Worcester Polytechnic Institute

Major Qualifying Project Report

Performance Evaluation of Round 2 Submissions for the NIST Post-Quantum Cryptography Project

Kevin Baptista

May 2020

Project Advisor: Professor Berk Sunar

A Major Qualifying Project Report submitted to

the faculty of Worcester Polytechnic Institute

in partial fulfillment of the requirements for

the Degree of Bachelor of Science

MQP BS2 1904

Contents

Chapter 1

Introduction

This paper looks at the submissions for round 2 of a competition held by National Institute of Standards and Technology (NIST) to find an encryption standard resistant to attacks by post-quantum computers. NIST announced its call for submissions in February 2016 with a deadline of November 2017 and announced the 69 algorithms that made the cut for round 1. In January 2019 the candidates for round 2 were announced with round 3 projected for 2020/2021.

1.1 Motivation

As quantum computers become more common the need for a new encryption standard becomes more prominent since they can potentially provide the power to brute force current encryption standards in a relatively short amount of time. Because of this NIST has begun looking for a new encryption standard that is secure against quantum computers.

In addition to security, the performance is an important factor when considering a new encryption standard. This study looks at all the round 2 submissions and compares the category 1 implementations (equivalent to AES-128) in terms of encryption/decryption time and its memory footprint.

Chapter 2

Background

This section gives a brief description of each submission along with the data collected from benchmarking. Each submission falls into one of three catagories: code-based, lattice-based or elliptic curve-based. Each of these have their respective trade-offs which will be discussed before getting into the submissions themselves.

2.1 Code-based submissions

Code-based cryptography refers to encryption schemes that use error correcting code. The advantage of code based schemes is that they are generally faster with the tradeoff of larger key sizes. A well known example of this is Classic McEliece which has been around for about thirty years and still does not have any known attacks even with a quantum computer. In the case of Classic McEliece, it uses a random code as the private key and a randomly permuted version of that code as the public key while the ciphertext is a codeword with errors that only the private key can decipher [1].

2.2 Lattice based submissions

Lattice based cryptography relies on the difficulty of problems related to lattices, namely finding the shortest non-zero vector in an arbitrary basis that represents a lattice. The most common variation of these schemes are Learning With Errors (LWE) and Ring Learning With Errors (RLWE). While this is a more recent method, it is thought to be secure against quantum computers [2].

2.3 Multivariate based submissions

Mutivariate cryptography schemes have their security based on the multivariate quadratic polynomial problem. This problem has been proven to be non-deterministic polynomial time over any field and believed to be hard on both classical and quantum computers [3]. Multivariate encryption is a popular choice amongst the signature schemes and its main advantages include its speed, modest computational requirements, and short signatures while its main drawback is its large public key sizes.

2.4 Elliptic Curve based submissions

Elliptic curve cryptography (ECC) is already widely used and well studied. While traditional ECC is more easily broken by quantum computers using Shor's algorithm isogeny based schemes are conjectured to be secure against quantum attacks [4].

2.5 Encryption Schemes

The following submissions are key encapsulation methods, an encryption scheme that uses a public key to create a ciphertext containing a symmetric key and in order to decrypt one must have the correct private key. This is primarily used for transmitting data that needs

to be secured over an insecure network.

2.5.1 BIKE

Bit Flipping Key Encapsulation (BIKE) is a code based submission which uses Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) code using bit-flipping coding techniques [5]. All variations of BIKE make use of ephemeral keys to protect against the GJS Reaction attack. BIKE has three variations of the scheme, each one with their optimizations focused on different areas; the exact differences and advantages/disadvantages each one has over the other can be found in the specification document.

2.5.2 Classic McEliece

While most code-based submissions are based off of the cryptosystem developed by McEliece in 1978 they all make their own adjustments whereas Classic McElice, as the name would imply, uses the original design which still doesn't have any known effective attacks [6]. Classic McEliece uses a random Goppa code and adds random errors in order to encrypt the message. This is extremely secure however as the documentation states, it comes at the cost of efficiency, with very large key sizes and key generation times. The documentation also claims that the hardware implementation is noticeably faster than the software implementation.

2.5.3 HQC

Hamming Quasi-Cyclic (HQC) is a code based encryption scheme proved IND-CPA assuming the hardness of the Syndrome Decoding on structured code [7]. Some of the other features it chooses to highlight are that the assumption that the family of codes being used is indistinguishable among random codes is not required and that it features a decryption failure probability analysis. HQC claims to be immune against attacks aiming at recovering the hidden structure of the code being used. A limitation of HQC is the low encryption rate;

increasing this rate requires increasing its other parameters.

2.5.4 NTS-KEM

NTS-KEM can be seen as variant of the McEliece and Niederreiter public key encryption schemes, however it is not concerned with the secure communication of an encrypted message but rather the secure communication of a random key [8]. NTS-KEM claims to have strong security guarantees and demonstrates a tight relationship between the IND-CCA security of NTS-KEM and the problem of inverting the McEliece PKE scheme. It also claims to be very flexible allowing for more fine-tuning of trade-offs. One of the notable disadvantages of NTS-KEM as noted by the documentation is the large key size.

2.5.5 ROLLO

ROLLO is a compilation of three other code based encryption schemes; namely LAKE, LOCKER, and Rank-Ouroboros. All three are based on rank metric codes and share the same decryption algorithm for Low Rank Parity Check (LRPC) codes [9]. ROLLO is designed to be efficient with its key size and computational complexity while also having a constant time decoding algorithm and a well studied failure probability. A limitation noted in the documentation is the fact that the harness of the problem ROLLO uses is relatively recent and is still being studied but considered hard by the community.

2.5.6 RQC

Rank Quasi-Cyclic is very similar to HQC and has the same advantages with the addition of the decryption algorithm being deterministic, thus having a decryption failure rate of zero, and having more attractive parameters than most Hamming based proposals [10]. It also shares the same limitations as HQC, namely that the hardness needs to be better studied and is difficult to improve on complexity.

2.5.7 CRYSTALS-Kyber

The Cryptographic Suite for Algebraic Lattices (CRYSTALs) key encapsulation method named Kyber has its security based on the hardness of the LWE in module lattices problem [11]. The construction of Kyber has a two-stage approach: the IND-CPA-secure public-key encryption scheme is introduced and encrypts messages of a fixed length of 32 bytes. Then a slightly tweaked Fujisaki–Okamoto (FO) transform is used to construct the IND-CCA2-secure KEM. Some of the notable advantages of Kyber according to the documentation include its ease of implementation and scalability. Its limitations come compared to other lattice-based schemes where various implementations have different trade-offs on security or efficiency. Ring Learning-with-errors for example requires many more outputs in order to extract a matrix from a seed.

2.5.8 FrodoKEM

FrodoKEM is a lattice based encryption scheme designed to be conservative yet practical and derives its security from from cautious parameterizations of the well-studied LWE problem, which in turn has close connections to conjectured-hard problems on generic, "algebraically unstructured" lattices [12]. Some of the advantages FrodoKEM's documentation claims to have include its ease of implementation, compatibility with current schemes, potential for an efficient hardware implementation, and resistance to side-channel attacks. The disadvantage of this scheme is its large key size.

2.5.9 LAC

Lattice-based Cryptosystems (LAC) is a cryptographic suite that derives its strength from the RWLE problem. LAC achieves the required security but its focus is on size [13]. Some of the advantages as specified in the documentation include its consideration for multiple implementations such as the parallel design making it a very sutible design for multi-core

processors. Other advantages include the simplicity and flexibility of the design. Some of the limitations include an inability to be sped up on processors that do not support vector instructions and a sacrifice of efficiency to protect against timing attacks.

2.5.10 NewHope

New Hope is a lattice based encryption scheme described as a variant of the encryption scheme by Lyubashevsky, Peikert and Regev which uses RWLE while also applying size reduction techniques [14]. The main advantages according to the documentation include its performance, ease of implementation, and conservative design. Some of its limitations are its small noise distribution, limited parameterization, and restrictions due to usage of Number Theory Transform.

2.5.11 NTRU

The lattice based scheme this implementation is based of off, NTRU, was first described by Hoffstein, Pipher, and Silverman with some changes by Hülsing, Rijnveld, Schanck, and Schwabe [15]. This scheme has a tight proof of IND-CCA2 security in the random oracle model assuming the DPKE is OW-CPA secure while also being secure in the quantum random oracle model. Some of the advantages include the fact that it is well studied, its flexibility, speed, and lack of a patent while some of its limitations include its size and lack of understanding when it comes to optimal parameters.

2.5.12 NTRU-Prime

NTRU-Prime is described as prime-degree large-Galois-group inert-modulus ideal-lattice-based cryptography which takes away various mathematical tools from an attacker [16]. Some of the advantages of NTRU-Prime include it being resistant to side-channel analysis and its efficiency. Some of the disadvantages mentioned in the paper include the security

levels being widely separated.

2.5.13 Round5

Round5 is a lattice based merger of 2 submissions: Round2 and HILA5 deriving its security from the General Learning With Rounding problem. It uses an error-correcting code to decrease decryption failure probability granting smaller key sizes and better performance [17]. Some of Round5's advantages include its flexibility, small public-key and ciphertext, and efficiency while no major limitations are specified in the documentation.

2.5.14 ThreeBears

ThreeBears is based on the lattice based Lyubashevsky-Peikert-Regev and Ding RWLE cryptosystems while also deriving from NewHope and Kyber. The main difference with ThreeBears is that it derives its security from the integers modulo a generalized Mersenne number, thereby making it integer module learning with errors, similar to Gu's work [18]. Some of the advantages of ThreeBears include its simplicity, size, and speed while its disadvantages include the fact that the problem it derives its security from is still fairly novel and potentially being more complex to protect from side channel or hybrid attacks.

2.5.15 SIKE

Supersingular Isogeny Key Encapsulation (SIKE) is an elliptic curve based submission which uses the Diffie-Hellman key exchange enhanced with supersingular isogeny in order to protect from quantum attacks [19]. The main advantages of SIKE are its small key sizes and how widely used and well studied ECC has been. The large amount of studies surrounding ECC allows most of the knowledge, such as implementations that avoid side-channel attacks, surrounding it to be transferred to SIKE. The main disadvantage of SIKE is its performance, taking orders of magnitude longer than the lattice or code based submissions.

2.6 Signature schemes

The following submissions are digital signature algorithms which also make use of asymetric cryptography but have the objective of ensuring that the data received is truly from who it is claimed to be from.

2.6.1 Crystals-Dilithium

Dilithium is based on the "Fiat-Shamir with Aborts" approach; this technique uses rejection sampling for compactness and security [20]. To be more efficient Dilithium uses a uniform distribution instead of the traditional Gaussian distribution. It shares most of the advantages and disadvantages that Kyber has such as its ease of implementation and small key size.

2.6.2 FALCON

Fast-Fourier Lattice-based Compact Signatures over NTRU (FALCON) has its security based on NTRU lattices and uses fast Fourier sampling as a trapdoor sampler [21]. FALCON was designed to be compact and thus this is its main advantage. Some of its other advantages include fast signature generation and verification, a modular design, and a message recovery mode. Some of the disadvantages documented in the specification include its delicate implementation, floating point arithmetic, and a lack of information on side-channel attacks.

2.6.3 qTesla

qTesla has its security based on the hardness of RLWE and is claimed to be an efficient variant of the Bai-Galbraith signature scheme which is based on the Fiat-Shamir with Aborts framework. It is designed to be simple, practical, and portable [22]. Some of the advantages according to the documentation of qTesla include its tight security foundation and ease of implementation and integration with other libraries. Although no disadvantages are explicitly stated, it likely suffers the same disadvantages most lattice-based submissions suffer.

2.6.4 GeMSS

A Great Multivariate Short Signature (GeMSS) is a multivariate signature scheme based off the QUARTZ and Gui signature schemes [23]. GeMSS is claimed to be a faster variant of QUARTZ, which currently has no practical attack reported against it. The main advantage GeMSS is centered around is its efficiency while its main drawback is the public key size being rather large.

2.6.5 MQDSS

MQDSS has its security based on the multivariate quadratic problem and follows the Fiat-Shamir paradigm of transforming identification schemes to signatures using the 5-Pass SSH scheme [24]. Some of the advantages MQDSS is designed to have are its small key sizes, security of the multivariate quadratic problem, simplicity, and ability to be parallelized. Some of the disadvantages described in the documentation include its large signature size and the problems associated with rewinding of the adversary and adaptively programming the random oracle.

2.6.6 Rainbow

Rainbow is a multivariate signature scheme originally proposed in 2005 and adapted for a post quantum application. It is described as a multi-layer Oil-Vinegar system [25]. Some of the advantages documented in the specification include its effiency, short signatures, and simplicity while the disadvantage is its large public and private key size.

2.6.7 LUOV

Lifted Unbalanced Oil and Vinegar has its security based on multivariate cryptography while attempting to reduce the size of public keys generally associated with oil and vinegar systems. [26] Some of the advantages the documentation makes note of include its small

signature and secret key, wide security margin, and flexibility. While the public key size is much smaller than that of other multivariate schemes, it's still quite large and considered one of LUOV's limitations although this is a trade-off for the small signature size and can be modified depending on the application.

2.6.8 Picnic

Picnic is unique in the sense that it does not rely on the harness of any mathematical problem, relying instead on zero-knowledge proof (i.e. one party can prove to the other that it knows a secret without disclosing the secret). Picnic has two variations, one that uses the Fiat-Shamir transform which is secure in the random oracle model (but also has no known quantum attacks) and the Unruh transform, secure in the quantum ransom oracle model but crates a larger signature [27].

2.6.9 SPHINCS+

SPHINCS+ is a stateless hash-based signature scheme, it authenticates a large number of few-time signature key pairs using hyper trees [28]. Some of the advantages the documentation claims SPHINCS+ to have are its small key sizes and that it is probably the most conservative design of a post-quantum signature scheme thanks to its minimal security assumptions. Its drawback is the signature size being quite large and its speed.

Chapter 3

Performance Analysis of Submissions

This chapter will go over how the methodology used to collect data on the performance of each scheme. Specifically, the time each program took to encrypt and decrypt was measured along with its memory footprint. The findings of the experiment and some comments on each scheme are also found in this chapter.

3.1 Methodology

All timing measurements were made using the standard `time.h` package which allows for nanosecond precision. The following code was used to measure encryption/decryption time. It ran encryption and decryption 1000 times and took the average times for each.

The program included with each submission for testing was also paused and the `pmap` function was used to record memory usage. This information can be useful for evaluating the viability of devices with limited memory such as embedded systems. While there may minor variations with how each submission implemented their program the memory usage recorded from this should still be a good approximation. Another important factor to consider is the key size as each encryption scheme has its own tradeoff of key size vs. speed in addition to the consideration for security.

```
crypto_kem_keypair(pk, sk);
for(int i = 0; i < TESTCOUNT; i++){
        clock_gettime(CLOCK_MONOTONIC, &start);
        crypto_kem_enc(c,k1,pk);
        clock_gettime(CLOCK_MONOTONIC, &end);
        encTime += BILLION * (end.tv_sec - start.tv_sec)
            + end.tv_nsec - start.tv_nsec;
        clock_gettime(CLOCK_MONOTONIC, &start);
        crypto_kem_dec(k2,c,sk);
        clock_gettime(CLOCK_MONOTONIC, &end);
        decTime += BILLION * (end.tv_sec - start.tv_sec)
            + end.tv_nsec - start.tv_nsec;
}

encTime = encTime/TESTCOUNT;
decTime = decTime/TESTCOUNT;

printf("Encrypt time: %lu \n",encTime);
printf("Decrypt time: %lu \n",decTime);
```

Listing 1: Code used for measuring time to encrypt/decrypt

This was done on a computer with an Intel i7-3520M @2.90 GHz (Ivy Bridge) running Xubuntu (a variant of Ubuntu 18.10 using the Xfce desktop environment).

3.2 Measurements

All timing measurements were made using the standard `time.h` library, specifically it used the monotonic clock. The monotonic clock allows elapsed time to be measured from an arbitrary point in time as opposed to a real time clock which would measure the difference between the system's best guess of the time at the beginning and end. Unless otherwise noted, all implementations were run as is with the default compiler options in the makefile.

Table 3.3 shows the average time elapsed from immediately before and after the encapsulation and decapsulation functions using the NIST Known Answer Tests (KATs). Table 3.4 shows the memory footprint, the 'Memory' refers to how much memory is allocated

to the process while the Resident Set Size (RSS) shows how much of that is actually located in RAM. The 'Unique' columns refer to the memory allocated excluding any libraries which may already be loaded on the device running the encryption scheme, this also excludes anonymous memory and the stack as these may also be attributed in part to shared libraries.

3.3 Encryption schemes

This section details the measurements obtained for the encryption schemes using the methods above. It will describe how each scheme compares to the others and issues or modifications if any were necessary.

Scheme	Encrypt (ms)	Decrypt (ms)	Public Key (bits)	Private Key (bits)
BIKE	0.167	1.963	1988	3090
HQC	1.068	1.597	2500	320
NTSKEM	0.034	0.234	319488	9248
ROLLO	0.179	0.596	3720	320
RQC	0.635	3.221	6824	320
Classic McEliece	0.091	22.856	2088960	51616
CRYSTALS-Kyber	0.211	0.263	13056	6400
FrodoKEM	0.718	0.694	159104	76928
LAC	0.064	0.091	4352	4096
NewHope	0.297	0.338	7424	15104
NTRU	1.361	3.927	5592	7480
NTRU Prime	8.598	19.995	7952	12144
Round5	0.067	0.024	5408	5664
ThreeBears	0.036	0.057	320	6432
SIKE	35.048	37.435	2992	2768

Table 3.1: Key sizes (in bits) and timings of encryption and decryption of schemes

Scheme	Memory (kB)	RSS (kB)	Unique Memory (kB)	Unique RSS (kB)
BIKE	18872	6544	56	56
HQC	17788	5496	64	64
NTSKEM	6936	4364	116	116
ROLLO	17784	5508	52	52
RQC	17784	5628	60	60
Classic McEliece	2564	1680	132	132
CRYSTALS-Kyber	5496	3084	40	40
FrodoKEM	7116	3248	60	60
LAC	5544	3140	88	88
NewHope	5492	2744	36	36
NTRU	5500	2908	44	44
NTRU Prime	2564	1544	132	128
Round5	5608	3140	152	152
ThreeBears	5560	2960	104	104
SIKE	2564	1680	132	132

Table 3.2: Memory footprint of encryption schemes

3.3.1 BIKE

BIKE is fairly quick when encrypting, however it is slightly slower than most other schemes when it comes to decrypting. It also has one of the larger key sizes, although this is expected for code-based algorithms. BIKE had very similar performance between its reference implementation and optimized implementation, however in the reference implementation `BIKE-1` was the default whereas in the optimized implementation `BIKE-2` was chosen and this had better performance. As for the memory profile, while it has the most memory allocated when accounting for the libraries, the memory of the encrypt/decrypt program itself is on the smaller end. The values shown in tables 3.3 and 3.4 refer to BIKE-2 targeting category 1 IND-CCA security.

3.3.2 HQC

The optimized implementation of HQC appears to be incompatible with the timing method used for the other encryption schemes. Using the same code that was used to time other

schemes causes it to fail to compile and changing some compiler options in the makefile allow it to compile but cause an illegal instruction when attempting to run it. Thus the information in the table refers to the reference implementation of `HQC-128-1` which may have worse performance than the optimized implementation. With this in mind, it is also worth noting that the key sizes are very small. HQC has one of the larger memory footprints with libraries accounted for but still fairly small when looking at the memory the program itself uses.

3.3.3 NTS-KEM

NTS-KEM is one of the faster schemes in both encryption and decryption but this comes at the cost of the second largest public key size and a fairly large private key. NTS-KEM had a significant improvement when going from the reference implementation to the optimized implementation, it was about 10 times faster than the reference implementation. The memory footprint is not particularly large or small when considering the use of libraries but the encrypt/decrypt is among the schemes that use a larger amount of memory. The values in the tables refer to the optimized implementation of `NTS-KEM(12,64)` which targets category 1 IND-CCA security.

3.3.4 ROLLO

ROLLO is fairly fast and has the smallest key sizes among the code-based submissions and is still one of the smaller options overall in terms of key size. The optimized version of ROLLO was about 1.4 times faster when compared to the reference implementation. ROLLO has a large memory footprint with all its libraries accounted for but has one of the smallest footprints on its own. The values in the tables refer to the optimized implementation of `ROLLO-I-128`, targeting category 1 IND-CCA security.

3.3.5 RQC

RQC is one of the slower encryption schemes but it has fairly small key sizes; it is tied with ROLLO for the smallest private key. The optimized implementation of RQC was marginally faster than the reference implementation. RQC, like most of the code based encryption schemes has a fairly large memory footprint when accounting for all the libraries but uses comparatively little memory on its own. The values in the tables refer to the optimized implementation of `RQC128`, targeting category 1 IND-CCA security.

3.3.6 Classic McEliece

Classic McEliece has one of the fastest encryption times, however it has the second slowest decryption time and some of the largest key sizes; its public key is the largest overall. It is extremely secure so it may be useful is specific scenarios where data is encrypted often but decrypted infrequently. The optimized implementation is a copy of the reference implementation. The values in the tables refer to `McEliece348864`, which targets category 1 IND-CCA security. Classic McEliece has one of the smallest memory footprints when accounting for the libraries but it's among the schemes that require the most amount of memory on its own, likely due to the large key sizes. The version used for this experiment was downloaded from the submitter website rather than the NIST submission page as there were issues compiling the the version on the NIST website while the one on the submitter website used OpenSSL.

3.3.7 Crystals-Kyber

Crystal-Kyber is not particularly fast or slow, its public key is larger than most but far from the largest while its private key is about in the middle. The optimized and reference implementations had very little difference in performance. The memory footprint in comparison to other schemes is fairly small both when considering the libraries and the memory allocated

for the program itself. The values in the tables refer to the optimized implementation of `Kyber512`, targeting category 1 IND-CCA security.

3.3.8 FrodoKEM

FrodoKEM is one of the slower implementations and has very large keys; it has the largest private key overall. The optimized implementation is about 4 times faster than the reference implementation. FrodoKEM does not have a particularly large or small memory footprint with libraries included but the memory allocated specifically for it is fairly small. The values in the table refer to the optimized version of `FrodoKEM-640`, which targets category 1 IND-CCA security.

3.3.9 LAC

LAC is one of the fastest schemes for both encryption and decryption and both the private and public key are among the smallest of all submissions. The optimized implementation of LAC is about 3 times faster than the reference implementation. When considering the memory footprint in comparison to other schemes, the memory allocated including the libraries is fairly small while the memory allocated specifically for it is not particularly large or small. The values in the tables refer to the speed test of `LAC128`, targeting category 1 IND-CCA security.

3.3.10 New Hope

New Hope is not especially fast nor slow and it has a larger key size than most, however these key sizes are far from the largest. There is almost no difference in performance between the reference and optimized implementation. New Hope has a small memory footprint with the libraries included and the smallest amount of memory allocated specifically for it among the compared schemes. The values in the table refer to `NewHope512CCA` which targets category

1 IND-CCA security.

3.3.11 NTRU

NTRU is one of the slower schemes but its key sizes are relatively small. There is little difference in performance between the optimized and reference implementation. NTRU has a fairly small footprint both when considering the memory allocated both with and without including libraries. The values in the tables refer to `NTRU-HRSS701`, targeting category 1 IND-CCA security.

3.3.12 NTRU Prime

NTRU Prime is one of the slowest schemes overall, it also has fairly large key sizes. The optimized implementation is a copy of the reference implementation so there is no difference in performance. NTRU Prime has the smallest memory footprint of all compared schemes when considering the memory allocated for libraries but one of the largest when considering memory allocated specifically for it. The values in the tables refer to `SNTRUP653` which targets category 1 IND-CCA security.

3.3.13 Round5

Round5 is very fast and has some of the smallest key sizes. The optimized implementation has significantly better performance than the reference implementation, nearly 100 times faster. Round5 has a small memory footprint when including the libraries' memory usage but has the largest amount of memory allocated specifically for it in comparison to other schemes. The values in the tables refer to `R5ND-1KEM-0D`, targeting category 1 IND-CCA security.

3.3.14 Three Bears

Three bears is very fast and has small keys, including the smallest public key. The optimized implementation is over 20 times faster than the reference implementation. The values in the tables refer to `BabyBear` which targets category 1 IND-CCA security. The memory footprint when considering libraries is fairly small while the footprint specific to the encrypt/decrypt program is not particularly large or small relative to other schemes. In order for the code responsible for timing to work the compiler flag `-std=c11` needed to be removed, this may cause minor changes to performance.

3.3.15 SIKE

SIKE has the longest encryption and decryption times, however it also has some of the smallest keys. The optimized implementation is over 10 times faster than the reference implementation. The values in the tables refer to `SIKEp434`, targeting category 1 IND-CCA security. This is the uncompressed implementation, with compression the key size can be even smaller at the sacrifice of speed. The memory footprint is one of the smallest when considering libraries while the amount of memory allocated for it specifically is among the largest.

3.4 Signature Schemes

Signature schemes were measured the same way as the encryption schemes, using the same library and code to measure signing and verifying time. Unlike the encryption schemes, however, many of the signature schemes did not have a version optimized for the test platform. In order to have all platforms be on equal ground the data collected and shown in the tables come from the reference implementation. For the signature schemes that did have an optimized version compatible with the test platform, the performance difference will be noted when going into more detail for each scheme.

Scheme	Sign (ms)	Verify (ms)	Public Key (bits)	Signature (bits)
Dilithium	0.840	0.101	9472	16352
LUOV	14.165	9.911	99496	256
FALCON	0.078	0.016	7176	6472
GeMSS	18.099	13.344	384000	282
MQDSS	45.327	32.153	128	166832
Picnic	45.525	29.817	32	34032
qTesla	0.187	0.03	119040	20736
Rainbow	0.173	0.138	152576	512
SPHINCS+	50.62	2.083	64640	256

Table 3.3: Key sizes (in bits) and timings of signature schemes

Scheme	Memory (kB)	RSS (kB)	Unique Memory (kB)	Unique RSS (kB)
Dilithium	5524	3048	68	68
LUOV	5964	3096	508	220
FALCON	4300	2708	256	256
GeMSS	19476	8388	160	160
MQDSS	5556	3180	40	40
Picnic	3752	2044	1052	232
qTesla	5532	3016	76	76
Rainbow	6952	5348	76	76
SPHINCS+	5508	3156	52	52

Table 3.4: Memory footprint of signature schemes

3.4.1 CRYSTALS-Dilithium

Dilithium when compared to all other signature schemes is among the faster option with a fairly small public key and signature size. Comparing Dilithium to the other lattice based entries makes it the slowest but also the smallest in terms of its key and signature size. The performance in the tables refers to `Dilithium2` as it is equivalent to NIST security level 1. Interestingly, one can get better performance by using Dilithium1, however this has a security level below that of NIST's lowest level. Dilithium does not have an optimized version with its submission.

3.4.2 FALCON

FALCON is the fastest of all schemes and has a fairly small public key and signature. `FALCON512` was used to obtain the values in the tables. Using the optimized version would cause signatures to fail for an unknown reason, this may be due to something being incompatible with the test platform.

3.4.3 qTesla

qTesla is one of the fastest signature schemes but also has the third largest public key and signature. `qTesla_I` was used to obtain the values in the tables, it targets NIST security level I and does not have an optimized version.

3.4.4 GeMSS

GeMSS is not particularly fast nor slow in comparison to other schemes and although it has the largest public key, it has one of the smallest signatures. `RedGemss128` was used as it had the best performance while the other variations traded off performance for additional security. In the optimized version, the sign time was about 10 times faster and the verification time 100 times faster.

3.4.5 MQDSS

MQDSS was one of the slowest signature schemes and has a fairly large signature, however it has one of the smallest public keys. `mqdss-48` was used to obtain the values shown in the tables, the optimized version appears to be incompatible with the test platform.

3.4.6 Rainbow

Rainbow is one of the fastest signature schemes and has one of the smallest signatures but has the second largest public key. Rainbow `Ia_classic` was used to obtain the values in the

tables and the optimized version has a slightly faster signing time and is about twice as fast to verify. The flag `-std=c11` needed to be removed from the makefile in order for the library used for timing to be compatible.

3.4.7 LUOV

LUOV is fairly fast and is tied for the smallest signature while its public key is not particularly large or small. `luov-8-58-237-keccak` was used to obtain the values in the tables. The optimized version has a signature time about 7 times faster and verification time about 10 times faster.

3.4.8 Picnic

Picnic is among the slower signature schemes and has the largest signature, however it has the smallest public key by far. `Picnicl1fs` was used to obtain the values in the tables. The optimized version tells a very different story and puts Picnic as the fastest in terms of sign time with an average sign time of 0.007 ms and a verify time of 3.068 ms.

3.4.9 SPHINCS+

SPHINCS+ is tied for the smallest signature, its public key is not particularly large or small compared to other signature schemes, and it's not particularly fast or slow when it comes to verifying, however it has the slowest sign time. `sphincs-sha256-128f-simple` was used to obtain the values in the table and the optimized version had very similar performance compared to the reference version. The flag `-std=c99` needed to be removed in the makefile in order for the code responsible for timing to be compatible.

Chapter 4

Conclusion

For both encryption and signature, each scheme makes it own performance trade-off to either minimize key size or maximize speed. Another important factor in evaluating signature schemes is the signature size, as reducing the key size usually required increasing signature size and vice versa. The NIST page for post-quantum encryption states that multiple submissions may be selected. Different scenarios may have different priorities when it comes to performance so having more than one standard would allow each user to use what is best suited for each application. This study considered the security of each scheme equal and compared them purely on performance; with this in mind some notable encryption schemes include ThreeBears, LAC, and Round5 as these were among the fastest while maintaining fairly small keys, albeit not the smallest. FALCON and CRYSTALS-Dilithium both have the public key and signature amongst the smallest while remaining fairly fast. While these schemes are balanced, the schemes with the smallest public key or signature is still multiple orders of magnitude smaller so depending on the application one may prefer another scheme.

Future work that could use information from this study would include taking into account the security when comparing the trade-offs one makes with each scheme. While all schemes were tested with the version that corresponds to NIST level 1 security, some implementations may be more secure than others and this was not really considered in this study.

Bibliography

[1] Jintai Ding and Bo-Yin Yang. Code-based cryptography. In *Post-quantum cryptography*, pages 93–145. Springer, 2009.

[2] Jintai Ding and Bo-Yin Yang. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

[3] Jintai Ding and Albrecht Petzoldt. Current state of multivariate cryptography. *IEEE Security & Privacy*, 15(4):28–36, 2017.

[4] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2011/506, 2011.

[5] Nicolás Aragón, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, and Gilles Zémor. BIKE: Bit Flipping Key Encapsulation. 2017.

[6] Daniel J Bernstein, Tung Chou, Tanja Lange, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Peter Schwabe, Jakub Szefer, and Wen Wang. Classic McEliece: conservative code-based cryptography 30 March 2019. 2019.

[7] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. Hamming Quasi-Cyclic (HQC). 2017.

[8] Martin Albrecht, Carlos Cid, Kenneth G Paterson, Jung Tjhai Cen, and Martin Tomlinson. NTS-KEM. 2019.

[9] rollo-rank-ouroboros, lake & locker.

[10] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Rank Quasi-Cyclic (RQC), 2017.

[11] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation. 2019.

[12] Erdem Alkim, Joppe W Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, Douglas Stebila, et al. FrodoKEM learning with errors key encapsulation, 2019.

[13] Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, Zhenfei Zhang, Zhe Liu, Hao Yang, Bao Li, and Kunpeng Wang. LAC: Lattice-based Cryptosystems (2019). *URL: https://csrc. nist. gov/projects/post-quantum-cryptography/round-2-submissions. Citations in this document*, 1(8.3).

[14] Thomas Pöppelmann, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, Douglas Stebila, Martin R Albrecht, Emmanuela Orsini, Valery Osheter, et al. NewHope. 2019.

[15] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU: algo-

rithm specifications and supporting documentation (2019). *URL: https://csrc. nist. gov/projects/post-quantum-cryptography/round-2-submissions. Citations in this document*, 1, 2019.

[16] Daniel J Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU Prime: reducing attack surface at low cost. In *International Conference on Selected Areas in Cryptography*, pages 235–260. Springer, 2017.

[17] Hayo Baan, Sauvik Bhattacharya, Scott Fluhrer, Oscar Garcia-Morchon, Thijs Laarhoven, Rachel Player, Ronald Rietman, Markku-Juhani O Saarinen, Ludo Tolhuizen, Jose-Luis Torre-Arce, et al. Round5: KEM and PKE based on (Ring) Learning With Rounding (2019). *URL: https://csrc. nist. gov/projects/post-quantum-cryptography/round-2-submissions. Citations in this document*, 1.

[18] Mike Hamburg. Post-quantum cryptography proposal: ThreeBears. *NIST Post-Quantum Cryptography Standardization*, 2019.

[19] Matthew Campagna, Craig Costello, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, David Urbanik, et al. Supersingular Isogeny Key Encapsulation. 2019.

[20] Láo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS–Dilithium: Algorithm Specification and Supporting Documentation. Round-2 submission to the NIST PQC project, 2019.

[21] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-Fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 2018.

[22] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo SLM Barreto, Johannes Buchmann, Juliane Krämer, Patrick Longa, Harun Polat, Jefferson E Ricardini, and Gustavo Zanon.

Submission to NIST's post-quantum project (2nd round): lattice-based digital signature scheme qTESLA. 2019.

[23] Antoine Casanova, Jean-Charles Faugère, Gilles Macario-Rat, Jacques Patarin, Ludovic Perret, and Jocelyn Ryckeghem. Gemss: A great multivariate short signature. *Submission to NIST*, 2017.

[24] Ming-Shing Chen, Andres Hülsing, Joost Rijneveld, and Peter Samardjiska, Simona nad Schwabe. QDSS Specifications. *Submission to NIST*, 2019.

[25] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang. Rainbow. *Submission to NIST*, 2019.

[26] W Beullens, B Preneel, A Szepieniec, and F Vercauteren. LUOV signature scheme proposal for NIST PQC project (Round 2 version), 2018.

[27] Z Greg et al. The Picnic Signature Algorithm Specification, 2019.

[28] JP Aumasson, DJ Bernstein, C Dobraunig, M Eichlseder, S Fluhrer, SL Gazdag, A Hülsing, P Kampanakis, S Kölbl, T Lange, et al. SPHINCS+–Submission to the NIST post-quantum project. *Submission to NIST*, 2019.