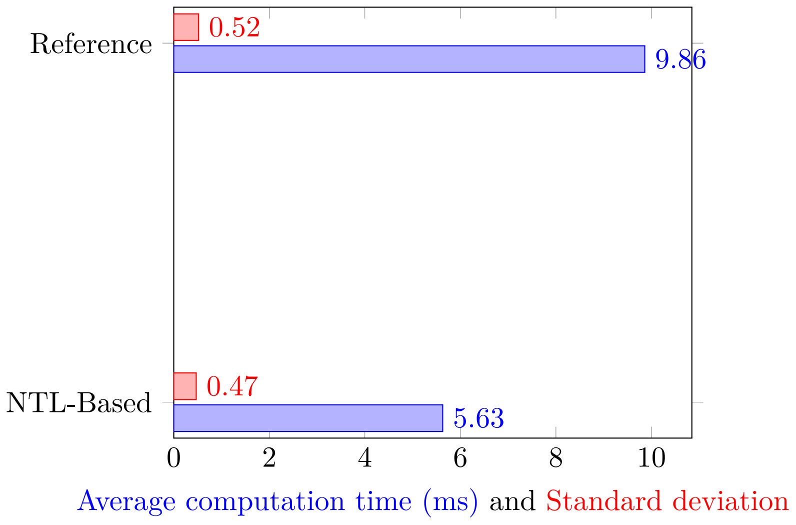

Figure 8.: Level 1 results (KEM computation)

5.2.3 *Results - Level 3*

For level 5 security, the security parameters are:

- N = 677
- q = 2053
- p = 3

The results can be observed in figure 9.

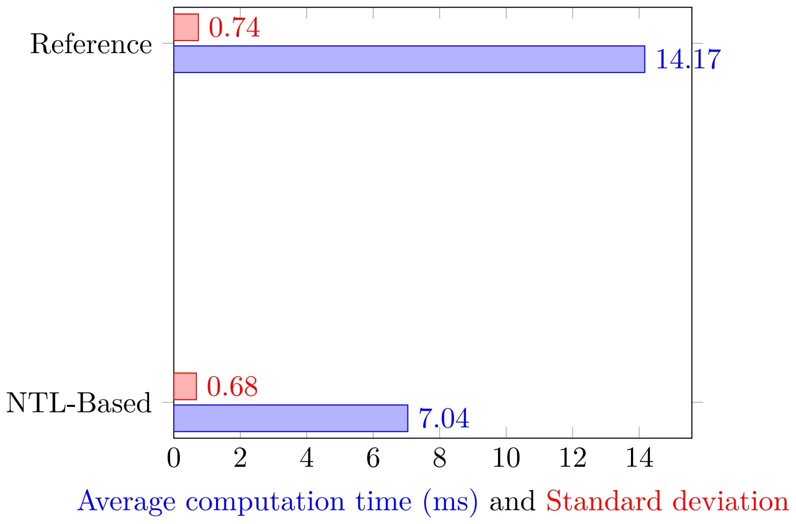

Figure 9.: Level 3 results (KEM computation)

5.2.4 *Results - Level 5*

For level 5 security, the security parameters are:

- N = 821
- q = 4093
- p = 3

The results can be observed in figure 10.

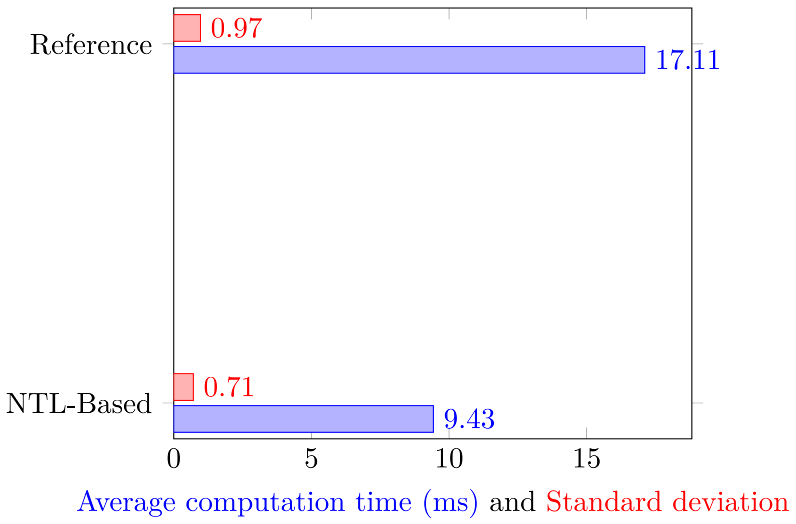

Figure 10.: Level 5 results (KEM computation)

5.3 RESULTS INTERPRETATION

From the results obtained, it is evident that NTRU Prime is the main outlier, performing significantly worse than any other key exchange mechanism. However, it is important to note that the implementation of NTRU Prime used does not, according to the authors, "sacrifice clarity for speed" (this observation can be found in the readme file that accompanies the implementation), which could lead us to the possibility of NTRU Prime performing significantly better than it did if a more speed-oriented implementation was provided.

NTRU and Falcon on the other hand performed very positively, even being faster than their pre-quantum counterparts in some security levels, which was not expected, specially considering that Falcon showed that, in some situations, it can perform faster than ECDSA working over curves with a adequate security level. It is interesting to note however, that despite Falcon being, according to its authors, an algorithm designed primarily with compactness and low communication complexity in mind, it is the signature scheme with the biggest certificate size. This could indicate that post-quantum signature schemes do not compete well against their pre-quantum counterparts in terms of communication complexity, but further tests with different algorithms would be required to assess that conclusion.

In regard to the optimization level of the implementations themselves, or more concretely, to the optimization level of the NTRU implementation, our local tests with the NTL-based implementation seem to indicate that there are still room for improvement. Our tests showed that the NTL-based implementation performed significantly better than the reference NTRU implementation, and while it is not possible to assure that the two implementations are completely identical in terms of functionality, the NTL-based implementation does performed all the required mathematical operations which should take most of the required computing power to run the algorithm. The test results seem to indicate that is still possible to further optimize the reference NTRU implementation to make it even more competitive with other classical or post-quantum techniques, and it would be interesting to assess if the results also hold true for other post-quantum algorithms.

Despite not being in the scope of this project, it was also noted that RSA performs better with a 8192 bit modulus than it does with a 7680 bit modulus (in both tests). This behaviour was not expected but proved consistent across multiple tests so it was included in the results.

6 CONCLUSIONS

This project aimed to assess the viability of post-quantum cryptography techniques in a real world use case - the TLS protocol. Currently, RSA, the algorithm used extensively both as a key exchange mechanism and signature scheme is starting to not offer enough security, at least with realistic key sizes, and that trend is expected to continue.

Consequently, key exchange mechanisms and signature schemes based on elliptic curves are expected to take over and become the *de facto* industry standard in the following years. However, it was already been demonstrated that the underlying mathematical problems that elliptic curve cryptography is based on are also vulnerable to attacks based on quantum computers, so it becomes increasingly important to study and test post-quantum cryptography techniques, specially in comparison with current elliptic curve cryptography, since this techniques are expected to become the industry standard in the not so distant future.

Our tests showed that, generally speaking, post-quantum cryptography algorithms can be very competitive with their classical counterparts, at least in terms of speed. The algorithms we tested performed well at performing both TLS handshakes and signing certificates, usually performing only slightly worse (although in some cases, they even performed slightly better) than the elliptic curve algorithms they were compared to.

NTRU Prime was the exception, having performed significantly worse than any other algorithm, but since the implementation tested did not favor speed, and since it was compared to OpenSSL implementations that were probably much better optimized, it is probably not appropriate to draw any conclusions only from this tests, specially because NTRU Prime is based on mathematical structures similar to the ones used in NTRU, so it is not expected that the algorithm would have such a big performance difference, when running under equal circumstances.

As far as memory and communication complexity are concerned, the post-quantum algorithms tested are still significantly behind their classical counterparts. Both certificate and key sizes are significantly larger, specially if compared with the certificate and key sizes of elliptic curve algorithms, although the sizes are within the same order of magnitude of the ones obtained with pre-quantum techniques and should be within “acceptable” levels for most use cases.

Generally speaking, the post-quantum algorithms tested could be deployed and utilized in the industry, assuming the security claims made by their authors hold true. Their speed is already competitive with pre-quantum algorithms and should not pose a problem, specially considering that the current implementations were still not properly scrutinised and it is probably still possible to further optimize them which would lead to even better performances than we saw in our tests. In fact, our tests with the C++ NTL-based implementation seem to confirm this hypothesis and is very likely that the studied post-quantum algorithms will be further optimized and will show better performance by the time their use becomes widespread.

The results in terms of size and memory complexity were not as encouraging but should still qualify the post-quantum algorithms studied for most use cases. Namely, the key and signature size differences should be negligible when performing encryption (key exchange) and authentication on most personal computing devices (laptops, smartphones, etc). However, the extra memory complexity that comes with using post-quantum techniques can be problematic for some use cases like smart cards or embedded devices where memory is more scarce (or not so easily expandable) and has to be more carefully managed.

In conclusion, the algorithms studied are generally suitable for real world use, even if further refinements and optimizations could make them more competitive or suitable for a wider range of use cases. However, it is important to note that evaluating the security claims made about each algorithm was not within the scope of this work and all those security claims were accepted as true.

BIBLIOGRAPHY

[1] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman. An Introduction to Mathematical Cryptography. 2008

[2] Enrique Martín-López, Anthony Laing, Thomas Lawson, Roberto Alvarez, Xiao-Qi Zhou and Jeremy L. O'Brien. Experimental realization of Shor's quantum factoring algorithm using qubit recycling. Nature Photonics 6. 2012. https://www.nature.com/articles/nphoton.2012.259

[3] Kyungmi Chung, Hyang-Sook Lee and Seongan Limb. An efficient lattice reduction using reuse technique blockwisely on NTRU. Discrete Applied Mathematics. 2016. https://www.sciencedirect.com/science/article/pii/S0166218X16302542

[4] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. CRYPTO 2007: Advances in Cryptology. 2007. https://link.springer.com/chapter/10.1007/978-3-540-74143-5_9

[5] Phong Q. Nguyen and Oded Regev. Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures. Journal of Cryptology. 2009. https://link.springer.com/article/10.1007/s00145-008-9031-0

[6] Yupu Hu, Baocang Wang and Wencai He. NTRUSign with a new perturbation. IEEE Transactions on Information Theory. 2008. https://www.researchgate.net/publication/3086927_NTRUSign_with_a_new_perturbation

[7] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime:reducing attack surface at low cost. 2017. https://ntruprime.cr.yp.to/ntruprime-20170816.pdf

[8] NIST PQ Submission: NTRUEncrypt. A lattice based encryption algorithm. (Documentation of NTRUEncrypt NIST submission.) https://github.com/NTRUOpenSourceProject/ntruencrypt_nist_submission/blob/b2fba3a0d365d08210e5f300eb7ec99a013c313a/document/NTRUEncrypt.pdf

[9] libntru (A C NTRU implementation). https://github.com/tbuktu/libntru

[10] Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. Algorithmic Number Theory. 1998. https://link.springer.com/chapter/10.1007/BFb0054868

[11] Joseph H. Silverman. NTRU and Lattice-Based Crypto : Past, Present, and Future. Presentation at “The Mathematics of Post-Quantum Cryptography” at DIMACS Center. 2015. `http://archive.dimacs.rutgers.edu/Workshops/Post-Quantum/Slides/Silverman.pdf`

[12] Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. Choosing Parameters for NTRUEncrypt. `https://eprint.iacr.org/2015/708.pdf`

[13] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime. 2018.

[14] M. Ajtai. Generating Hard Instances of Lattice Problems. Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing. 1996

[15] A Tour of NTL: NTL Implementation and Portability, https://www.shoup.net/ntl/doc/tour-impl.html

A

INTEGRATION GUIDE

This guide describes in detail how to integrate a new algorithm into *liboqs*. It is based on the integration of NTRU Prime done in the context of this project and contains a list of steps that must be done in any order prior to compiling *liboqs*.

The required steps are as follows:

- Create the directory /src/kem/<NewAlgName> and copy the algorithm's implementation there.
- Create a "Makefile.am" (GNU automake file) that compiles the implementation into a static library.
- Inside the new directory, create the file kem_<NewAlgName>.h defining the correct values for size (in bytes) of the public key, secret key and ciphertext of the algorithm and declaring the liboqs API functions that will be invoked when initializing and using the algorithm.
- Inside the new directory, create the file kem_<NewAlgName>.c, implementing the function that will initialize the algorithm's data structures.
- Change the file /src/kem/kem.h
  - Declare the algorithm identifier (string, must be unique).
  - Increment the *OQS_KEM_algs_length* variable by the appropriate ammount.
  - Include oqs/kem_<NewAlgName>.h (the file will be placed in the correct location by the build scripts at compile time).
- Change the file /src/kem/kem.c
  - Add the algorithm identifier to the *OQS_KEM_alg_identifier* structure.
  - In the *OQS_KEM_new* function, add the required code to invoke the new algorithm initializer.

- Change the file /config/features.m4 to add the required conditions to enable the new algorithm.
- Change the file /configure.ac to add the new algorithms makefile to the *AC_CONFIG_FILES* array.
- Change the file /Makefile.am
  - Check the condition to enable the new algorithm, and include its makefile if the condition is true.
  - Add the file oqs/kem_<NewAlgName>.h to the *instalheader_HEADERS* list.
  - Add the required instructions to copy the algorithms header file(s) to the appropriate directory.
- Set the environment variables *KEM_DEFAULT* and *SIG_DEFAULT* with the algorithm identifiers of the desired default KEM and signature scheme, respectively. If this step is omitted, *liboqs* will use default values for both options.
- Build *liboqs* according to the instruction on the readme file. When executing the ./configure file, the *prefix* flag should de set to the directory of the OpenSSL instalation we mean to use with liboqs.
- Build OpenSSL according to the instructions on the config file.

B

PYTHON (CRYPTOGRAPHY) USE GUIDE AND DOCUMENTATION

B.1 USE GUIDE

In order to use the Python package Cryptography with the post-quantum algorithms available, the following steps must be taken in order (assuming the respective algorithms were already built). This guide is based on a Python 3.6 installation.

- Install the Cryptography package from source with the new algorithms included. This can be done, for example, by opening a terminal window on the root directory of the package source code and running the following command (assuming the user uses pip as a Python package manager).
  - pip3 install -e .
- Set the environment variable **LIBOQS_INSTALL_PATH** with the path to liboqs shared object (liboqs.so). This can be done by setting the variable permanently on the OS settings or exporting it only to the current terminal session.
- On the Python source code, include the appropriate import statements (one or more of the following):
  - from cryptography.hazmat.primitives.ntru import NTRU
  - from cryptography.hazmat.primitives.ntrup import NTRUP
  - from cryptography.hazmat.primitives.falcon import Falcon
- Instantiate and use the imported classes according to the provided documentation.

B.2 DOCUMENTATION

This section contains more detailed documentation about the Cryptography's integration of post-quantum KEMs and signature schemes. The NTRU Prime integration was omitted since it follows the same structure of NTRU.

B.3 KEM - NTRU

The NTRU class contains the following methods and functions:

- ___init___(self, N)
  - N - Security parameter (must be valid or an error will be thrown).
- generate_keys(self)
  - Returns a touple (public_key, private_key)
- encap(self, public_key)
  - Returns a touple (ciphertext, shared_secret)
- decap(self, cyphertext)
  - Returns the shared secret

B.4 SIGNATURE - FALCON

The Falcon class contains the following methods and functions:

- ___init___(self, N)
  - N - Security parameter (must be valid or an error will be thrown).
- sign(self, message)
  - Returns a touple (signature, public_key)
- verify(self, message, signature, public_key)
  - Returns the result of the verification (True or False).

C

NTRU C++ IMPLEMENTATION - DOCUMENTATION

This C++ implementation of the NTRU algorithm consists simply of the NTRU class, which contains all the required functions to use the NTRU algorithm as a key exchange mechanism (KEM). Note that in many cases, the output buffer size is not directly set by the functions because it can be directly computed from the parameter N, which is public and should be a known value for every party in the handshake. The functions correspondig to the public API of the class are as follows:

- Constructor: NTRU(int N, int q, int p)
  - N, q and p are simply the security parameters of NTRU (integers).
- int generate_keys()
  - Generates a random key pair and stores both keys (public and private) as properties of the class. Returns 0 on success.
- int get_public(int* pk)
  - Stores the public key in the parameter pk. Must be called only after generate_keys() since the keys are not created when instantiating the class. Returns 0 on success.
- int encap(int* pk, int* ss_e, int* ss_c)
  - pk - Public key to be used for encapsulation.
  - ss_e - The shared secret (encrypted).
  - ss_c - The shared secret (cleartext).
  - Encapsulates a shared secret with the provided public key and outputs it in both encrypted and cleartext form. Since this class is meant to be used as a key exchange mechanism (KEM), not as a generalist public key cipher, the shared secret is based on a (pseudo)random number generator. Returns 0 on success.
- int get_ss(int* ss)

  - Returns the (clear text) shared secret. It must be called after encap() since the random shared secret is only generated when encap() is called. Returns 0 on success.

- int decap(int* ss_e, int* ss_c)
  - ss - The encrypted shared secret.
  - Decapsulates the encrypted shared secret (ss_e) and stores the cleartext shared secret (decrypted using the private key stored as a property of the class) in ss_c. It must be called after generate_keys(), otherwise no private key exists that can be used to decapsulate. Returns 0 on success.