NTNU
Norwegian University of
Science and Technology

On the Development and Standardisation of Post-Quantum Cryptography

A Synopsis of the NIST Post-Quantum
Cryptography Standardisation Process, its
Incentives, and Submissions

Maja Worren Legernæs

Master of Science in Communication Technology
Submission date: June 2018
Supervisor: Danilo Gligoroski, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Problem Description

This Master's Thesis is written at the Norwegian University of Science and Technology during the spring semester of 2018.

The National Institute of Standards and Technology announced in December 2016 that they were taking submissions for quantum-resistant algorithms and that they would be testing and evaluating these submissions after the submission deadline in November 2017. In conjunction with this development, this thesis contains background information on the motivation and idea behind this announcement, general information about today's cryptography as compared to post-quantum cryptography, information about several of the submission types, as well information about the specific algorithms, their characteristics, specifications, strengths, and weaknesses. It is assumed that the reader has moderate, general knowledge within the fields of mathematics, physics, and cryptography.

Responsible professor: Danilo Gligoroski

Trondheim, Monday 11th June, 2018

Maja Worren Legernæs

Acknowledgements

I want to thank my supervisor, Professor Danilo Gligoroski, for his efficient explanations and feedback throughout. Your knowledge within the field of cryptography is truly inspiring.

I would also like to thank my parents, Dag Legernæs and Mai Britt Worren, who, despite their differences, never once doubted my intelligence or ability to achieve. You always encouraged the acquisition of knowledge as the noblest of pursuits, and I carry that with me every day.

I would also like to extend a thank you to Maren Grøndahl for countless discussions and abundant encouragement over the last year. My motivation would have suffered greatly without you.

Maja Worren Legernæs

Abstract

Due to developments within the field of quantum computers, the need for developing and implementing quantum-resistant cryptographic algorithms has become more urgent. Using such computers, many of today's most prominent algorithms will be broken by Shor's Algorithm. This is an algorithm which utilises quantum computing to compare the phases of prime numbers represented as sine waves to factorise great integers, effectively solving the discrete logarithm problem on which many modern cryptographic algorithms are based. While the development of quantum computers is by no means finished, we know from previous experience that the efforts needed to fully replace a well-established cryptographic algorithm are long and laborious, both when it comes to development, testing, standardisation, and distribution. In addition to this, such algorithms must be put to use significantly longer before the older, non-quantum-resistant algorithms are broken by a quantum computer, to ensure that sensitive or secret information which is now encrypted with today's non-quantum resistant algorithms will no longer be sensitive or desirable when this encryption is no longer secure.

Due to all of these factors, the National Institute of Standards and Technology (NIST) has issued a call for public submissions for quantum-resistant asymmetric cryptographic algorithms. The deadline for the submissions was 30$^{\text{th}}$ of November 2017.

This paper is written as an overview of the most recent developments towards post-quantum cryptography standardisation, and the motivations behind it. Insight into the field of cryptography, quantum computers, Shor's algorithm, and the mathematical construction of several of the most vital non-quantum resistant cryptographic algorithms used today are given, as well as the reasons why they are not quantum-resistant. In addition to this, it looks into the most promising quantum-resistant cryptography families, and their mathematical construction. Most vitally, the paper gives an overview of all the non-withdrawn algorithm submissions given to NIST during their Post-Quantum Standardisation process, including their mathematical type, specifications, characteristics, as well as size and execution time comparisons of all their proposed implementations. A sorting of the submissions with the lowest space requirements, fastest execution times, as well as an intersection between these two is also created. A detailed account of the requirements used during the creation of this ranking is presented. This sorting is created using the original submissions given to NIST, and only takes into account any attacks discovered against these as of June 2018, but the methodology used can be utilised for any future versions of the submitted algorithms.

Due to the nature of post-quantum cryptography research and testing, this thesis is constructed as a general guide into the subject as well as a study of the cryptographic submissions given to NIST and their characteristics. The thesis has been limited to the most vital cryptography and theory, mathematically and otherwise, analysis of the submitted algorithms, as well as any discussion of these. This is both due to the time constraints which follow a master's thesis, as well as to ensure that the thesis attains the correct focus.

Sammendrag

Med bakgrunn i utviklingen av kvantedatamaskiner og utviklingens hastighet, har behovet for å utarbeide kvanteresistante, kryptografiske algoritmer blitt stadig mer pressende. Ved bruk av kvantedatamaskiner vil en stor del av dagens fremtredende, kryptografiske algoritmer bli løst av Shors algoritme. Forenevnte algoritme benytter kvanteberegning for å sammenligne faser av primtall representert som sinusbølger for å faktorisere store heltall. Dette medfører en effektiv løsning av det diskrete logaritme-problemet, hvilket flere av de mest moderne kryptografiske algoritmer bygger på.

Utviklingen av kvantedatamaskiner er ikke ferdig, og det eksisterer bred enighet i fagmiljøet om at arbeidet som trengs for å erstatte en veletablert, kryptografisk algoritme er både langtekkelig og vanskelig. Dette gjelder alle faser av utviklingens forløp, som utvikling, testing, standardisering, og distribusjon. Det er også viktig at eventuelle løsninger som utvikles må tas i bruk lenge før de eldre, ikke-kvanteresistente algoritmene blir løst. Dette er for å sikre at informasjon kryptert med dagens ikke-kvanteresistente algoritmer er eldet til den grad at det ikke lenger er sensitiv så snart krypteringen knekkes av kvantedatamaskiner.

På bakgrunn av blant annet disse faktorene har National Institute of Standards and Technology (NIST) utstedt en innkalling til offentlige innleveringer av kvanteresistante, asymmetriske, kryptografiske algoritmer. Fristen for innleveringene var 30. november 2017.

Denne mastergraden er skrevet som en oversikt over de siste utviklingene innen standardisering av kvanteresistant kryptografi, og deres bakenforliggende motivasjoner. Det gis innsikt i kryptografi, kvantedatamaskiner, Shors algoritme, og den matematiske konstruksjonen av flere av de mest vitale ikke-kvanteresistante, kryptografiske algoritmene som brukes i dag. I tillegg framgår flere av årsakene til at de ikke er kvanteresistante, samt informasjon om de mest lovende, kvanteresistante, kryptografiske algoritmene, og deres samsvarende, matematiske konstruksjon. Kanskje aller viktigst, gir prosjektet en oversikt over alle innleverte algoritmer til NIST over forløpet av deres standardiseringsprosess. Oversikten omfatter algoritmenes matematiske type, spesifikasjoner, egenskaper, størrelses- og kjøretidssammenligninger av alle deres foreslåtte implementeringer. Algoritmer som senere har blitt trukket fra standardiseringsprosessen er ikke tatt hensyn til. En sorterig av de innleverte algoritmene er konstruert ved bruk av nødvendig lagringsplass, kjøretider, og en kombinasjon av disse to. Detaljer om kravene brukt i denne prosessen er presentert. Denne sorteringen tar kun hensyn til de algoritmene som originalt ble sendt inn til NIST, og alle angrep som har blitt oppdaget mot dem til dags dato (Juni 2018), men metoden kan brukes på alle senere versjoner av de samme algoritmene.

Med bakgrunn i forskningens og testingens karakter innen kvanteresistent kryptografi, er denne avhandlingen konstruert som en generell guide til emnet.

Studiet er begrenset til kun grunnleggende og matematisk konstruksjon av relevante kryptografiske algoritmer, analyse av de innleverte algoritmene, og generelle tanker rundt dette. Dette er grunnet tidsbegrensningene som følger med å skrive en mastergrad, og for å sikre riktig fokus i oppgaven. Dernest er videre studier av kvantedatamaskiner og kvantefysikk ikke er relevant for denne avhandlingen.

Contents

List of Figures

List of Tables

Abbreviations

AKE	Authenticated Key-Establishment
BCH	Bose–Chaudhuri–Hocquenghem
CCA	Chosen-Ciphertext Attack
CFS	Courtois, Finiasz, and Sendrier
CMA	Chosen-Message Attack
CPA	Chosen-Plaintext Attack
cpb	Cycles per Byte
CSSI	Computational Supersingular Isogeny
CVP	Closest Vector Problem
Compact-LWE	Learning With Secretly Scaled Errors in Dense Lattice
DFR	Decoding Failure Rate
DLP	Discrete Logarithm Problem
DQCSD	Decisional Quasi-Cyclic SD
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie–Hellman
EIP	Extended Isomorphsm of Polynomials
EUF-CMA	Existential Unforgeability under Chosen Message Attack
FALCON	Fast Fourier lattice-based compact signatures over NTRU
FIPS	Federal Information Processing Standards Publication
GeMSS	a Great Multivariate Signature Scheme
G-LWR	General Learning with Rounding
IFP	Integer Factorization Problem
I-MLWE	Integer Module LWE
IND-CCA	Indistinguishability under Chosen-Ciphertext Attack
IND-CPA	Indistinguishability under Chosen-Plaintext Attack
IoT	Internet of Things
ISD	Information Set Decoding
KAT	Known Answer Test
KDF	Key Derivation Function
KEM	Key Encapsulation Module
LDPC	Lattice-Based Parity Check
LPN	Learning Parity with Noise
LRPC	Low Rank Parity Check
LUOV	Lifted Unbalanced Oil and Vinegar
LWE	Learning With Errors
LWR	Learning With Rounding
MI	Matsumoto and Imai

MIT	Massachusetts Institute of Technology
M-LWE	Modular LWE
MSS	Merkle Signature Scheme
MPKC	Multivariate Public-Key Cryptography
MQE	Multivariable Quadratic Equations
MQPKC	Multivariate quadratic public-key Cryptosystem
NIST	National Institute of Standards and Technology
NP-hard	Non-Deterministic Polynomial-Time Hard
NTNU	Norwegian University of Science and Technology
OTS	One-Time Signature Scheme
OW-CPA	One-Wayness Against Chosen-Plaintext Attack
OWF	One-Way Function
PSIS	Polynomial SIS
PKC	Public-Key Cryptography/Cryptosystem
PKE	Public-Key Encryption
P-LWE	Polynomial LWE
PoSSo	Polynomial System Solving
PQC	Post-Quantum Cryptography
pqRSA	Post-Quantum RSA
QC	Quasi-Cyclic
QD	Quasi-Dyatic
QC-MDPC	QC Moderate Density Parity Check
QC-LDPC	QC Low-Density Parity Check
R-LWE	Ring LWE
RM	Reed-Muller
RSA	Rivest-Shamir–Adleman
SBP	Szepieniec Beullens Preneel
SD	Syndrome Decoding
SIDH	Supersingular Isogeny Diffie-Hellman
SIS	Small Integer Solution
SVP	Shortest Vector Problem
SP	Special Publication
UOV	Unbalanced Oil and Vinegar
XMSS	eXtended Merkle Signature Scheme

Chapter 1

Introduction

This chapter introduces the motivation behind the cryptographic developments discussed in the paper, as well as the scope and limitations. It also contains a reading guide for the thesis as a whole.

1.1 History and Motivation

Since the computer was invented, the technology used to construct and advance it has been improving at an extremely rapid pace. This increase in processing power has naturally also increased the capability of breaking cryptographic algorithms and has thus prompted a push for development within this field, to ensure secure communication and storage. With the new advancements in the field of quantum computers, we face not only another giant leap towards even greater technological achievements, but also the challenges that follow.

One of the newest, and also greatest technological leaps within the world of computer science is undeniably quantum computers. The successful construction of such computers will certainly alter the field profoundly. While there is no way of knowing exactly when a full-scale quantum computer will be built, there are naturally several interest groups competing to achieve this goal. IBM are currently developing commercially available quantum computers in their *IBM Q* project [1], and are already offering their users the opportunity to run tests on their five quantum bit processor through their project *IBM Quantum Experience* (See section: 2.2) [2]. Google has announced that they are on track towards achieving their goal of *quantum supremacy* [3] [4], which they say postulate is achieved *"...- when a formal computational task is performed with an existing quantum device which cannot be performed using any known algorithm running on an existing classical supercomputer in a reasonable amount of time."* [5]. This hastened development has prompted a growing academic interest for public-key cryptographic algorithms which can withstand attacks by a quantum computer, also known as "quantum-resistant cryptography" or "post-quantum cryptography".

This interest is not unwarranted, as a successful implementation of quantum computers will have an enormous impact on digital security. A stable quantum computer with enough processing power will effectively break any public key algorithm which utilises the factorisation of large integers as the basis for its security. This is due to the fact that while there is no known classical algorithm which can solve these problems in polynomial time, there is one which can do so using quantum computers. This algorithm is known as Shor's algorithm and was invented by MIT professor Peter W. Shor in 1994 [6]. Shor's algorithm utilises quantum computing to compare the phases of prime numbers represented as sine waves to factorise great integers, effectively solving the discrete logarithm problem on which many modern cryptographic algorithms are based [7].

Many of the most widely used public key algorithms to date base themselves upon many of the same problems, with three categories encompassing a majority of our most used cryptographic algorithms. These three categories of problems are known as the integer factorisation problem, the discrete logarithm problem, and the newest of them, the elliptic curve discrete algorithm problem. All of these three categories will be broken by Shor's algorithm with a quantum computer. This is far more than an inconvenience, seeing as these algorithms are used extensively for ensuring secure transfer of sensitive data over the internet and creating digital signatures, as well as securing other connections over insecure networks [8].

From previous experience, we know that the efforts needed to fully replace a well-established cryptographic algorithm are long and laborious, both when it comes to development, testing, standardisation, and distribution. All of this needs to be put into place not only before a full-scale quantum computer is built, but preferably significantly longer before. This is to ensure that sensitive or secret information which is now encrypted with today's non-quantum resistant algorithms will no longer be sensitive or desirable when this encryption is no longer secure.

This can be more clearly expressed using a theorem presented by Michele Mosca, often referred to as Mosca's theorem [9]. The theorem states that given inequality 1.1, where x is the duration for which any encryption is expected/needs to be secure, y is the amount of time needed to re-tool the existing infrastructure with large-scale quantum-secure solutions, and z is the time until a large-scale, stable quantum computer is created, sensitive information encrypted with non-quantum resistant algorithms will be exposed during the time $t_{exposed}$ given in equation 1.2. In other words, this theorem is a formalisation of the reasoning that we need quantum-resistant cryptography standardised long before the quantum-computer is produced, such that all information encrypted with older, non-quantum resistant algorithms, is no longer relevant.

$$x + y > z \tag{1.1}$$

$$t_{exposed} = (x + y) - z \tag{1.2}$$

With this incentive, the United States' National Institute of Standards and Technology (NIST) declared that they were accepting public submissions for quantum-resistant asymmetric cryptographic algorithms [10]. The deadline for the submissions was 30^{th} of November 2017.

This master's thesis will take a closer look at the submissions received for the post-quantum cryptography standardisation competition issued by NIST. It will also explain how quantum computers break many traditional asymmetrical algorithms, just which algorithms these are, and why this is important to confront. This will be explained in chapter 2. The submission implementations' statistics such as key lengths, signature lengths, ciphertext lengths will be presented in chapter 3, as well as running times for the different phases of the submissions.

To investigate these points and aspects around them, the following research questions have been constructed:

Question 1:	What is the motivation for development of quantum resistant cryptography?
Question 2:	What are the current approaches to creating quantum resistant cryptographic algorithms?
Question 3:	What cryptographic algorithms have been developed and submitted to NIST as quantum resistant, and how do they compare?

1.2 Scope and Limitations

This master thesis is weighted as a full semester (30 points) at the Norwegian University of Science and Technology. A pre-project was done on the same topic weighted as a quarter of a semester (7.5 points) during the previous semester. Both the pre-project and the master thesis were written by a single author.

Due to the nature of post-quantum cryptography research and testing, this thesis is constructed as a general guide into the subject as well as a study of the cryptographic submissions given to NIST and their characteristics. Time constraints are a limiting factor, and thus the study has been limited to this topic only. This means that further study of quantum computing and quantum physics are out of scope for this study.

1.3 Reading Guide

The thesis is structured sequentially as follows:

- **Chapter 1: Introduction**
  The motivation behind new developments, goals, research questions, scope, limitations, and reading guide.

- **Chapter 2: Background and Theory**
  Necessary history, theory, and mathematical concepts for understanding further chapters.

- **Chapter 3: Submitted Algorithms**
  Basic information and any relevant discussions for all non-withdrawn submissions for all categories.

- **Chapter 4: Comparative Analysis**
  Comparative tables containing execution times and space requirements for all non-withdrawn submissions.

- **Chapter 5: Results and Discussion**
  Brief discussions on the PQC standardisation process, the algorithms submitted, their types, and a tentative ranking of some of the most promising submissions as of June, 2018.

- **Chapter 6: Conclusion**
  A concise summation of the thesis' different parts, their contents, and their intents, as well as some short concluding remarks on the continuation of this development.

- **Appendices**
  Scripts referenced in Chapter 3.

- **Bibliography**
  All sources used in the thesis.

Chapter 2

Background and Theory

To grasp how and why it is vital that new quantum resistant algorithms are produced, one must first comprehend the nature of quantum computing, the broken algorithms in question, and most importantly, the algorithm which breaks them. This chapter will briefly explain these points, to further the understanding of quantum computing and its potential consequences on security in today's digital world. It will also provide some basic guidelines for understanding cryptography in general. In addition to this, some essential information about the NIST post-quantum standardisation competition and its criteria and security categories is presented. Lastly, information on the families of cryptography which are potentially quantum-resistant, their origins, and their mathematical construction is presented.

2.1 Cryptography

This section contains an elemental description of cryptography and its functions. If the reader is already familiar with this, this section can be skipped without loss of continuation.

Secret communication between two parties is a venerable craft. Encoding messages, storing secret information, and ensuring that whoever saw any of this would not be able to read it has been an interest which has spanned thousands of years. Replacing letters in a text to make it unreadable ciphertext, exchanging secret keys with which one could decode secrets with, and many other techniques have been employed by man throughout the ages. In modern times, this field has grown exponentially more intricate and advanced, utilising far more developed mathematical theory to ensure secure transfer, storage, and authentication.

Modern cryptography as we know it was first introduced in the 19^{th} century, spurred on by world war II and the development of the computer. This field encompasses more than just encryption, and has many goals and properties. The most integral objectives of modern cryptography are given below.

- **Authentication:** Verification of the identity of sender and/or receiver
- **Authorisation:** Confirmation of the authority of sender and/or receiver

- **Confidentiality:** Ensuring that access to information is strictly for those authorised
- **Integrity:** Information can not be altered in any way without detection
- **Non-repudiation:** Previous messages or actions can not be denied at a later stage

In addition to having numerous goals, modern cryptography also employs many different techniques to achieve them. Goals aside, the field can be split into two main categories, namely symmetric and asymmetric cryptography.

Any cryptographic algorithms which utilises the same key for both encryption and decryption falls under the category of *symmetric cryptography.* This kind of cryptography is known to be fast and secure, but seeing as they utilise a shared secret key they are difficult to establish securely, especially when using computers, where a secure channel is often not established before communication ensues.

Differing from symmetric cryptography is *asymmetric cryptography*, also known as *public key cryptography*, which uses different keys for encryption and decryption. To achieve this, both a public key and a private key are used. The public key is used to encrypt plaintext intended for the owner of the corresponding private key. Only this private key can decrypt what has been encrypted with the public key, and thus the plaintext is secret for anyone other than the owner of this key. This is a much younger field, but it is a fundamental building block for today's most prominent cryptosystems. These kinds of algorithms are often much more cumbersome in their key management requirements, as well as encryption and decryption computations, but they are superior at establishing secure contact when no secure channel exists. This is why this type of cryptography is absolutely imperative for secure transfers and communications across the internet.

2.2 Quantum Computers

Almost every computer in the world today use bits as their most fundamental unit of information. The bit is a binary data unit, and thus has a binary value, either 1 or 0, true or false, yes or no [11]. This is the rudimentary idea behind the modern digital computer, but there are many ways to physically represent a bit. Any physical manifestation which has two mutually exclusive states can be used, some of the simpler representations being an on/off switch or a hole in a punch card. However, these are not especially efficient methods of storing or processing data, and as a result it is much more common to use variations in electrical voltages, a pulse of light, or a stored magnetic flux [12].

Differing from this mindset are the principles of quantum computers. In 1982, Richard P. Feynman published a paper called "Simulating Physics with Computers", suggesting how quantum computing could be used to vastly surpass the processing power of today's leading computers [13]. At the time, this was a theoretical idea, and it took more than 30 years before it was possible to start building such computers.

The basic idea behind a quantum computer is to replace binary digits with quantum bits, or qubits for short. As opposed to binary bits, qubits can exist in additional states in between the two binary states. This is defined as a superposition of the digital states [14]. In other words, the state of a qubit can be described by formula 2.1, where α and β are the probability amplitudes for the states 0 and 1, respectively. The fact that a quantum computer can contain numerous such states concurrently, ensures its potential dominance over traditional computers.

$$\alpha|0\rangle + \beta|1\rangle \tag{2.1}$$

It follows that α and β must satisfy the constraints given in equation 2.2, ensuring a collected probability of 1 [15].

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.2}$$

Despite this inherent superiority of a quantum computer, there are many demanding problems which arise when trying to compute using quantum physics. Controlling these subatomic particles is a strenuous task, and reading them is no simpler. Reading these values without either changing their value or only seeing them as zero or one is difficult at best [16]. While many new and promising ways of controlling qubits are being researched, this is out of scope for this paper and will therefore not be explained in any further detail.

2.3 Affected cryptographic algorithm families

To further understand why the introduction of functioning quantum computers will have such a serious impact on today's cryptography, it is vital to look into the most relevant cryptographic algorithms in question. To better describe how and why the algorithms can be broken, a short explanation of their key generation, and any encryption and decryption will be included.

It is important to note that the most essential hard mathematical problems used in today's cryptography are *discrete logarithm*, *integer factorisation*, and *elliptic curve discrete logarithm*. There are a vast number of algorithms which are based on these problems. This is why only some of the most well known and most frequently used algorithms will be explained, to supplement the understanding of further theory. The Sections 2.3.1, 2.3.2, and 2.3.3 can be skipped without loss of continuity, if their contents are familiar to the reader.

2.3.1 Discrete Logarithm Problem

Perhaps one of the most famous hard problems within the cryptographic field, the discrete logarithm problem (DLP) is the basis for an extensive number of the most used cryptographic algorithms today. Given a prime p and its multiplicative group G with a generator g, the discrete logarithm for a number x with respect to the given parameters is the integer k which solves equation 2.3 [6].

$$g^x \;\equiv\; x \; (mod\; p) \qquad where \quad 0 \leq k \leq p-1 \tag{2.3}$$

This problem is possible to solve for some special cases, but is still recognised as being computationally hard. Until now there has been no proven, practical, and efficient method for solving this problem with a non-quantum approach.

2.3.1.1 Diffie-Hellman

The Diffie-Hellman key generation algorithm was published in 1976 by Martin Hellman and Whitfield Diffie [17], and utilises the hardness of DLP. This algorithm allows two parties with no prior shared secret to generate a shared, secret key, which can not be seen by observing the exchange. The algorithm can also be used with more than one party, but this is not relevant to this example.

The basic idea is quite simple: Two large prime numbers p and q are generated. Both Alice and Bob choose a secret integer a and b, and calculate their respective A and B as seen in equations 2.4 and 2.5.

$$A = p^a \ mod \ p \tag{2.4}$$

$$B = p^b \ mod \ p \tag{2.5}$$

This is sent to the other party in plain text. Bob uses his secret integer b, to calculate equation 2.6, while Alice does the same using B and a.

$$A^b \ mod \ p = (p^a)^b \ mod \ p \tag{2.6}$$

Bob and Alice now share a secret s, as given in equation 2.7.

$$(p^a)^b \ mod \ p = (p^b)^a \ mod \ p = s \tag{2.7}$$

To calculate the secrets a and b using the information sent between Alice and Bob, without previous information of either of them, requires the ability to calculate discrete logarithms.

2.3.2 Integer factorisation Problem

While being constructed using quite basic mathematics, the integer factorisation problem (IFP) has no known efficient solution using traditional computers. Integer factorisation in itself is the practice of reducing a number into the integers of which it is a product. Performing this action until all numbers are prime numbers is known as prime factorisation. As with most problems, this is easily solvable when reduced sufficiently in size, but is widely known to be computationally hard for numbers of any substantial size. It is also important to remember that semiprimes, which are the product of two primes p and q, are known to be the hardest of all IFP, and are therefore often used in cryptography. When these two primes are sufficiently large and randomly chosen, even the best algorithms for traditional computers today fail to solve them efficiently.

2.3.2.1 Rivest–Shamir–Adleman

The RSA algorithm is one of the most popular public-key algorithms today. It was first published by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977 at the Massachusetts Institute of Technology. While not a revolutionary idea, it implemented a one-way function which was exceedingly difficult to invert. The RSA algorithm is one of many algorithms using factorisation of large integers to remain secure [18].

The fundamental idea behind RSA is that it makes it possible to exchange encrypted information without first exchanging a shared key. It is therefore often used as a way to exchange keys for symmetric algorithms, which are invariably faster. It is also used extensively for digital signatures, using the signatory's private key to encrypt a hash, which can subsequently be verified by anyone in possession of the corresponding public key.

To generate a set of public and private keys, two significantly large prime numbers p and q are multiplied together to create a product n. The totient is then found using any totient function, such as the one shown in equation 2.8.

$$\phi(n) = (p-1)\cdot(q-1) \tag{2.8}$$

Another number e, which is relatively prime to $\phi(n)$, is also chosen. This e and t are then used to find a number d, which must be such that equation 2.9 if fulfilled, where Z is an integer.

$$\frac{(d\cdot e)-1}{\phi(n)} \;=\; Z \tag{2.9}$$

This means that equation 2.10 is true.

$$d \equiv e^{-1}\ mod\ \phi(n) \tag{2.10}$$

The public key is then n and e, while the private key is n and d.

Any message m can be encrypted into a ciphertext c using the freely distributed public key, by solving equation 2.11. This ciphertext can only be encrypted using the private key, by solving equation 2.12.

$$c \equiv m^e\ (mod\ n) \tag{2.11}$$

$$c^d \equiv (m^e)^d \equiv m\ (mod\ n) \tag{2.12}$$

2.3.3 Elliptic Curve Discrete Logarithm Problem

The idea behind the elliptic curve discrete logarithm problem (ECDLP) is to find the discrete logarithm of an elliptic curve element with a non-secret base point [19]. An elliptic

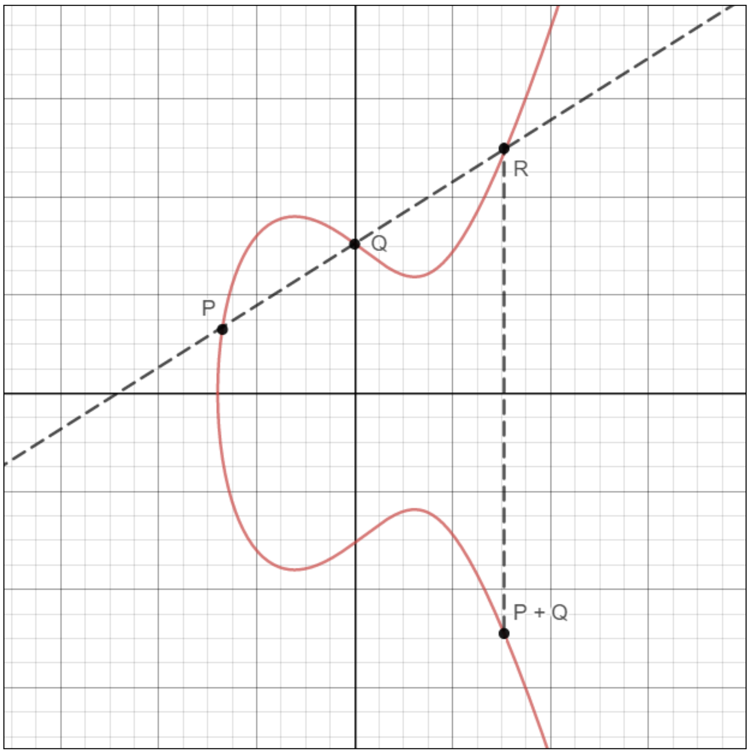

Figure 2.1: An illustration of an elliptic curve (red).

curve E is an algebraic curve over a finite field, defined as shown in equation 2.13. An example of such a curve can be seen in Figure 2.1.

$$y^2 = x^3 + ax + b \tag{2.13}$$

This elliptic curve E must be chosen carefully, and it must be cyclic, meaning that for any point P on the curve will form a cyclic group. A cyclic group will have a primitive element, also known as a generator G, such that adding this element to itself repeatedly will eventually yield the entire group. Using this property, a discrete logarithm problem will be created. Despite knowing the starting point P and the endpoint Q for such a group, knowing how many times d to which P was added to itself is hard. This is what is known as the elliptic curve discrete logarithm problem [20].

All elliptic curve cryptography (ECC) relies on the presumed hardness of the ECDLP. By replacing the multiplicative group of a finite field used in older PKC systems with one created using an elliptic curve defined over the same finite field, ECC can provide equivalent security to algorithms using the discrete logarithm problem [21]. It is also worth noting that ECC algorithms use relatively shorter keys compared to most non-ECC algorithms.

Despite these advantages, and the apparent security of the ECC schemes, this also entails that the algorithms utilising ECC will also be subject to many of the same weaknesses as algorithms relying on the discrete logarithm problem in itself.

2.3.3.1 Elliptic Curve Diffie-Hellman

Elliptic Curve Diffie-Hellman (ECDH) is, as the name implies, a version of the Diffie-Hellman key agreement which utilises Elliptic-Curve Cryptography (ECC) to calculate its secret keys.

With ECDH, Alice and Bob both agree on an elliptic curve E, most often a standardised curve. It follows that each party uses the given curve E to generate a private key and a public key. Alice's private and public keys are shown in equation 2.14 and 2.15 respectively [22]. Here, G is the generator of the curve E, while $\#E$ denotes the group cardinality of E, i.e, the number of points on the given curve E. Once these keys are calculated for both Alice and Bob, the algorithm continues as a standard Diffie-Hellman algorithm (See Section 2.3.1.1).

$$K_{prA} = d_A \in \{2, \#E\} \tag{2.14}$$

$$K_{puA} = d_A \cdot G \tag{2.15}$$

2.4 Shor's Algorithm

The algorithm at the heart of the post-quantum cryptography question is known as Shor's algorithm. As previously mentioned, it was discovered in 1994 by Peter Williston Shor, Professor at the Massachusetts Institute of Technology [6].

This algorithm uses quantum computers to efficiently solve two hard problems: factoring large integers (See Section 2.3.2) and finding discrete logarithms(See Section 2.3.1) using a quantum computer. In doing so, it also solves any problems which base themselves upon these two hard problems, such as the elliptic curve discrete logarithm problem (See Section 2.3.3). The idea behind Shor's algorithm is to utilise quantum computing to compare the phases of prime numbers as sinus waves to factorise great integers [7]. Peter Shor himself explained how this works, by comparing it to shining lights onto a diffraction grating to get a pattern [23]. Using number theory, the problem of number factorisation can be converted into a search for the period of a really long sequence, or rather, the length at which a sequence repeats itself. Then, just as with light diffraction, this periodic pattern is run through a quantum computer which functions as a computational interferometer, creating an interference pattern. This will output the period, which can be processed using a classical computer, to factorise the number.

The reason why this works is that instead of finding a number, we are aiming towards finding a period, which is a global property rather than a singular point. While this is by no means easier if we were to use a traditional computer, a quantum computer can solve this efficiently. By using the qubits (See Section 2.2), we can create an extensive superposition across factors from the period, which can be obtained using a traditional computer [24]. To do this, we must find a nontrivial factor of the number which is to be factorised, N.

This factor is then used in the calculations which are done on the quantum computer. While quantum physics is no easy thing, the most essential part of these calculations is the quantum Fourier transform, or the QFT. The QFT maps two vectors of complex numbers to each other, effectively mapping a periodic sequence to its period [24].

2.5 Post-Quantum Cryptography standardisation

When the NIST post-quantum competition submissions were all in on the 30$^{\text{th}}$ of November 2017, the process of finding the leading candidates commenced. While many post-quantum cryptosystems have been suggested, these cryptosystems need to undergo scrutinising research, testing, and evaluation, to ensure the most secure and reliable outcome.

In advance of the submissions being received, NIST has already specified that they wish to replace quantum resistant counterparts to many of their previously established standards, such as their key establishment algorithms and digital signature schemes. Any potential new standards which are agreed upon will be published as Special Publications (SP) or Federal Information Processing Standards (FIPS), respectively [25].

All submitted algorithms are evaluated, by both a selection panel of NIST employees, as well as other members of the scientific community. NIST themselves state that *"Although NIST will be performing its own analyses of the submitted algorithms, NIST strongly encourages public evaluation and publication of the results."* [26]. It is worth noting that the complexity of Post-Quantum Cryptography (PQC) standardisation is generally believed to be significantly more complex than standardisation for classical computers, seeing as not only are the requirements far more intricate, but the understanding of quantum computers and their potential capabilities is far from extensive [25].

The evaluation criteria will now be explained briefly in the three following subsections.

2.5.1 Security

The first, and undeniably the most principal evaluation criteria is naturally the security of the cryptographic scheme [27]. There are several factors within this category, after which the panel will judge a submission.

- Applications of Public-Key Cryptography
- Security Definition for Encryption/Key-Establishment
- Security Definition for Ephemeral-Only Encryption/Key-Establishment
- Security Definition for Digital Signatures
- Security Strength Categories
- Additional Security Properties
- Other Consideration Factors

The security will also be categorised according to different levels. The levels are defined by NIST, and are as follows.

I Security is comparable to or greater than a block cipher with a 128-bit key against an exhaustive key search (e.g. AES128)

II Security is comparable to or greater than a 256-bit hash function against a collision search (e.g. SHA256)

III Security is comparable to or greater than a block cipher with a 192-bit key against an exhaustive key search (e.g. AES192)

IV Security is comparable to or greater than a 384-bit hash function against a collision search (e.g. SHA384)

V Security is comparable to or greater than a block cipher with a 256-bit key against an exhaustive key search (e.g. AES256)

2.5.2 Cost

The second criteria is the cost of the cryptosystem. NIST has expressed their wish for public opinion and input with regards to the performance of the algorithm, as well as the needs of different systems. There may very well be a need for standardising more than one algorithm due to the vastly different use cases and performance needs, and NIST recognises this [28]. In this category, as well as the previous, there are many different subcategories.

- Public Key, Ciphertext, and Signature Size
- Computational Efficiency of Public and Private Key Operations
- Computational Efficiency of Key Generation
- Decryption Failures

2.5.3 Algorithm and Implementation Characteristics

The final criteria is the algorithm itself, ie. its implementation and characteristics. It is important to evaluate this as well as security and cost, since the modifiability, simplicity, and implementation possibilities of an algorithm can greatly affect its possible usefulness [29]. There are three subcategory evaluation criteria in this category.

- Flexibility
- Simplicity
- Adoption

2.6 Possible Replacement Algorithm Types

The submissions sent to NIST are all of varying types, with different characteristics and techniques being used. It is therefore vital that these basic types are understood by the reader, to comprehend the full picture painted by the data presented further on. The following subsections will go into general detail about each of these varieties of algorithms, as well as a more mathematical understanding of the underlying problem used to create such algorithms.

2.6.1 Lattice-Based Cryptography

Lattice-based algorithms were pioneered by Miklós Ajtai in 1996, with the idea that secure cryptographic algorithms could be constructed based on a hard lattice problem [30]. A lattice-based public-key encryption scheme was presented in the same year [31], but a scheme which was adequately and provably secure was not presented until 2005, when Oded Regev presented his scheme. This scheme utilises both lattices as well as a generalisation of the parity learning problem [32].

A lattice is a set of points with a periodic structure, given in n-dimensional space, and is used in a variety of fields. Lattice-based cryptographic algorithms are often based on either closest vector problem (CVP) or the shortest vector problem (SVP). These are further explained in section 2.6.1.1. The cryptographic constructors used in most lattice-based cryptographic algorithms are quite time-effective and simple, while still possessing security proofs based on worst-case hardness [33]. A number of the elementary problems used in this type of cryptographic algorithms also seem to be quantum resistant, as they are not reliant on any of the hard problems solved by Shor's algorithm [34]. This results in lattice-based cryptography being one of a few kinds of algorithms which are believed to hold promise as possible candidates for post-quantum cryptography.

2.6.1.1 General Mathematical Construction

Most relevant for cryptographic algorithms based upon lattices, is a lattice Λ in $\mathbb{R}^n$, which will have the form given in equation 2.16. The basis of the lattice Λ is a set of n-linearly independent vectors as given in equation 2.17, from which the lattice Λ is formed [33] [32].

$$\Lambda \; = \; \left\{ \sum_{i=1}^{n} a_i v_i \;\middle|\; a_i \in \mathbb{Z} \right\} \tag{2.16}$$

$$\{v_1, \; ..., \; v_n\} \tag{2.17}$$

Lattice-based cryptography is naturally dependent on lattices, or more specifically, the presumed hardness of different lattice problems. Lattice problems are a class of problems which consist of different optimisation problems performed on lattices. Most common of these are the algorithms mentioned in Section 2.6.1, namely CVP and SVP.

The Shortest Vector Problem (SVP) consists of finding the shortest nonzero vector for the lattice Λ, of a norm N in the vector space V within Λ [35]. This problem has been shown to be non-deterministic polynomial-time hard (NP-hard) for randomised reductions in lattices with L_2 norm [36]. An illustration of this problem can be seen in Figure 2.2.

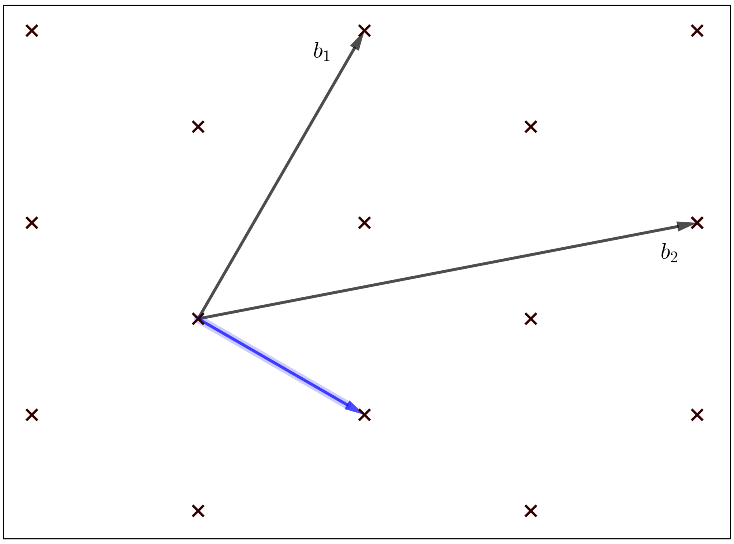

Figure 2.2: An illustration of the SVP problem with basis vectors b_1, b_2, and the shortest vector in blue.

The Closest Vector Problem (CVP) is a generalisation of SVP, and consists of finding the vector within the lattice Λ for a norm N which is closest to the target vector v. While any norm N can be used to define CVP, the Euclidean norm is common [37]. CVP has been shown to be NP-hard, as the SVP cannot be harder than CVP, and thus any hardness assumed for SVP implies at minimum the same hardness for CVP [38]. An illustration of this problem can be seen in Figure 2.3.

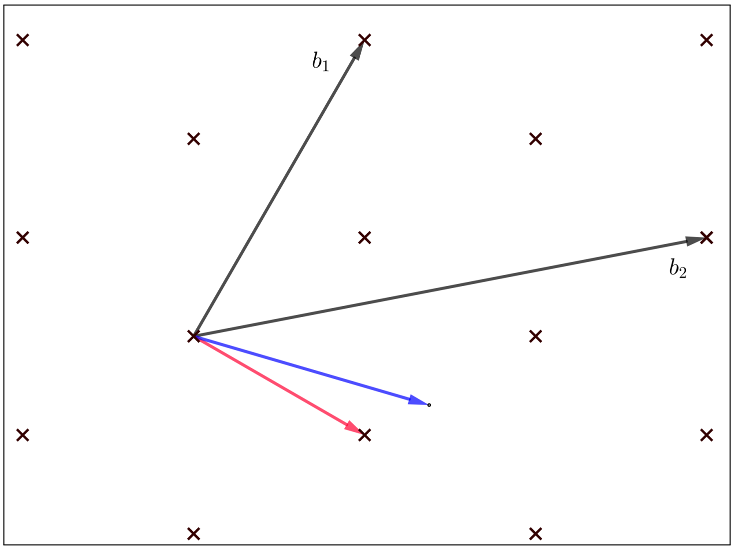

Figure 2.3: An illustration of the CVP problem with basis vectors b_1, b_2, the target v in blue and the closest vector in red.

2.6.2 Code-Based Cryptography

In 1978, Robert McEliece published his proposition for a code-based cryptosystem [39]. McEliece created a public-key encryption scheme which, with a few adjustments, seems to be secure against both classical computers as well as quantum computers. The scheme uses a one-way binary Goppa code as a private key and a randomly permuted version of this code as a public key. Secure encryption is then achieved by including errors in the ciphertext which only those in possession of the private key can revert [40] [41].

Using this method of encryption ensures high speed and low complexity during encryption and decryption, but it does have the disadvantage of requiring a substantial amount of memory, due to its large public key [40]. This is a weakness which is in no way exclusive to McEliece's public-key encryption algorithm, and is shared among most code-based encryption schemes.

All code-based cryptosystems utilise an error correcting code C in their basic algorithmic primitive [40]. This is the code from which they are named after. Many different types of error correction codes are used, and any code-based cryptosystem relies on the hardness of decoding it [42].

2.6.2.1 General Mathematical Construction

Seeing as there are a number of possible error correcting codes which can be utilised, McEliece will be used to mathematically demonstrate the idea behind code-based cryptosystems.

McEliece relies on general Goppa codes and general linear codes, and the fact that a known, efficient decoding algorithm exists for the former, but not the latter. The fact that decoding a randomly generated general linear code is a computationally difficult problem was shown by E. Berlekamp, R. McEliece, and H. van Tilborg in their 1978 publication "On the Inherent Intractability of Certain Coding Problems" [43]. This article shows that given a matrix H and a vector s, finding the minimum-weight solution to equation 2.18 will take at least exponential time. More specifically, given a parity-check matrix as given in formula 2.19, a syndrome as seen in formula 2.20, and a weight as given in formula 2.21, finding e as given in formula 2.22 such that equation 2.23 is fulfilled, is conjectured to be NP-difficult. A graphical representation of this problem can be seen in Figure 2.4, and is known as the syndrome decoding problem.

$$ZH = s \tag{2.18}$$

$$H \in \mathbb{F}_2^{n\times(n-k)} \tag{2.19}$$

$$s \in \mathbb{F}_2^{(n-k)} \tag{2.20}$$

$$w \in \mathbb{Z} \tag{2.21}$$

$$e \in \mathbb{F}_2^n \qquad of \quad weight \quad w_H(e) \leq w \tag{2.22}$$

$$eH^T = s \tag{2.23}$$

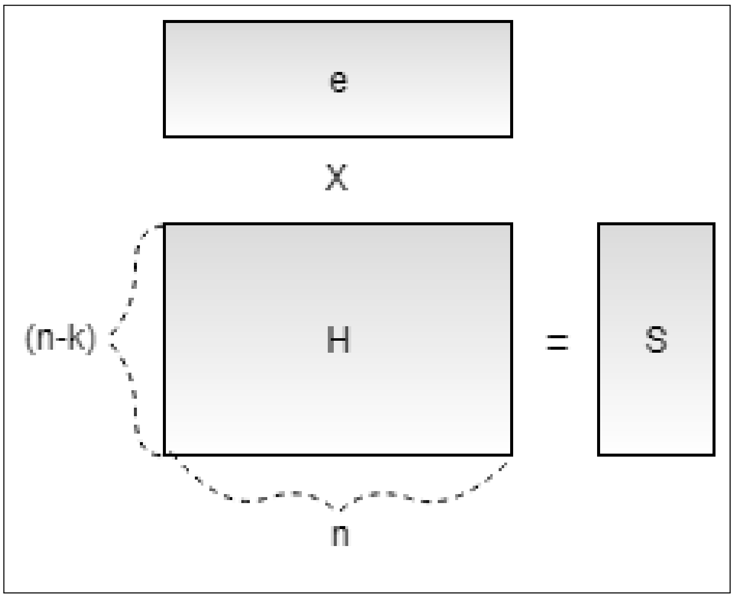

Figure 2.4: A graphical representation of the syndrome decoding problem

The McEliece PKC utilises a public generator matrix G_p as shown in equation 2.24, the number of possible errors t corrected by the code used, and a private key consisting of S, G_s, and P.

$$G_p \;=\; SG_sP \qquad where \quad G_p \in \mathbb{F}_2^{k \times n} \tag{2.24}$$

Here, S is a random invertible matrix as given in equation 2.25, G_s is the generator matrix for the secret code, and P is a $n \times n$ random matrix.

$$T \in \; \mathbb{F}_2^{k \times k} \tag{2.25}$$

After encoding a plaintext $p \in \mathbb{F}_2^k$ using the private key and a random vector $z \in \mathbb{F}^n$ of weight t as shown in equation 2.26, only the corresponding secret error correction code C can recover the error vector e, thus recovering the plaintext p.

$$c \;=\; pG_p \oplus z \tag{2.26}$$

The decoding is shown first in equation 2.27, before the code word is recovered using the secret generator matrix G_s. The plaintext p is then recovered as shown in equation 2.29 2.28 [39] [40].

$$cP^{-1} \;=\; (pS)G_p \oplus zP^{-1} \tag{2.27}$$

$$pSG = G_s(G \oplus zP^{-1}) \tag{2.28}$$

$$p = (pSG)_J(G_{\cdot J})^{-1}S^{-1} \qquad where \quad J \subseteq \{1, ..., n\} \tag{2.29}$$

A graphical representation of how this error coding and its subsequent trapdoor is used during encryption and decryption can be seen in Figure 2.5.

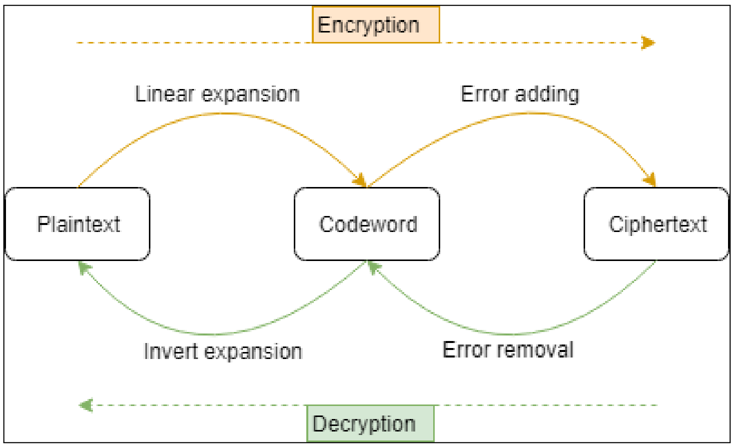

Figure 2.5: An illustration of the encryption and decryption process of the McEliece encryption scheme, with encryption in orange and decryption in green.

2.6.3 Multivariate Cryptography

The first scheme to introduce the idea behind multivariate was called C*, and was first presented by Tsutomu Matsumoto and Hideki Imai in 1998 [44]. While this scheme has since been broken, many cryptographic schemes have been made using the same fundamental idea, as the design proved to be especially efficient and potentially quite usable in practical situations [45].

All Multivariate Public-Key Cryptosystems (MPKC) use the same basic design, as they all rely on the use of multivariate polynomials over a finite field. In most cases, the polynomial equations are of degree two, resulting in multivariate quadratic polynomials, the solving of which is still credited to being NP-hard. [46].

As opposed to many other types of PKC, the MQPKC cannot be solved any more rapidly using Shor's algorithm than with a classical computer, as it does not rely on any of the hard problems which Shor's algorithms can solve. It is therefore a possible candidate category for a quantum resistant encryption scheme [46].

2.6.3.1 General Mathematical Construction

As mentioned in Section 2.6.3, all MPKC rely on the use of multivariate polynomials over a finite field as their trapdoor function. These are polynomials which contain more than one indeterminate, or in other words, have more than one variable. Most commonly, the polynomials used are of degree two, and are thus called multivariate quadratics [45]. All quadratic polynomials are of the form shown in equation 2.30.

$$f(x) = ax^2 + bx + c \tag{2.30}$$

A general setup for any multivariate PKC will have a private key consisting of two affine maps S and T, which are maps between two affine spaces which preserves collinearity and ratios of distances within the two spaces. It also consists of a quadratic map $\mathcal{Q}$, and a public key, which is a set of quadratic polynomials as shown in equation 2.31, where all p_i are non-linear polynomials within $\mathbf{w}$, as given in equation 2.32. Quadratic maps are always of the form given in equation 2.33.

$$\mathcal{P} \;=\; (p_1(w_1, ..., w_n), ..., p_m(w_1, ..., w_n)) \tag{2.31}$$

$$p_k(\mathbf{w}) \;:=\; \sum_i P_{ik} w_i \;+\; \sum_i Q_{ik} w_i^2 \;+\; \sum_{i\,j} R_{ijk} w_i w_j \tag{2.32}$$

$$x_{n+1} = f_\mu(x_n) = \mu x_n(1 - x_n) \tag{2.33}$$

Inverting such a set of equations over a finite field is the problem upon which the hardness of all MPKC are based, and is as the equivalent of solving the set. This problem is known to be NP-hard.

2.6.4 Hash-Based Signatures

Hash-based signatures were first introduced by Ralph Merkle, in the form of his Merkle Signature Scheme (MSS) [47]. Many other hash-based signature schemes have since been introduced, such as the eXtended Merkle Signature Scheme (XMSS) [48] and SPHINCS [49] [50].

Unlike many other digital signature schemes, hash-based signatures do not rely on the factorisation problem, but rather the use of hash trees and one-time signatures. As a consequence of this, hash-based signatures are not subject to the same weaknesses and breaks as many other signature schemes, and could therefore possibly be quantum-resistant [51].

2.6.4.1 General Mathematical Construction

Hash-based signatures vary widely in mathematical construction, mainly with the changing of the underlying secure cryptographic hash function, and are therefore a quite diverse category of signatures. These hash functions project a value from one set with potentially infinite members, to a value from a set with a fixed number of members, typically fewer than the first set.

It is important that these functions are not only irreversible (or computationally hard to reverse), but also that they fulfil three security requirements. They must be both pre-image resistant (also known as "one-way") and second-preimage resistant. This means that given any output from the hash function, it should be hard to find the input used to create it, or a different input which results in the same output. In addition to this, the hash function must be collision resistant. This means that it should be hard to find any two input values which result in the same output value. These properties are mathematically expressed for a hash function H in equations 2.34, 2.35, and 2.36, respectively. It is also coveted that a minor change in the original message should produce such a difference in the resulting hash that the hashes of the two messages should appear unrelated.

$$Given \quad H(p) = c \quad \text{find} \quad p \tag{2.34}$$

$$Given \quad H(p) = H(p') = c \quad \text{find} \quad p' \tag{2.35}$$

$$Given \quad H(p_1) = H(p_2) \quad \text{find any two} \quad p_1 \text{ and } p_2 \tag{2.36}$$

As mentioned in Section 2.6.4, the first hash-based signature was the MSS, which utilises a Merkle tree. An illustration of a binary version of such a tree can be seen in Figure 2.6. Named after its creator, a Merkle tree is a hash-based data structure, where each node is a hashing of the concatenation of its parent nodes. All leaf nodes are hashes of a given block of data. While they are often implemented as binary trees, the same structure can be created with n children per node. In MSS, the leaf nodes are composed of the hashed values of a one-time signature scheme (OTS), while the public key for the scheme is defined as the root node. Thus, the OTS values can be authenticated using the shorter, public key, while the values themselves stay secret. The OTS key pairs are used in the order of the leaves from left to right, to prevent reuse. This results in the signature for message n being calculated as shown in equation 2.37, where σ_{OTS} is the signature on the message, p_{kOTSi} is the public key of the nth OTS, and AP_n is the authentication path of the nth leaf.

$$\Sigma \;=\; (n, \sigma_{OTS}, p_{kOTSi}, AP_n) \tag{2.37}$$

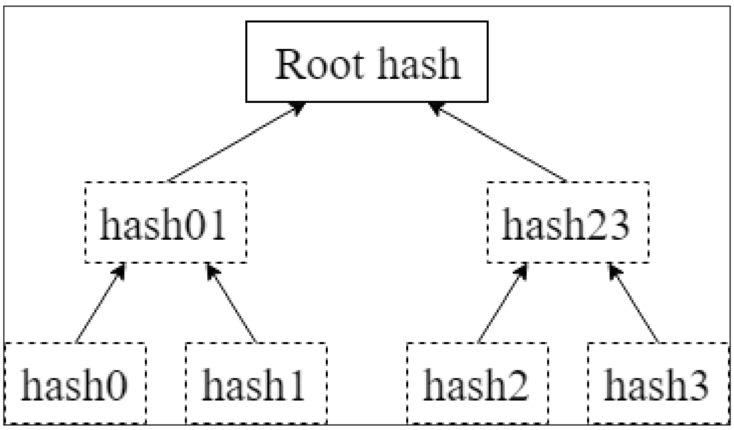

Figure 2.6: An illustration of a binary Merkle tree.

2.6.5 Isogeny-Based Cryptography

As one of the youngest of the main cryptographic categories in this competition, isogeny-based cryptography's ideas are based upon those of elliptic curve cryptography (ECC). Isogeny-based cryptography was brought into the light of interest in the early 2010's, when it became clear that quantum-resistant cryptographic algorithms would soon have to be made a reality [52]. This means that all newer isogeny-based cryptographic algorithms were made with the intention of being secure against quantum computers and Shor's algorithm.

2.6.5.1 General Mathematical Construction

The definition of an isogeny between two elliptic curves E_1 and E_2 is a morphism which maps the infinity point O of each curve to the infinity point of the other, as given in equation 2.38.

$$\phi : E_1 \longrightarrow E_2 \quad satisfying \quad \phi(O) = O \tag{2.38}$$

This mapping will, for all except for the zero isogeny defined in equation 2.39, be a finite map of curves.

$$[0](P) = O \qquad for\ all\ P \in E_1 \tag{2.39}$$

A classic illustration of a single elliptic curve can be seen in Figure 2.1, but for the purposes of illustrating an isogeny between the two, a different graphical representation can be utilised, which is shown in Figure 2.7 [52]. In this illustration, the elliptic curve $\mathbb{C}/\Lambda$ expressed as a disjoint union between two linearly independent complex numbers $\omega_1 \in \mathbb{C}$ and $\omega_2 \in \mathbb{C}$, as shown in equation 2.40.

$$\Lambda = w_1\mathbb{Z} \oplus w_2\mathbb{Z} \tag{2.40}$$

If there is a ℓ-torsion point $p \in \mathbb{C}/\Lambda_1$, where the ℓ-torsion subgroup is constructed of $(\frac{i\omega_1}{\ell}, \frac{j\omega_2}{\ell})$, and there is an elliptic curve Λ_2 which is defined as shown in the equation 2.41, then Λ_1 is a subset of Λ_2, and we can define an isogeny ϕ of Λ_1 and Λ_2 as shown in equation 2.42. Creating dual isogenies can be done by using a point q not in the kernel of the original isogeny, creating a new degree ℓ cover. This dual isogeny $\phi\ o\ \hat{\phi}$ is then of degree ℓ^2.

$$\Lambda_2 = p\mathbb{Z} \oplus \Lambda_1 \tag{2.41}$$

$$\phi : \mathbb{C}/\Lambda_1 \to \mathbb{C}/\Lambda_2 \tag{2.42}$$

The rudimentary idea behind isogeny-based cryptography is to utilise the properties of such isogenies to create a hard problem, which can be used to encrypt data.

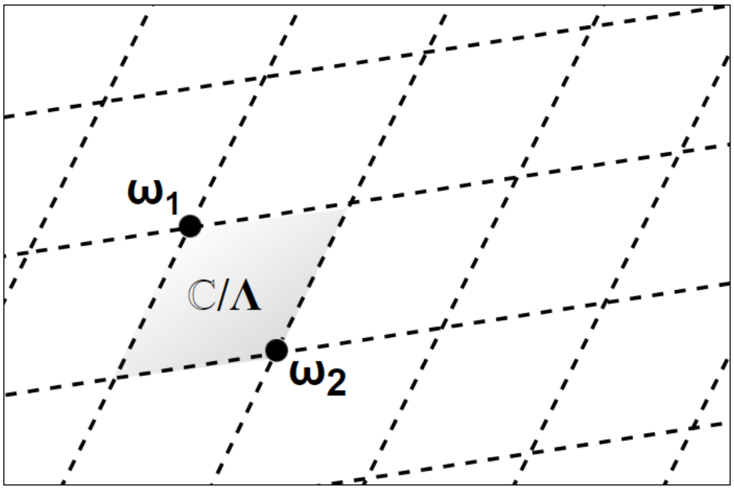

Figure 2.7: An illustration of an elliptic curve $\mathbb{C}/\Lambda$ expressed as a disjoint union between two linearly independent complex numbers $\omega_1 \in \mathbb{C}$ and $\omega_2 \in \mathbb{C}$.

Chapter 3

Submitted Algorithms

During round 1 of submissions for the NIST PQC standardisation, there were a total of 69 submissions received within the deadline, 5 of which have since been withdrawn. Of these algorithms, there were 26 lattice-based, 20 code-based, 9 multivariate, and 3 hash-based, with 8 algorithms falling outside any of these categories.

In this Chapter, all of the submitted and non-withdrawn algorithms will be briefly presented, as well as any attacks presented against them. Zipped files for all the presented algorithms can be found in NIST's round 1 submissions' overview [53].

3.1 Encryption Schemes

3.1.1 Compact LWE

Compact LWE is a lattice-based encryption algorithm submitted by Dongxi Liu, Nan Li, Jongkil Kim, and Surya Nepa. This algorithm is designed to be used in resource-constrained devices, specifically mentioning IoT-devices as a possible use case. Basing itself upon the Learning With Secretly Scaled Errors in Dense Lattices Problem, often referred to as the Compact-Learning With Errors (LWE) problem, this algorithm attempts to achieve efficient levelled authentication [54].

The idea behind the algorithm is to use considerably smaller dimensions than other lattice-based schemes, down to a dimension parameter of 13, while retaining the required level of security. To verify the security of the algorithm, the creators chose to reduce the LWE problem to a Compact-LWE. In doing so, the authors also claimed *"-...even if the closest vector problem (CVP) in lattices can be solved, Compact-LWE is still hard, due to the high density of lattices constructed from Compact-LWE samples and the relatively longer error vectors."*. This has later been put into question by several other publications [54].

A cryptoanalysis was performed by Jonathan Bootle and Mehdi Tibouchi, and was released in August 2017. In this report, it is shown that a plaintext-recovery attack can be performed on ciphertexts encrypted using the algorithm in question using only the public key. An algorithm for performing this decryption attack was published in the same note [55].

An extension of this attack was discovered by Jonathan Bootle, Mehdi Tibouchi, and

Keita Xagawa, and a script which performs this attack was made public in early January of 2018 [56].

Cryptoanalysis of this algorithm was also performed and published by Haoyu Li, Renzhang Liu, Yanbin Pan, and Tianyuan Xie, where it was demonstrated that the proposed lattice-scheme was not secure, due to its small parameter size. Using a lattice basis reduction algorithm, CVP is solved efficiently. The paper also disputes the Compact-LWE algorithm authors' claims of Compact-LWE being secure even if CVP can be efficiently solved. The properties exploited in this attack were considered in the original publication of the algorithm, but it was falsely assumed that approximation methods were not viable [57].

Despite modifications made to the algorithm to avoid the discovered plaintext recovery attacks, which includes avoiding direct use of exploitable variables during construction of the public key. This modified version of the algorithm was also attacked, using the same base idea as the previous attacks mentioned [58].

3.1.2 EMBLEM and R.EMBLEM

EMBLEM and R.EMBLEM are lattice-based encryption algorithms, submitted by Minhye Seo, Jong Hawn Park, Dong Hoon Lee, Suhri Kim, and Seung-Joon Lee.

3.1.3 Giophantus

Giophantus is a lattice-based public-key encryption algorithm, submitted by Koichiro Akiyama, Yasuhiro Goto, Shinya Okumura, Tsuyoshi Takagi, Koji Nuida, Goichiro Hanaoka, Hideo Shimizu, and Yasuhiko Ikematsu. Designed specifically to counter the threat towards lattice-based algorithms posed by approximation attacks, this algorithm is presumed secure by indistinguishability under chosen-plaintext attack (IND-CPA) [59]. This was presumed in the algorithm's publication because there does not exist any efficient algorithm which could find the smallest solution in a non-linear solution space of multivariate indeterminate equations, upon which this algorithm bases itself [60].

The presumption that the Giophantus algorithm was secure by IND-CPA was later put into question by Wouter Castryck and Frederik Vercauteren in early January of 2018 in official comments to the submission via NIST's official comments. An attack which exploited the algorithm's chosen base ring, a ring homomorphism to Z_q, was presented. This attack could, however, be prevented by changing to another base ring R_q [61].

3.1.4 Guess Again

"Guess Again" is a Random Walk encryption algorithm, submitted by Vladimir Shpilrain, Mariya Bessonov, Alexey Gribov, and Dima Grigoriev.

An attack against this algorithm was presented by Lorenz Panny [62]. The attack based itself upon certain statistical properties in the ciphertext and therefore used neither the public or the private key. The specific script presented did assume the contents of the given directory's KAT archive, it is modifiable [63].

3.1.5 LEDApkc

LEDApkc is a code-based public-key cryptosystem (PKC), submitted by Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. This module relies on the use of Quasi-Cyclic Low-Density Parity Check (QC-LDPC) codes and is at its core built from the cryptosystem McEliece. This team submitted both an encryption system and a key encapsulation algorithm (See Section 3.2.13) [64].

The cryptosystem is designed to provide compact key pairs and quick decoding, with the innovations presented by the creators of the cryptosystem as follows.

A reaction attack against the type of cryptosystem LEDApkc is has been presented and documented by Tomáš Fabšič, Viliam Hromada, and Pavol Zajac. In this attack, it was suggested that using decoding failure rate (DFR) analysis could render useful information about the secret masking matrix Q, which again could be used to create a set of candidates for Q which is small enough to further create a set of candidates for the generator matrix of the secret Low-Density Parity Check (LDPC) code. Applying Stern's algorithm [65] to this will then recover the secret matrix H, and through this, the private key [66].

This attack and its efficiency was called into question by the creators of LEDApkc, due to two different assumptions about the attacked algorithm: The size of n_0 being 2, and the fact that the attack has modified certain system parameters which has artificially increased the DFR. According to the creators of the algorithm, the number of candidates for G will increase according to equation 3.1, which ruins the attack's efficiency for all other sizes for n_0. [67].

$$2^{n_0^2} p^{n_0^2} \tag{3.1}$$

3.1.6 McNie

McNie is a code-based McEliece Niederreiter encryption cryptosystem, submitted by Lucky Galvez, Jon-Lark Kim, Myeong Jae Kim, Young-Sik Kim, and Nari Lee. The idea behind this algorithm is to have smaller key sizes as compared to RSA, by utilising the quasi-cyclicity of matrices. Despite this compact key size, documentation for the submitted algorithm claimed equal or better security than McEliece [68].

An attack on this algorithm was presented by Philippe Gaborit [69]. By using an Information Set Decoding (ISD) adapted for rank metric [70], this attack can be used to reduce the security of the algorithm by a factor of two. The authors have recognised this attack, and have modified the parameters for the McNie algorithm to increase the security of the algorithm [69].

3.1.7 Odd Manhattan

Odd Manhatten is a lattice-based encryption scheme submitted by Thomas Plantard. The algorithm claims CCA security by transforming a chosen-plaintext attack (CPA) resistant algorithm using methods proposed by Dent [71] [72].

An attack against this algorithm was presented by Tancrede Lepoint in the official NIST comments. During decapsulation, if the algorithm fails during re-encryption, the return flag is set to -1, while the shared secret key is not modified. Running a CCA attack, discarding the return flag, it is possible to guess the secret key using what is in the shared secret key [73]. A script for this specific attack was also presented, which can be found in appendix A.1.

Lepoint also noted that this attack is avoidable, by changing the actions taken during failure, such that the shared secret is not set without a successful verification process. Le Trieu Phong provided such a patch for the code [73].

3.1.8 Post-Quantum RSA Encryption

Post-Quantum RSA (pqRSA) Encryption is an encryption algorithm submitted by Daniel J. Bernstein, Josh Fried, Nadia Heninger, Paul Lou, and Luke Valenta. The pqRSA team submitted a signature, key encapsulation, and encryption. The algorithm is a version of the original RSA algorithm which utilises extremely large keys to counter Shor's algorithm. For the computations to be possible for the average user, the algorithm uses relatively small secret primes, as well as encryption exponents and decryption exponents. The pqRSA team submitted a signature (See Section 3.4.13) and an encryption algorithm [74].

The authors themselves describe the algorithm as an easy option for current RSA users who need to become quantum resistant. They believe that this will inevitably happen, and thus it is of high importance to figure out if this is a secure solution.

3.2 KEM

3.2.1 BIG QUAKE

BIG QUAKE is a code-based KEM, submitted by Alain Couvreur, Magali Bardet, Élise Barelli, Olivier Blazy, Rodolfo Canto-Torres, Philippe Gaborit, Ayoub Otmani, Nicolas Sendrier, and Jean-Pierre Tillich [75]. The scheme utilises based upon quasi-cyclic Goppa codes, instead of traditional binary Goppa codes, with the goal of achieving lower complexity, by trading off an affordable security loss in comparison with the original McEliece system [39].

3.2.2 BIKE

BIKE is a code-based KEM, submitted by Nicolas Aragon, Paulo S. L. M. Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Güneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, and Gilles Zémor [76]. The KEM is based on the use of Quasi-Cyclic Moderate Density Parity Checks (QC-MDPC), which can be decoded by use of bit flipping techniques.

Three variations of this scheme were submitted, named BIKE-1, BIKE-2, and BIKE-3. The first variant is focused on fast key generation and uses a variation of McEliece to obtain this. This version does not require polynomial inversion. The second variant uses a systematic parity check and follows Niederreiter's framework. This version is made to be compact in size, but uses polynomial inversion, and may therefore be notably slower than other variants. The third builds the work presented with Ouroboros [77], and the final variant is in many ways like the first, with the exception of the decapsulation method used invoking decoding on a noisy key.

The fact that the ciphertexts produced by BIKE-1 and BIKE-2 were not indistinguishable from random data was brought to attention by Danilo Gligoroski and was discussed as a possible weakness. Questions regarding the security proof (Theorem 2 and 3 in [76]) were noted by Ray Perlner [78].

3.2.3 CFPKM

CFPKM is a multivariate quadratic KEM, submitted by O. Chakraborty, J.-C. Faugére, and L. Perret [79]. The scheme is based on the hardness of solving a system of noise non-linear polynomials, known as the PoSSo with Noise. Two different version of the KEM were submitted, CFPKM128 and CFPKM182.

A function breaking the IND-CPA security of both versions of the KEM was presented by Ron Steinfeld. This function efficiently decrypts the private key $k_{private}$, given the ciphertext c and the public key k_{public} [80]. The attack decryption function script can be seen in Appendix A.3.

Another attack was presented by Fernando Viridia and Martin R. Albrecht, which recovers the higher order bits of k_b. This script can be seen in Appendix A.4.

3.2.4 Classic McEliece

Classic McEliece is a code-based KEM, submitted by Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang [81]. This submission is heavily based on the original McEliece cryptosystem [39], and is made specifically for achieving IND-CCA2 level security against classical as well as quantum computers. This is done by creating the KEM using Niederreiter's dual version of McElice which uses binary Goppa codes [82].

3.2.5 CRYSTALS-KYBER

CRYSTALS-KYBER is a lattice-based KEM, submitted by Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé [83]. The algorithm is created to be IND-CCA2-secure and bases itself upon the hardness of the LWE problem over module lattices [84].

3.2.6 DAGS

DAGS is a code-based KEM, submitted by Gustavo Banegas, Paolo S. L. M. Barreto, Brice Odilon Boidje, Pierre-Louis Cayrel, Gilbert Ndollane Dione, Kris Gaj, Cheikh Thiecoumba Gueye, Richard Haeussler, Jean Belo Klamti, Ousmane N'diaye, Duc Tri Nguyen, Edoardo Persichetti, and Jefferson E. Ricardini [85]. The KEM uses quasi-dyadic (QD) generalised Srivastava codes and aims to achieve an IND-CCA level of security [86].

An issue with the KEM's shared key entropy length was pointed out by Daniel Smith-Tone. This was agreed to be a minor, and easily correctable issue [87].

3.2.7 Ding Key Exchange

Dig Key Exchange is a lattice-based KEM, submitted by Jintai Ding, Tsuyoshi Takagi, Xinwei Gao, and Yuntao Wang [88]. The KEM relies on the hardness of the R-LWE problem and is presented as a possible direct replacement for the non-quantum resistant Diffie-Hellman key exchange.

3.2.8 DME - KEM

The DME cryptosystem signature is a public-key signature scheme based on double exponentiation, submitted by Ignacio Luengo, Martin Acendaño and Michel Marco. The DME cryptosystem is composed of both a signature (See Section 3.4.2) and a KEM system [89].

3.2.9 FrodoKEM

FrodoKEM is a lattice-based KEM, submitted by Michael Naehrig, Erdem Alkim, Joppe Bos, Leo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila [90]. The KEM is based upon the hardness of the LWE problem, as well as algebraically unstructured lattices.

There are two proposed sizes submitted for this scheme, FrodoKEM-640 and FrodoKEM976, targeting subsequently level 1 and level 3 security as described by NIST (See Section 2.5.1). Two variants of each of these schemes are also presented, using different methods (AES-128 and cSHAKE) to generate a pseudo-random matrix [91].

3.2.10 HILA5

HILA5 is a lattice-based KEM, submitted by Markku-Juhani O. Saarinen [92]. The KEM uses R-LWE, but with a new reconciliation method which has shown to have a lower failure rate than previously used methods.

Daniel J. Bernstein, Leon Groot Bruinderink, Tanja Lange, and Lorenz Panny published a key-reuse key-recovery attack on this KEM, showing that its claims of IND-CCA security were not correct [93]. It is noted that this does not break the IND-CPA security of the scheme.

3.2.11 HQC

Hamming Quasi-Cyclic (HQC) is a code-based KEM, submitted by Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor [94]. As the name suggests, the scheme makes use of quasi-cyclic codes as well as Bose–Chaudhuri–Hocquenghem (BCH) codes. The scheme claims IND-CPA level security.

It was pointed out by Zhen Liu and Yanbin Pan that the assumptions made about the hardness of Decisional Quasi-Cyclic Syndrome Decoding (DQCSD) problem were not true for the given case [95].

3.2.12 LAKE

LAKE is a code-based KEM, submitted by Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zemor [96]. This team also submitted the KEM LOCKER, See Section 3.2.14, and is very similar to that submission. The scheme uses Ideal Low Rank Parity Check (LRPC) codes, and claims IND-CPA security, and have three different variants for security categories 1, 3 and 5 (See Section 2.5.1).

A problem with the ciphertexts' lack of random distribution was pointed out by Danilo Gligoroski. This problem applied to both LAKE and LOCKER. A fix for this problem was created by the submitting team, without further impact on the submitted KEM. [97]

3.2.13 LEDAkem

LEDAkem is a code-based key encapsulation module (KEM), submitted by Marco Baldi, Alessandro Barenghi, Franco Chiaraluce, Gerardo Pelosi, and Paolo Santini. This module relies on the use of Quasi-Cyclic Low-Density Parity Check (QC-LDPC) codes and is at its core built from the cryptosystem Niederreiter [98]. This team submitted both an encryption system (See Section 3.1.5) and a key encapsulation algorithm [64].

An apparent flaw with this KEM was pointed out by Keita Xagawa, who stated that due to several factors, this algorithm fails to achieve sufficient security against chosen-ciphertext attacks (CCA) [99]. The team proposed to achieve this level of security by applying a hybrid construction [100] based on the Nieddereiter framework [98] to a deterministic Public-Key Encryption (PKE) which is secure against One-Wayness Against Chosen-Plaintext Attacks (OW-CPA). Xagawa cited several CCA which would not be noticed using this method, primarily Appendix K in [101] and [102].

The authors of the algorithm confirmed that the algorithm itself only provided IND-CPA security, but could be modified using a KDF to provide IND-CCA security. The team acknowledged that to achieve this IND-CCA security, the modifications needed further additions of secret bitstrings as input to the key derivation function (KDF). They also reiterated that this flaw found by Xagawa did not affect the algorithm's IND-CPA security, which is what was required by NIST [99].

3.2.14 LOCKER

LOCKER is a code-based KEM, submitted by Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zemor [103]. This team also submitted the KEM LAKE, see Section 3.2.12, as is very similar to that submission. The scheme uses Ideal LRPC codes, and claims IND-CCA2 security, and have three different variants for security categories 1, 3 and 5 (See Section 2.5.1).

A problem with the ciphertexts' lack of random distribution was pointed out by Danilo Gligoroski. This problem applied to both LAKE and LOCKER. A fix for this problem was created by the submitting team, without further impact on the submitted KEM. [104]

3.2.15 Mersenne-756839

Mersenne-756839 is a lattice-based KEM, submitted by Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Mikos Santha [105]. This scheme uses Mersenne numbers, which are of the form $p = 2^n - 1$, i.e. prime numbers which are one less than a power of two. The KEM relies on the hardness of the calculation of the arithmetic modulo of these numbers.

3.2.16 NewHope

NewHope is a lattice-based KEM, submitted by Thomas Pöppelmann, Erdem Alkim, Roberto Avanzi, Joppe, Léo Ducas, Antonio de la Piedra, Peter Schwabe, and Douglas Stebila [106]. The scheme is based on the R-LWE problem, and there are four proposed versions. NewHope512-CPA-KEM and NewHope1024-CPA-KEM respectively target level 1 and 5 security, and claim IND-CPA level security. NewHope512-CCA-KEM and NewHope1024-CCA-KEM respectively target level 1 and 5 security, and claim IND-CCA level security [107].

3.2.17 NTRU Prime

NTRU Prime is a lattice-based KEM, submitted by Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal [108]. The scheme has two proposed mechanisms for key encapsulation, NTRU LPrime and Streamlined NTRU Prime. The latter of these is optimised implementation-wise, while the first trades some of this type of optimisation for security. Both mechanisms are created for providing IND-CCA2 level security [109].

3.2.18 NTRU-HRSS-KEM

NTRU-HRSS-KEM is a lattice-based KEM, submitted by John M. Schanck, Andreas Hülsing, Joost Rijneveld, and Peter Schwabe [110]. The scheme is based upon the OW-CPA secure PKE named NTRU-HRSS [111], and claims CCA2 level security.

3.2.19 NTS-KEM

NTS-KEM is a code-based KEM, submitted by Martin Albrecht, Carlos Cid, Kenneth G. Paterson, Cen Jung Tjhai, and Martin Tomlinson [112]. The scheme is described as a variant of the McEliece PKE scheme and claims IND-CCA level security. Goppa codes are also used for decapsulation, and the KEM itself aims towards compact size, making it suitable for low bandwidth communication.

3.2.20 Ouroboros-R

Ouroboros-R is a code-based KEM, submitted by Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor [113]. The submission utilises both LRPC codes and quasi-cyclic (QC) codes, and builds upon the idea of the original Ouroboros-scheme [77]. The scheme has several proposed parameter sets, 128-bits, 192-bits, and 256-bits, all aiming for security levels corresponding to level 1, 3, and 5 as specified by NIST(See Section 2.5.1), respectively.

3.2.21 QC-MDPC KEM

QC-MDPC KEM is a code-based KEM, submitted by Atsushi Yamada, Edward Eaton, Kassem Kalach, Philip Lafrance, and Alex Parent [114]. This scheme is based on the QC-MDPC McEliece encryption scheme and claims IND-CPA security. It is noted that this KEM is not subject to the same weakness that was found for QC-MDPC, in which decoded failures could be used to reconstruct the secret key, as it does not use static keys [115].

3.2.22 Ramstake

Ramstake is a lattice-based KEM, submitted by Alan Szepieniec [116]. The scheme has a high focus on the simplicity of the mechanism, and is therefore not optimised for speed or size, and thus, is most suitable for short messages.

It was noted by Jacob Alperin-Sheriff that it should be noted in the documentation that the most significant byte and least significant chunk should not be used. He also noted that "bad" cycles of the decapsulation decoding loop could render useful information for mounting a CCA attack. The submitter acknowledged these notes [117].

3.2.23 RLCE-KEM

RLCE-KEM is a code-based KEM, submitted by Yongge Wang [118]. The KEM is based upon the Random Linear Code Based PKE (RLCE) scheme, whose security is thought to be contingent on the hardness of decoding random linear codes. The scheme proposes several different implementations, namely RLCE_KEM-128A, RLCE_KEM-192A, RLCE_KEM-1256A, RLCE_KEM-128B, RLCE_KEM-192B, RLCE_KEM-1256B.

An attack which breaks all implementations denoted with A was presented by Alain Couvreur, Matthieu Lequesne, and Jean-Pierre Tillich. This attack does not affect the implementations denoted with B [119].

3.2.24 RQC

Rank Quasi-Cyclic (RQC) is a code-based KEM, submitted by Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor [120]. The scheme claims IND-CCA2 level security, and has proposed parameters to match NIST's security levels 1, 3, and 5.

3.2.25 SABER

SABER is a lattice-based KEM, submitted by Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren [121]. This submission is reliant on the hardness of the Module-Learning With Rounding (LWR) problem, halving the amount of randomness needed as compared to when using the LWE problem. In addition to this, all integers moduli are powers of two, simplifying the mechanism used. The submission claims IND-CCA level security.

3.2.26 SIKE

The Supersingular Isogeny Key Encapsulation (SIKE) is an isogeny-based KEM, submitted by David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik [122]. The KEM is based on Supersingular Isogeny Diffie-Hellman (SIDH), which in turn relies on the hardness of the Computational Supersingular Isogeny (CSSI) problem. This submission claims IND-CCA level security for all suggested implementations, SIKEp503, SIKEp751, and SIKEp964.

The quantum security of the CSSI problem, as well as other related problems, continues to be discussed within the community, and there are disagreements regarding which attacks are more efficient against it [123].

3.2.27 Three Bears

Three Bears is a lattice-based KEM, submitted by Mike Hamburg. The main goal with this scheme is to *"...- encourage exploration of potentially desirable, but less conventional designs."* [124]. Keeping with this spirit, Three Bears's private key k_p is a seed and it relies on Integer Module LWE (I-MLWE). The scheme claims IND-CCA level security.

3.3 KEM Encryption

3.3.1 KCL (OKCN/AKCN/CNKE)

Key Consensus from Lattice (KCL) [125] is a lattice-based KEM and encryption proposal, submitted by Yunlei Zhao, Zhengzhong Jin, Boru Gong, and Guangye Sui [126]. This proposal contains new authenticated key-establishment (AKE) and PKE schemes based on LWE and its subproblems, Ring LWE (R-LWE) and Modular LWE (M-LWE). All proposed AKE and PKE claim CCA level security.

3.3.2 KINDI

Key encapsulatIoN and encryption baseD on lattIces (KINDI) is a lattice-based KEM and encryption proposal, submitted by Rachid El Bansarkhani [127]. The encryption in this scheme relies on a trapdoor, resulting in secret vectors and error terms being available for inspection by recipients. The message is embedded into the latter terms. These choices result in a compact ciphertext, and thus higher message throughput [128].

3.3.3 LAC

LAC is a lattice-based KEM and encryption proposal, submitted by Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, and Zhenfei Zhang [129]. The submission contains four PKC primitives, all based on R-LWE. The primitives are comprised on an IND-CPA level secure PKE scheme and a key exchange protocol converted from this scheme, an IND-CCA level secure KEM, and an AKE protocol.

Questions regarding the submitting team's choice of BCH codes over binary Goppa codes as error correction codes were brought to question by Martin Tomilson during discussion of the submission. There were also discussions about whether or not the connection to the LWE problem's worst-case hardness problem was preserved in LAC as assumed in the submission [130].

3.3.4 Lepton

Lepton is a KEM and encryption proposal, submitted by Yu Yu and Jiang Zhang [131]. This submission is based on the hardness of the Learning Parity with Noise (LPN) problem, and can thus be categorised as both a lattice-based scheme and a code-based scheme.

3.3.5 LIMA

LattIce MAthematics (LIMA) is a lattice-based KEM and encryption proposal, submitted by Nigel P. Smart, Martin R. Albrecht, Yehuda Lindell, Emmanuela Orsini, Valery Osheter, Kenny Paterson, and Guy Peer [132]. The submission is based on R-LWE and claims IND-CCA security for both the presented PKE scheme and KEM [133].

3.3.6 Lizard

Lizard is a lattice-based KEM and encryption proposal, submitted by Jung Hee Cheon, Sangjoon Park, Joohee Lee, Duhyeong Kim, Yongsoo Song, Seungwan Hong, Dongwoo Kim, Jinsu Kim, Seong-Min Hong, Aaram Yun, Jeongsu Kim, Haeryong Park, Eunyoung Choi, Kimoon kim, Jun-Sub Kim, and Jieun Lee [134].

3.3.7 LOTUS

Learning with errOrs based encryption with chosen ciphertexT secUrity for poSt quantum era (LOTUS) is a lattice-based KEM and encryption proposal, submitted by Le Trieu Phong, Takuya Hayashi, Yoshinori Aono, and Shiho Moriai [135]. The submission relies on the hardness of the LWE problem and claims IND-CCA2 level security with parameters as low as 256-bit [136].

It was noted by Tancrede Lepoint that the shared secret used in the LOTUS KEM is not modified after failure, an is thus to be considered as a weakness. An attack exploiting this weakness was also presented. The submitting team released a patch for this exploit, mitigating the weakness [137].

3.3.8 NTRUEncrypt

NTRUEncrypt is a lattice-based KEM and encryption proposal, submitted by Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, and William Whyte [138]. The cryptosystem contains two PKE algorithms and two KEM algorithms. The first PKE is based upon the original NTRU, while the second is based upon the proveable secure NTRU, which achieves CCA-2 security [139]. Both KEMs are based upon each of the different PKE schemes.

3.3.9 Round 2

Round 2 is a lattice-based KEM and encryption proposal, submitted by Oscar Garcia-Morchon, Zhenfei Zhang, Sauvik Bhattacharya, Ronald Rietman Ludo Tolhuizen, and Jose-Luis Torre-Arce [140]. Both the submitted PKE and KEM rely on the hardness of the General Learning with Rounding (GLWR) problem.

3.3.10 Titanium

Titanium is a lattice-based KEM and encryption proposal, submitted by Ron Steinfeld, Amin Sakzad, and Raymond K. Zhao [141]. This submission contains both a KEM and a PKE cryptosystem, claiming IND-CCA2 and IND-CPA level security, respectively [142]. Titanium uses the polynomial variants of the Small Integer Solution (SIS) and LWE problems, namely Polynomial SIS (PSIS) and Polynomial LWE (P-LWE).

3.4 Signature Schemes

3.4.1 CRYSTALS-DILITHIUM

CRYSTALS-DILITHIUM is lattice-based signature module submitted by Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. The developing team focused on several main points when creating the signature scheme: implementation simplicity, parameter conservatism, modularity, as well as achieving the minimal size for the public key and signature [143].

The algorithm is made to be strongly secure against CPA, and it is based on the "Fiat-Shamir with Aborts" approach [144]. It utilises rejection sampling to make these schemes more secure, and more compact [143] [145].

3.4.2 DME - Signature

The DME cryptosystem signature is a public-key signature scheme based on double exponentiation, submitted by Ignacio Luengo, Martin Acendaño and Michel Marco. To ensure that the public key size is not too large, a small number of variables are used, as well as a special non-dense linear map at each end of the composition. The DME cryptosystem is composed of both a signature and a KEM system (See Section 3.2.8) [89].

It was pointed out by Ward Beullens that this system might not be reaching the level of security which was claimed in the documentation, due to the structure of the public map, and its possible exploitability. This proposed attack consists of two parts. The first step involves creating a polynomial map of degree 4 by representing the public key map over F_2, before decomposing this map into the composition of two quadratic polynomial maps [146]. The next step in the attack involves solving the isomorphism of the quadratic components to a known quadratic map [147]. By finding these isomorphisms, breaking the system is possible by inversion of the public key [148].

In response to this, Ingacio Luengo proposed a change in parameters for the system, which would prevent this exploitation by doubling the security bits. This change in parameters, however, only secured the system from the latter part of the attack, not the first [148].

3.4.3 DRS

DRS is a lattice-based signature scheme, submitted by Thomas Plantard, Arnaud Sipasseuth, Cédric Dumondelle, and Willy Susilo [149]. The scheme utilises diagonal dominant lattices and was inspired by GHH [150].

3.4.4 DualModeMS

DualModeMS is a multivariate-based signature scheme, submitted by J.-C. Faugeére, L. Perret, and J. Ryckeghem [151]. The scheme is complementary to the MeMSS submission.

In this scheme, the public key size is kept small, at the cost of the signature's size, which is larger. While this is not in keeping with traditional multivariate signature schemes, it has been done before in several signature schemes based on the Matsumoto and Imai (MI) [152]. Utilising a technique proposed by Szepieniec, Beullens, and Preneel (SBP) [153], MI-based multivariate signature schemes such as DualModeMS can be transformed into schemes with shorter public keys and longer signatures. Due to this technique and its flexibility, the authors also propose that this could be useful for other submissions of the same type.

3.4.5 FALCON

Fast Fourier lattice-based compact signatures over NTRU (FALCON) is a lattice-based signature scheme submitted by Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang [154]. This scheme aims towards solving the communication complexity problem, which is presumed to arise when switching to PQ signatures. Thus, FALCON is constructed for minimising the bit sizes of the public key and the signature.

The scheme itself is constructed using three components: the framework proposed by Gentry, Peikert and Vaikuntanathan [155], a class of cryptographic lattices, and a trapdoor sampler. For the latter two, FALCON uses NTRU lattices, and a newly developed technique known as fast Fourier sampling [156].

3.4.6 GeMSS

GeMSS, or a Great Multivariate Signature Scheme, is a multivariate-based quadratic signature scheme, submitted by A. Casanova, J.-C. Faugere, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem [157]. The scheme's primary goal is to ensure that signature sizes remain small and the verification process is fast, while still retaining the required level of security. In its essence, GeMSS is a variant of the multivariate scheme QUARTZ [158] which improves security and efficiency.

3.4.7 Gravity-SPHINCS

Gravity-SPHINCS is a hash-based signature scheme, submitted by Jean-Phillippe Aumasson and Guillaume Endignoux [159]. The scheme is a further developed variant of SPHINCS [50], modifying many procedures to ensure higher speeds and reduced key sizes. While the scheme's modifications do ensure high assurance, allow for speed and size trade-offs and batch signing, it comes with an increased complexity as well as larger signature sizes. This entails that the algorithm may not be suitable for systems which are dependent upon speed and low-complexity, such as real-time and IoT systems.

Supporting documentation for this submission includes a master's thesis written by Guillaume Endignoux [160].

3.4.8 Gui

Gui is a multivariate-based quadratic signature scheme, submitted by Jintai Ding, Ming-Shen Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang [161]. The scheme's goal is to decrease the HFE polynomial of the QUARTZ signature scheme upon which it is based [158], increase generation, which substantially increases the number of vinegar variables and minus equations. In doing so, security is weakened somewhat but is traded for the speed of the signature.

3.4.9 HiMQ-3

HiMQ-3 is a multivariate-based quadratic signature scheme, submitted by Kyung-Ah Shim, Cheol-Min Park, and Aeyoung Kim [162]. The scheme is designed for high speeds, relative to other multivariate-based signature schemes. Its general underlying problems are Extended Isomorphism of Polynomials (EIP), the MinRank Problem, and Polynomial System Solving (PoSSo) Problem.

3.4.10 LUOV

The Lifted Unbalanced Oil and Vinegar (LUOV) is a multivariate-based quadratic signature scheme, submitted by Ward Beullens, Bart Preneel, Alan Szepieniec, and Frederik Vercauteren [163]. The scheme is a modified version of one of the oldest multivariate signature schemes, namely the Unbalanced Oil and Vinegar (UOV) scheme from 1997 [164]. The modifications reduce the public key size, thus improving the scheme.

3.4.11 MQDSS

MDDSS is a multivariate-based quadratic signature scheme, submitted by Simona Samardjiska, Ming-Shing Chen, Andreas Hulsing, Joost Rijneveld, and Peter Schwabe [165].

3.4.12 Picnic

Picnic is a public key signature scheme, submitted by Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha [166]. The scheme is based upon a zero-knowledge proof [167] and symmetric key principles [168] [169].

3.4.13 Post-Quantum RSA Signature

Post-Quantum RSA (pqRSA) Signature is a signature scheme submitted by Daniel J. Bernstein, Josh Fried, Nadia Heninger, Paul Lou, and Luke Valenta. The pqRSA team submitted a signature and an encryption algorithm (See Section 3.1.8).

3.4.14 pqNTRUsign

pqNTRUsign is a modular lattice-based signature scheme, submitted by Cong Chen, Jeffrey Hoffstein, William Whyte, and Zhenfei Zhang [170] [171]. The algorithm can utilise NTRU lattices with both Gaussian and uniform samplers.

3.4.15 pqsigRM

pqsigRM is a punctured Reed-Muller (RM) code-based signature scheme, submitted by Wijik Lee, Young-Sik Kim, Yong-Woo Lee, and Jong-Seon No [172]. The algorithm improves upon the Courtois, Finiasz, and Sendrier (CFS) scheme's lack of existential forgeability under CMA as well as its parameter scaling [173] [174].

3.4.16 qTESLA

qTESLA is a lattice-based signature scheme, submitted by Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon [175]. The scheme is based on the decisional ring LWE (R-LWE) problem and is designed to both be conservative in memory use, while at the same time being secure. The algorithm targets three different NIST security levels using the key sizes, namely 128-bit, 192-bit, and 256-bit.

3.4.17 RaCoSS

RaCoSS is a code-based signature scheme, submitted by Kazuhide Fukushima, Partha Sarathi Roy, Rui Xu, Shinsaku Kiyomoto, Kirill Morozov, and Tsuyoshi Takagi [176].

During the first round of evaluations, Andreas Huelsing, Daniel J. Bernstein, Lorenz Panny, and Tanja Lange noted several problems with the scheme. Due to a misunderstanding of bits and bytes, only 12.5% of entries in c were compared, and thus almost any message would pass as valid given certain circumstances. They also noted memory leaks in two functions, as well as a weakness towards both random message attacks and chosen message attacks in the low-weight hash function. A specific attack against the scheme was also presented, which could sign any message for any k_{public}, without knowledge of the corresponding $k_{private}$. To remedy this, a change of parameter n and the weight of z was suggested [177].

These flaws and attacks were addressed by the submitting team, and a need to increase the size of suggested parameters was acknowledged.

3.4.18 Rainbow

Rainbow is a multivariate-quadratic based signature scheme, submitted by Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, and Bo-Yin Yang [178]. The scheme is a multilayer generalisation of the Oil-Vinegar construction, with the intention to improve the efficiency of the Unbalanced Oil-Vinegar construction [179].

3.4.19 SPHINCS+

SPHINCS+ is a hash based signature scheme, submitted by Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, and Peter Schwabe [180]. This submission is an improvement upon the SPHINCS algorithm and differentiates itself from the older algorithms on several points. Most notably, it provides multi-attack protection and verifiable index selection.

3.4.20 WalnutDSA

WalnutDSA is a signature scheme, submitted by Derek Atkins, Iris Anshel, Dorian Goldfeld, and Paul E. Gunnells [181]. The algorithm is based on the difficulty of the reversing E-Multiplication problem, a hard problem among braid groups [182].

A paper presenting an attack on an older version of WalnutDSA was published before the scheme was submitted to NIST [183]. The scheme was changed from using one braid as a $k_{private}$ to using two braids, to prevent this attack. After the submission, a workaround for this change was presented by Ward Beullens [184]. This workaround did, however, produce signatures which are significantly longer than legitimate signatures, exposing the attack. Due to this, the scheme is not considered broken by this attack alone.

The assumptions made in the security proof for the scheme were also put into question by Mr Beullens, and this was addressed by the WalnutDSA submitter team.

A weakness in the scheme towards square root attacks was presented by Simon Blackburn [184]. To mitigate this weakness, a parameter increase (from $N = 8$ to $N = 10$) was needed in the scheme.

Chapter 4

Comparative Analysis

This Chapter contains comparative Tables of all non-withdrawn submissions to the NIST Post Quantum Standardisation. These Tables are to aid with the analysis and evaluation of the given submissions and include public key lengths, private key lengths, ciphertext lengths, signature lengths, as well as execution times for all reference implementations and/or recommended implementations for all non-withdrawn submissions.

All numbers are retrieved from the original submission documentation and reference implementation for each subsequent submission, or the NIST submission forum. For references, see Chapter 3. The column after which every Table is sorted is highlighted. Any fields marked with a dash (-) are left blank for lack of data or viable values.

4.1 Space Requirements

This section contains public key lengths, private key length, ciphertext length, and signature lengths. Each of the subsections 4.1.1, 4.1.2, and 4.1.3 contains all Tables for space requirements for all encryption algorithms implementations, KEM algorithms implementations, and signature algorithm implementations, respectively.

4.1.1 Encryption Space Requirements

For all Tables containing space requirements for encryption implementations, *sec* denotes the claimed NIST level of security for the implementation (See Section 2.5.1), and $k_{private}$ (private key), k_{public} (public key), and c (ciphertext) entries all denote length in bytes. If left blank, the implementation does not fulfil any of the NIST security levels' requirements, or lacks information about the given property.

Table 4.1 contains all encryption algorithm implementations, and is sorted alphabetically after submission implementation names. This Table is meant as a referencing and look-up Table for all implementations. Following this Table, there are $3 \cdot 3$ Tables containing the same algorithm implementations, divided into the NIST security levels 1 and 2, 3 and 4, and 5. There are 3 different Tables for each level, each sorted after space requirements each of the previously mentioned property lengths. Below is an overview of the Tables for each level.

- Levels 1 and 2:
  - Table 4.2, sorted by private key length.
  - Table 4.3, sorted by public key length.
  - Table 4.4, sorted by ciphertext length.
- Levels 3 and 4:
  - Table 4.5, sorted by private key length.
  - Table 4.6, sorted by public key length.
  - Table 4.7, sorted by ciphertext length.
- Level 5:
  - Table 4.8, sorted by private key length.
  - Table 4.9, sorted by public key length.
  - Table 4.10, sorted by ciphertext length.

Table 4.1: Encryption implementation security levels, key lengths and ciphertext lengths, sorted alphabetically after submission implementation names.

Submission Implementation	Sec	$k_{private}$[B]	k_{public}[B]	c[B]
Compact LWE	3	232	2064	360
Giophantus 602	1	602	14412	28824
Giophantus 868	3	868	20796	41592
Giophantus 1134	5	1134	27204	54408
Guess Again	5	2000	4000	9216000
KCL AKCN-MLWE-CCA	4	1312	992	-
KINDI-256-3-4-2	2	1472	1184	1792
KINDI-256-5-2-2	5	1712	1456	2496
KINDI-512-2-2-2	4	2112	1728	2688
KINDI-512-2-4-1	4	2304	1984	2688
KINDI-512-3-2-1	5	2752	2368	3328
LAC-CPA-128	1	1056	544	1024
LAC-CPA-192	3	2080	1056	1536
LAC-CPA-256	5	2080	1056	2048
LEDApkc-1-2	1	668	3480	6960
LEDApkc-1-3	1	844	4688	7032
LEDApkc-1-4	1	1036	6408	8544
LEDApkc-3-2	3	972	7200	14400
LEDApkc-3-3	3	1196	10384	15576
LEDApkc-3-4	3	1364	13152	17536
LEDApkc-5-2	5	1244	12384	24768
LEDApkc-5-3	5	1548	18016	27024
LEDApkc-5-4	5	1772	22704	30272
LIMA-CCA-sp-1018	1	9163	6109	6105
LIMA-CCA-sp-1306	2	15673	10449	10443
LIMA-CCA-sp-1822	3	21865	14577	14555
LIMA-CCA-sp-2062	5	24745	16497	16475
LIMA-CCA-2p-1024	3	9217	6145	6147
LIMA-CCA-2p-2048	5	18433	12289	12291
LIMA-CPA-sp-1018	1	9163	6109	6105
LIMA-CPA-sp-1306	2	15673	10449	10443
LIMA-CPA-sp-1822	3	21865	14577	14555
LIMA-CPA-sp-2062	5	24745	16497	16475
LIMA-CPA-2p-1024	3	9217	6145	6147
LIMA-CPA-2p-2048	5	18433	12289	12291
Lizard-CATEGORY1-N536	1	137216	162016	1648
Lizard-CATEGORY1-N663	1	169728	1882112	983
Lizard-CATEGORY3-N816	3	313344	2457600	2496
Lizard-CATEGORY3-N925	3	365568	2736128	2768
Lizard-CATEGORY5-N1088	5	557056	6553600	3328
Lizard-CATEGORY5-N1300	5	665600	3710976	3752
RLizard-CATEGORY1	1	257	4096	2208
RLizard-CATEGORY3-N1024	3	513	4096	4272

RLizard-CATEGORY3-N2048	3	369	8192	8496
RLizard-CATEGORY5	5	513	8192	8512
LOTUS-128	1	714240	658944	1144
LOTUS-192	3	1126400	1025024	1456
LOTUS-256	5	1630720	1470976	1768
McNie-3Q-128-1	1	194	431	547
McNie-3Q-128-2	1	218	486	621
McNie-3Q-192-1	3	247	569	732
McNie-3Q-192-2	3	274	631	814
McNie-3Q-256-1	5	337	819	1065
McNie-3Q-256-2	5	348	829	1078
McNie-4Q-128-1	1	340	347	390
McNie-4Q-128-2	1	401	417	473
McNie-4Q-192-1	3	465	487	558
McNie-4Q-192-2	3	512	539	619
McNie-4Q-256-1	5	584	630	619
McNie-4Q-256-2	5	601	647	749
NTRUEncrypt-443	1	701	611	611
NTRUEncrypt-734	4	1173	1023	1023
NTRUEncrypt-1024	5	8194	4097	4097
Odd Manhatten-128	1	1627648	1626240	180224
Odd Manhatten-192	3	2565055	2563260	344640
Odd Manhatten-256	5	4456650	4454241	616704
PQRSA-ENCRYPT-15	-	98304	32768	32768
PQRSA-ENCRYPT-20	-	3145728	1048576	1048576
PQRSA-ENCRYPT-25	-	100663296	33554432	33554432
PQRSA-ENCRYPT-30	2	3221225472	1073741824	1073741824
SABER light	1	832	672	736
SABER	3	2304	1248	1088
SABER fire	5	1664	1312	1472
Titanium CPA toy	-	32	11552	2560
Titanium CPA lite	-	32	13088	2976
Titanium CPA standard	1	32	14720	3520
Titanium CPA medium	1	32	16448	4512
Titanium CPA high	3	32	17952	8320
Titanium CPA super	5	32	23552	8320

Table 4.2: NIST security category 1 and 2 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
Titanium CPA standard	1	32	14720	3520
Titanium CPA medium	1	32	16448	4512
McNie-3Q-128-1	1	194	431	547
McNie-3Q-128-2	1	218	486	621
RLizard-CATEGORY1	1	257	4096	2208
McNie-4Q-128-1	1	340	347	390
McNie-4Q-128-2	1	401	417	473
Giophantus 602	1	602	14412	28824
LEDApkc-1-2	1	668	3480	6960
NTRUEncrypt-443	1	701	611	611
SABER light	1	832	672	736
LEDApkc-1-3	1	844	4688	7032
LEDApkc-1-4	1	1036	6408	8544
LAC-CPA-128	1	1056	544	1024
KINDI-256-3-4-2	2	1472	1184	1792
LIMA-CCA-sp-1018	1	9163	6109	6105
LIMA-CPA-sp-1018	1	9163	6109	6105
LIMA-CCA-sp-1306	2	15673	10449	10443
LIMA-CPA-sp-1306	2	15673	10449	10443
Lizard-CATEGORY1-N536	1	137216	162016	1648
Lizard-CATEGORY1-N663	1	169728	1882112	983
LOTUS-128	1	714240	658944	1144
Odd Manhatten-128	1	1627648	1626240	180224
PQRSA-ENCRYPT-30	2	3221225472	1073741824	1073741824

Table 4.3: NIST security category 1 and 2 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
McNie-4Q-128-1	1	340	347	390
McNie-4Q-128-2	1	401	417	473
McNie-3Q-128-1	1	194	431	547
McNie-3Q-128-2	1	218	486	621
LAC-CPA-128	1	1056	544	1024
NTRUEncrypt-443	1	701	611	611
SABER light	1	832	672	736
KINDI-256-3-4-2	2	1472	1184	1792
LEDApkc-1-2	1	668	3480	6960
RLizard-CATEGORY1	1	257	4096	2208
LEDApkc-1-3	1	844	4688	7032
LIMA-CCA-sp-1018	1	9163	6109	6105
LIMA-CPA-sp-1018	1	9163	6109	6105
LEDApkc-1-4	1	1036	6408	8544
LIMA-CCA-sp-1306	2	15673	10449	10443
LIMA-CPA-sp-1306	2	15673	10449	10443
Giophantus 602	1	602	14412	28824
Titanium CPA standard	1	32	14720	3520
Titanium CPA medium	1	32	16448	4512
Lizard-CATEGORY1-N536	1	137216	162016	1648
LOTUS-128	1	714240	658944	1144
Odd Manhatten-128	1	1627648	1626240	180224
Lizard-CATEGORY1-N663	1	169728	1882112	983
PQRSA-ENCRYPT-30	2	3221225472	1073741824	1073741824

Table 4.4: NIST security category 1 and 2 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
McNie-4Q-128-1	1	340	347	390
McNie-4Q-128-2	1	401	417	473
McNie-3Q-128-1	1	194	431	547
NTRUEncrypt-443	1	701	611	611
McNie-3Q-128-2	1	218	486	621
SABER light	1	832	672	736
Lizard-CATEGORY1-N663	1	169728	1882112	983
LAC-CPA-128	1	1056	544	1024
LOTUS-128	1	714240	658944	1144
Lizard-CATEGORY1-N536	1	137216	162016	1648
KINDI-256-3-4-2	2	1472	1184	1792
RLizard-CATEGORY1	1	257	4096	2208
Titanium CPA standard	1	32	14720	3520
Titanium CPA medium	1	32	16448	4512
LIMA-CCA-sp-1018	1	9163	6109	6105
LIMA-CPA-sp-1018	1	9163	6109	6105
LEDApkc-1-2	1	668	3480	6960
LEDApkc-1-3	1	844	4688	7032
LEDApkc-1-4	1	1036	6408	8544
LIMA-CCA-sp-1306	2	15673	10449	10443
LIMA-CPA-sp-1306	2	15673	10449	10443
Giophantus 602	1	602	14412	28824
Odd Manhatten-128	1	1627648	1626240	180224
PQRSA-ENCRYPT-30	2	3221225472	1073741824	1073741824

Table 4.5: NIST security category 3 and 4 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
Titanium CPA high	3	32	17952	8320
Compact LWE	3	232	2064	360
McNie-3Q-192-1	3	247	569	732
McNie-3Q-192-2	3	274	631	814
RLizard-CATEGORY3-N2048	3	369	8192	8496
McNie-4Q-192-1	3	465	487	558
McNie-4Q-192-2	3	512	539	619
RLizard-CATEGORY3-N1024	3	513	4096	4272
Giophantus 868	3	868	20796	41592
LEDApkc-3-2	3	972	7200	14400
NTRUEncrypt-734	4	1173	1023	1023
LEDApkc-3-3	3	1196	10384	15576
SABER	3	2304	1248	1088
KCL AKCN-MLWE-CCA	4	1312	992	-
LEDApkc-3-4	3	1364	13152	17536
LAC-CPA-192	3	2080	1056	1536
KINDI-512-2-2-2	4	2112	1728	2688
KINDI-512-2-4-1	4	2304	1984	2688
KINDI-512-3-2-1	5	2752	2368	3328
LIMA-CCA-2p-1024	3	9217	6145	6147
LIMA-CPA-2p-1024	3	9217	6145	6147
LIMA-CCA-sp-1822	3	21865	14577	14555
LIMA-CPA-sp-1822	3	21865	14577	14555
Lizard-CATEGORY3-N816	3	313344	2457600	2496
Lizard-CATEGORY3-N925	3	365568	2736128	2768
LOTUS-192	3	1126400	1025024	1456
Odd Manhatten-192	3	2565055	2563260	344640

Table 4.6: NIST security category 3 and 4 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
McNie-4Q-192-1	3	465	487	558
McNie-4Q-192-2	3	512	539	619
McNie-3Q-192-1	3	247	569	732
McNie-3Q-192-2	3	274	631	814
KCL AKCN-MLWE-CCA	4	1312	992	-
LAC-CPA-192	3	2080	1056	1536
NTRUEncrypt-734	4	1173	1023	1023
SABER	3	2304	1248	1088
KINDI-512-2-2-2	4	2112	1728	2688
Compact LWE	3	232	2064	360
KINDI-512-2-4-1	4	2304	1984	2688
KINDI-512-3-2-1	5	2752	2368	3328
RLizard-CATEGORY3-N1024	3	513	4096	4272
LIMA-CCA-2p-1024	3	9217	6145	6147
LIMA-CPA-2p-1024	3	9217	6145	6147
LEDApkc-3-2	3	972	7200	14400
RLizard-CATEGORY3-N2048	3	369	8192	8496
LEDApkc-3-3	3	1196	10384	15576
LEDApkc-3-4	3	1364	13152	17536
LIMA-CCA-sp-1822	3	21865	14577	14555
LIMA-CPA-sp-1822	3	21865	14577	14555
Titanium CPA high	3	32	17952	8320
Giophantus 868	3	868	20796	41592
LOTUS-192	3	1126400	1025024	1456
Lizard-CATEGORY3-N816	3	313344	2457600	2496
Odd Manhatten-192	3	2565055	2563260	344640
Lizard-CATEGORY3-N925	3	365568	2736128	2768

Table 4.7: NIST security category 3 and 4 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
Compact LWE	3	232	2064	360
McNie-4Q-192-1	3	465	487	558
McNie-4Q-192-2	3	512	539	619
McNie-3Q-192-1	3	247	569	732
McNie-3Q-192-2	3	274	631	814
NTRUEncrypt-734	4	1173	1023	1023
SABER	3	2304	1248	1088
LOTUS-192	3	1126400	1025024	1456
LAC-CPA-192	3	2080	1056	1536
Lizard-CATEGORY3-N816	3	313344	2457600	2496
KINDI-512-2-2-2	4	2112	1728	2688
KINDI-512-2-4-1	4	2304	1984	2688
Lizard-CATEGORY3-N925	3	365568	2736128	2768
KINDI-512-3-2-1	5	2752	2368	3328
RLizard-CATEGORY3-N1024	3	513	4096	4272
LIMA-CCA-2p-1024	3	9217	6145	6147
LIMA-CPA-2p-1024	3	9217	6145	6147
Titanium CPA high	3	32	17952	8320
RLizard-CATEGORY3-N2048	3	369	8192	8496
LEDApkc-3-2	3	972	7200	14400
LIMA-CCA-sp-1822	3	21865	14577	14555
LIMA-CPA-sp-1822	3	21865	14577	14555
LEDApkc-3-3	3	1196	10384	15576
LEDApkc-3-4	3	1364	13152	17536
Giophantus 868	3	868	20796	41592
Odd Manhatten-192	3	2565055	2563260	344640

Table 4.8: NIST security category 5 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
Titanium CPA super	5	32	23552	8320
McNie-3Q-256-1	5	337	819	1065
McNie-3Q-256-2	5	348	829	1078
RLizard-CATEGORY5	5	513	8192	8512
McNie-4Q-256-1	5	584	630	619
McNie-4Q-256-2	5	601	647	749
Giophantus 1134	5	1134	27204	54408
SABER fire	5	1664	1312	1472
KINDI-256-5-2-2	5	1712	1456	2496
Guess Again	5	2000	4000	9216000
LAC-CPA-256	5	2080	1056	2048
KINDI-512-3-2-1	5	2752	2368	3328
LEDApkc-5-2	5	1244	12384	24768
LEDApkc-5-3	5	1548	18016	27024
LEDApkc-5-4	5	1772	22704	30272
NTRUEncrypt-1024	5	8194	4097	4097
LIMA-CCA-2p-2048	5	18433	12289	12291
LIMA-CPA-2p-2048	5	18433	12289	12291
LIMA-CCA-sp-2062	5	24745	16497	16475
LIMA-CPA-sp-2062	5	24745	16497	16475
Lizard-CATEGORY5-N1088	5	557056	6553600	3328
Lizard-CATEGORY5-N1300	5	665600	3710976	3752
LOTUS-256	5	1630720	1470976	1768
Odd Manhatten-256	5	4456650	4454241	616704

Table 4.9: NIST security category 5 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	c[B]
McNie-4Q-256-1	5	584	630	619
McNie-4Q-256-2	5	601	647	749
McNie-3Q-256-1	5	337	819	1065
McNie-3Q-256-2	5	348	829	1078
LAC-CPA-256	5	2080	1056	2048
SABER fire	5	1664	1312	1472
KINDI-256-5-2-2	5	1712	1456	2496
KINDI-512-3-2-1	5	2752	2368	3328
Guess Again	5	2000	4000	9216000
NTRUEncrypt-1024	5	8194	4097	4097
RLizard-CATEGORY5	5	513	8192	8512
LIMA-CCA-2p-2048	5	18433	12289	12291
LIMA-CPA-2p-2048	5	18433	12289	12291
LEDApkc-5-2	5	1244	12384	24768
LIMA-CCA-sp-2062	5	24745	16497	16475
LIMA-CPA-sp-2062	5	24745	16497	16475
LEDApkc-5-3	5	1548	18016	27024
LEDApkc-5-4	5	1772	22704	30272
Titanium CPA super	5	32	23552	8320
Giophantus 1134	5	1134	27204	54408
LOTUS-256	5	1630720	1470976	1768
Lizard-CATEGORY5-N1300	5	665600	3710976	3752
Odd Manhatten-256	5	4456650	4454241	616704
Lizard-CATEGORY5-N1088	5	557056	6553600	3328

Table 4.10: NIST security category 5 encryption implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**
McNie-4Q-256-1	5	584	630	619
McNie-4Q-256-2	5	601	647	749
McNie-3Q-256-1	5	337	819	1065
McNie-3Q-256-2	5	348	829	1078
SABER fire	5	1664	1312	1472
LOTUS-256	5	1630720	1470976	1768
LAC-CPA-256	5	2080	1056	2048
KINDI-256-5-2-2	5	1712	1456	2496
KINDI-512-3-2-1	5	2752	2368	3328
Lizard-CATEGORY5-N1088	5	557056	6553600	3328
Lizard-CATEGORY5-N1300	5	665600	3710976	3752
NTRUEncrypt-1024	5	8194	4097	4097
Titanium CPA super	5	32	23552	8320
RLizard-CATEGORY5	5	513	8192	8512
LIMA-CCA-2p-2048	5	18433	12289	12291
LIMA-CPA-2p-2048	5	18433	12289	12291
LIMA-CCA-sp-2062	5	24745	16497	16475
LIMA-CPA-sp-2062	5	24745	16497	16475
LEDApkc-5-2	5	1244	12384	24768
LEDApkc-5-3	5	1548	18016	27024
LEDApkc-5-4	5	1772	22704	30272
Giophantus 1134	5	1134	27204	54408
Odd Manhatten-256	5	4456650	4454241	616704
Guess Again	5	2000	4000	9216000

4.1.2 KEM Space Requirements

For all Tables containing space requirements for KEM implementations, *sec* denotes the claimed NIST level of security for the implementation (See Section 2.5.1), and $k_{private}$ (private key), *pk* (public key), *c* (ciphertext), and (pk+c) (public keys + ciphertexts) entries all denote length in bytes. The latter of these properties is especially important, as the sum of a KEM's public key and ciphertext is the sum of the properties which are to be transferred between two parties when producing ephemeral keys. If left blank, the implementation does not fulfil any of the NIST security levels' requirements or lacks information about the given property.

Table 4.11 containing all KEM algorithm implementations, and is sorted alphabetically after submission implementation names. This Table is meant as a referencing and look-up Table for all implementations. Following this Table, there are $3 \cdot 4$ Tables containing the same algorithm implementations, divided into the NIST security levels 1 and 2, 3 and 4, and 5. There are 4 different Tables for each level, each sorted after space requirements each of the previously mentioned property lengths. Below is an overview of the Tables for each level.

- Levels 1 and 2:
  - Table 4.12, sorted by private key length
  - Table 4.13, sorted by public key length
  - Table 4.14, sorted by ciphertext length
  - Table 4.15, sorted by the length of the sum of ciphertext + public key
- Levels 3 and 4:
  - Table 4.16, sorted by private key length
  - Table 4.17, sorted by public key length
  - Table 4.18, sorted by ciphertext length
  - Table 4.19, sorted by the length of the sum of ciphertext + public key
- Level 5:
  - Table 4.20, sorted by private key length
  - Table 4.21, sorted by public key length
  - Table 4.22, sorted by ciphertext length
  - Table 4.23, sorted by the length of the sum of ciphertext + public key

Table 4.11: KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted alphabetically after submission implementation names.

Submission Implementation	Sec	$k_{private}$[B]	k_{public}[B]	c[B]	pk+c[B]
BIG QUAKE 1	1	14772	25482	201	25683
BIG QUAKE 3	3	30860	84132	406	84538
BIG QUAKE 5	5	41804	149800	492	150292
BIKE-1 1	1	267	2542	2542	5084
BIKE-1 3	3	287	5474	5474	10948
BIKE-1 5	5	548	8188	8188	16376
BIKE-2 1	1	267	1272	1272	2544
BIKE-2 3	3	412	1744	1744	3488
BIKE-2 5	5	548	4096	4096	8192
BIKE-3 1	1	252	2758	2758	5516
BIKE-3 3	3	396	5422	5422	10844
BIKE-3 5	5	566	9034	9034	18068
CFPKM128	1	128	696	729	1425
Classic McEliece mceliece6960119	5	13908	1047319	226	1047545
Classic McEliece mceliece8192128	5	14080	1357824	240	1358064
Compact LWE	3	232	2064	360	2424
CRYSTALS-KYBER 512	1	1632	736	800	1536
CRYSTALS-KYBER 768	3	2400	1088	1152	2240
CRYSTALS-KYBER 1024	5	3168	1440	1504	2944
DAGS 1	1	432640	6760	552	7312
DAGS 3	3	1284096	8448	944	9392
DAGS 5	5	2230272	11616	1616	13232
DING Key Exchange 512	1	1536	1040	1088	2128
DING Key Exchange 1024	3/5	3072	2064	2176	4240
DME-144	1	144	1152	144	1296
DME-288	5	288	2304	288	2592
FrodoKEM 640	1	19872	9616	9736	19352
FrodoKEM 976	3	31272	15632	15768	31400
HILA5	5	1824	1824	2012	3836
HQC Basic I	1	2859	2819	5622	8441
HQC Basic II	1	3049	3009	6002	9011
HQC Basic III	1	3165	3125	6234	9359
HQC Advanced I	3	5155	5155	10214	15369
HQC Advanced II	3	5539	5499	10982	16481
HQC Advanced III	3	5924	5884	11752	17636
HQC Paranoiac I	5	7457	7417	14818	22235
HQC Paranoiac II	5	8029	7989	15962	23951
HQC Paranoiac III	5	8543	8503	16990	25493
HQC Paranoiac IV	5	8937	8897	17778	26675
KCL AKCN-MLWE	4	288	991	1120	2111
KCL AKCN-RLWE	5	1664	1696	2083	3779
KCL OKCN-MLWE	4	288	992	1120	2112

KCL OKCN-RLWE	5	1664	1696	1995	3691
KINDI-256-3-4-2	2	1472	1184	1792	2976
KINDI-256-5-2-2	5	1712	1456	2496	3952
KINDI-512-2-2-2	4	2112	1728	2688	4416
KINDI-512-2-4-1	4	2304	1984	2688	4672
KINDI-512-3-2-1	5	2752	2368	3328	5696
LAC-CCA-128	1	1056	544	1024	1568
LAC-CCA-192	3	2080	1056	1536	2592
LAC-CCA-256	5	2080	1056	2048	3104
LAKE I	1	40	423	423	846
LAKE II	3	40	636	636	1272
LAKE III	5	40	826	826	1652
LEDAkem-1-2	1	668	3480	3480	6960
LEDAkem-1-3	1	844	4688	2344	7032
LEDAkem-1-4	1	1036	6408	2136	8544
LEDAkem-3-2	3	972	7200	7200	14400
LEDAkem-3-3	3	1196	10384	5192	15576
LEDAkem-3-4	3	1364	13152	4384	17536
LEDAkem-5-2	5	1244	12384	12384	24768
LEDAkem-5-3	5	1548	18016	9008	27024
LEDAkem-5-4	5	1772	22704	7568	30272
Lepton.CPA Light I	-	32	1045	1585	2630
Lepton.CPA Light II	1	40	1045	1966	3011
Lepton.CPA Moderate I	1	38	2052	2465	4517
Lepton.CPA Moderate II	1	48	2052	2719	4771
Lepton.CPA Moderate III	3	56	2052	2973	5025
Lepton.CPA Moderate IV	5	74	2052	3989	6041
Lepton.CPA Paranoid I	5	70	4128	5303	9431
Lepton.CPA Paranoid II	5	80	4128	5557	9685
Lepton-CCA Light I	-	1077	1045	1617	2662
Lepton-CCA Light II	1	1085	1045	1998	3043
Lepton-CCA Moderate I	1	2090	2052	2497	4549
Lepton-CCA Moderate II	1	2100	2052	2751	4803
Lepton-CCA Moderate III	3	2108	2052	3005	5057
Lepton-CCA Moderate IV	5	2126	2052	4021	6073
Lepton-CCA Paranoid I	5	4198	4128	5335	9463
Lepton-CCA Paranoid II	5	4208	4128	5589	9717
LIMA-CCA-sp-1018	1	9163	6109	4209	10318
LIMA-CCA-sp-1306	2	15673	10449	6763	17212
LIMA-CCA-sp-1822	3	21865	14577	8827	23404
LIMA-CCA-sp-2062	5	24745	16497	9787	26284
LIMA-CCA-2p-1024	3	9217	6145	4227	10372
LIMA-CCA-2p-2048	5	18433	12289	7299	19588
LIMA-CPA-sp-1018	1	9163	6109	3825	9934
LIMA-CPA-sp-1306	2	15673	10449	6251	16700
LIMA-CPA-sp-1822	3	21865	14577	8315	22892

LIMA-CPA-sp-2062	5	24745	16497	9275	25772
LIMA-CPA-2p-1024	3	9217	6145	3843	9988
LIMA-CPA-2p-2048	5	18433	12289	6915	19204
Lizard-CATEGORY1-N536	1	8608	1130496	17696	1148192
Lizard-CATEGORY1-N663	1	10640	1390592	10896	1401488
Lizard-CATEGORY3-N816	3	19632	1720320	26928	1747248
Lizard-CATEGORY3-N925	3	22896	1998848	31280	2030128
Lizard-CATEGORY5-N1088	5	34880	4587520	35904	4623424
Lizard-CATEGORY5-N1300	5	41664	2727936	42688	2770624
RLizard-CATEGORY1	1	385	4096	2080	6176
RLizard-CATEGORY3-N1024	3	641	4096	4144	8240
RLizard-CATEGORY3-N2048	3	625	8192	8240	16432
RLizard-CATEGORY5	5	769	8192	8256	16448
LOCKER I	1	787	747	875	1622
LOCKER II	3	1119	1079	1207	2286
LOCKER III	5	1286	1246	1374	2620
LOCKER IV	1	1050	1010	1138	2148
LOCKER V	3	1279	1339	1467	2806
LOCKER VI	5	1482	1442	1570	3012
LOCKER VII	1	1679	1639	1767	3406
LOCKER VIII	3	1977	1937	2065	4002
LOCKER IX	5	2238	2198	2326	4524
LOTUS-128	1	714240	658944	1160	660104
LOTUS-192	3	1126400	1025024	1480	1026504
LOTUS-256	5	1630720	1470976	1800	1472776
Mersenne-756839	5	32	189248	160160	349408
NewHope-CPA-512	1	869	928	1088	2016
NewHope-CPA-1024	5	1792	1824	2176	4000
NewHope-CCA-512	1	1120	928	1120	2048
NewHope-CCA-1024	5	3680	1824	2208	4032
NTRU-HRSS-KEM-701	1	1418	1138	1278	2416
NTRU Prime ntrulpr4591761	5	1238	1047	1175	2222
NTRU Prime sntrup4591761	5	1600	1218	1047	2265
NTRUEncrypt-443	1	701	611	611	1222
NTRUEncrypt-734	4	1173	1023	1023	2046
NTRUEncrypt-1024	5	8194	4097	4097	8194
NTS-KEM (12,6)	1	9216	319488	128	319616
NTS-KEM (13, 80)	3	17524	929760	162	929922
NTS-KEM (13, 136)	5	19890	1419704	253	1419957
Ouroboros-R-128	1	40	676	1272	1948
Ouroboros-R-192	3	40	807	1534	2341
Ouroboros-R-256	5	40	1112	2144	3256
PQRSA-KEM-15	-	98304	32768	32768	65536
PQRSA-KEM-20	-	3145728	1048576	1048576	2097152
PQRSA-KEM-25	-	100663296	33554432	33554432	67108864
PQRSA-KEM-30	2	3221225472	1073741824	1073741824	2147483648

QC-MDPC KEM	5	548	4097	8226	12323
Ramstake 216091	1	54056	27044	28064	55108
Ramstake 756839	5	189242	94637	96167	190804
RLCE-KEM-128A	1	179946	118441	785	119226
RLCE-KEM-128B	1	310116	188001	988	188989
RLCE-KEM-192A	3	440008	287371	1238	288609
RLCE-KEM-192B	3	747393	450761	1545	452306
RLCE-KEM-256A	5	1048176	742089	2023	744112
RLCE-KEM-256B	5	1773271	1232001	2640	1234641
Round2-nround2-nd-l1	1	100	417	464	881
Round2-nround2-nd-l2	2	122	519	614	1133
Round2-nround2-nd-l3	3	139	581	652	1233
Round2-nround2-nd-l4	4	165	707	898	1605
Round2-nround2-nd-l5	5	165	691	818	1509
Round2-uround2-nd-l1	1	105	435	482	917
Round2-uround2-nd-l2	2	131	555	618	1173
Round2-uround2-nd-l3	3	135	565	636	1201
Round2-uround2-nd-l4	4	175	749	940	1689
Round2-uround2-nd-l5	5	169	709	868	1577
Round2-uround2-n1-fn0-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn0-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn0-l3	3	945	5223	6972	12195
Round2-uround2-n1-fn0-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn0-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn1-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn1-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn1-l3	3	945	5223	6972	12195
Round2-uround2-n1-fn1-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn1-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn2-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn2-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn2-l3	3	945	5223	6972	12195
Round2-uround2-n1-fn2-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn2-l5	5	1572	8679	8710	17389
RQC-128	1	826	786	1556	2342
RQC-192	3	1451	1411	2806	4217
RQC-256	5	1835	1795	2574	4369
SABER light	1	1568	672	736	1408
SABER	3	2304	992	1088	2080
SABER fire	5	3040	1312	1472	2784
SIKEp503	1	434	378	402	780
SIKEp751	3	644	564	596	1160
SIKEp964	5	826	726	766	1492
Three Bears BabyBear	2	40	804	917	1721
Three Bears BabyBear Ephem	2	40	804	917	1721
Three Bears MamaBear	4	40	1194	1307	2501

Three Bears MamaBear Ephem	4	40	1194	1307	2501
Three Bears PapaBear	5	40	1584	1697	3281
Three Bears PapaBear Ephem	5	40	1584	1697	3281
Titanium CCA toy	-	12224	12192	2720	14912
Titanium CCA lite	-	14752	14720	3008	17728
Titanium CCA standard	1	16384	16352	3552	19904
Titanium CCA medium	1	18304	18272	4544	22816
Titanium CCA high	3	20544	20512	6048	26560
Titanium CCA super	5	26944	26912	8352	35264

Table 4.12: NIST security category 1 and 2 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	c[B]	pk+c[B]
LAKE I	1	40	423	423	846
Lepton.CPA Light II	1	40	1045	1966	3011
Three Bears BabyBear	2	40	804	917	1721
Three Bears BabyBear Ephem	2	40	804	917	1721
Ouroboros-R-128	1	40	676	1272	1948
Lepton.CPA Moderate I	1	38	2052	2465	4517
Lepton.CPA Moderate II	1	48	2052	2719	4771
Round2-nround2-nd-l1	1	100	417	464	881
Round2-nround2-nd-l2	2	122	519	614	1133
Round2-uround2-nd-l1	1	105	435	482	917
CFPKM128	1	128	696	729	1425
Round2-uround2-nd-l2	2	131	555	618	1173
DME-144	1	144	1152	144	1296
BIKE-1 1	1	267	2542	2542	5084
BIKE-2 1	1	267	1272	1272	2544
BIKE-3 1	1	252	2758	2758	5516
RLizard-CATEGORY1	1	385	4096	2080	6176
SIKEp503	1	434	378	402	780
Round2-uround2-n1-fn0-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn1-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn2-l1	1	625	3455	4837	8292
LEDAkem-1-2	1	668	3480	3480	6960
NTRUEncrypt-443	1	701	611	611	1222
LOCKER I	1	787	747	875	1622
RQC-128	1	826	786	1556	2342
LEDAkem-1-3	1	844	4688	2344	7032
NewHope-CPA-512	1	869	928	1088	2016
LEDAkem-1-4	1	1036	6408	2136	8544
LOCKER IV	1	1050	1010	1138	2148
LAC-CCA-128	1	1056	544	1024	1568
Lepton-CCA Light II	1	1085	1045	1998	3043
NewHope-CCA-512	1	1120	928	1120	2048
Round2-uround2-n1-fn0-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn1-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn2-l2	2	1160	6413	6428	12841
NTRU-HRSS-KEM-701	1	1418	1138	1278	2416
KINDI-256-3-4-2	2	1472	1184	1792	2976
DING Key Exchange 512	1	1536	1040	1088	2128
SABER light	1	1568	672	736	1408
CRYSTALS-KYBER 512	1	1632	736	800	1536
LOCKER VII	1	1679	1639	1767	3406
Lepton-CCA Moderate I	1	2090	2052	2497	4549

Lepton-CCA Moderate II	1	2100	2052	2751	4803
HQC Basic I	1	2859	2819	5622	8441
HQC Basic II	1	3049	3009	6002	9011
DING Key Exchange 1024	3/5	3072	2064	2176	4240
HQC Basic III	1	3165	3125	6234	9359
Lizard-CATEGORY1-N536	1	8608	1130496	17696	1148192
LIMA-CCA-sp-1018	1	9163	6109	4209	10318
LIMA-CPA-sp-1018	1	9163	6109	3825	9934
NTS-KEM (12,6)	1	9216	319488	128	319616
Lizard-CATEGORY1-N663	1	10640	1390592	10896	1401488
BIG QUAKE 1	1	14772	25482	201	25683
LIMA-CCA-sp-1306	2	15673	10449	6763	17212
LIMA-CPA-sp-1306	2	15673	10449	6251	16700
Titanium CCA standard	1	16384	16352	3552	19904
Titanium CCA medium	1	18304	18272	4544	22816
FrodoKEM 640	1	19872	9616	9736	19352
Ramstake 216091	1	54056	27044	28064	55108
RLCE-KEM-128A	1	179946	118441	785	119226
RLCE-KEM-128B	1	310116	188001	988	188989
DAGS 1	1	432640	6760	552	7312
LOTUS-128	1	714240	658944	1160	660104
PQRSA-KEM-30	2	3221225472	1073741824	1073741824	2147483648

Table 4.13: NIST security category 1 and 2 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	c[B]	pk+c[B]
SIKEp503	1	434	378	402	780
Round2-nround2-nd-l1	1	100	417	464	881
LAKE I	1	40	423	423	846
Round2-uround2-nd-l1	1	105	435	482	917
Round2-nround2-nd-l2	2	122	519	614	1133
LAC-CCA-128	1	1056	544	1024	1568
Round2-uround2-nd-l2	2	131	555	618	1173
NTRUEncrypt-443	1	701	611	611	1222
SABER light	1	1568	672	736	1408
Ouroboros-R-128	1	40	676	1272	1948
CFPKM128	1	128	696	729	1425
CRYSTALS-KYBER 512	1	1632	736	800	1536
LOCKER I	1	787	747	875	1622
RQC-128	1	826	786	1556	2342
Three Bears BabyBear	2	40	804	917	1721
Three Bears BabyBear Ephem	2	40	804	917	1721
NewHope-CCA-512	1	1120	928	1120	2048
NewHope-CPA-512	1	869	928	1088	2016
LOCKER IV	1	1050	1010	1138	2148
DING Key Exchange 512	1	1536	1040	1088	2128
Lepton.CPA Light II	1	40	1045	1966	3011
Lepton-CCA Light II	1	1085	1045	1998	3043
DME-144	1	144	1152	144	1296
NTRU-HRSS-KEM-701	1	1418	1138	1278	2416
KINDI-256-3-4-2	2	1472	1184	1792	2976
BIKE-2 1	1	267	1272	1272	2544
LOCKER VII	1	1679	1639	1767	3406
DING Key Exchange 1024	3/5	3072	2064	2176	4240
Lepton-CCA Moderate I	1	2090	2052	2497	4549
Lepton-CCA Moderate II	1	2100	2052	2751	4803
Lepton.CPA Moderate I	1	38	2052	2465	4517
Lepton.CPA Moderate II	1	48	2052	2719	4771
BIKE-1 1	1	267	2542	2542	5084
BIKE-3 1	1	252	2758	2758	5516
HQC Basic I	1	2859	2819	5622	8441
HQC Basic II	1	3049	3009	6002	9011
HQC Basic III	1	3165	3125	6234	9359
Round2-uround2-n1-fn0-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn1-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn2-l1	1	625	3455	4837	8292
LEDAkem-1-2	1	668	3480	3480	6960
RLizard-CATEGORY1	1	385	4096	2080	6176

LEDAkem-1-3	1	844	4688	2344	7032
LIMA-CCA-sp-1018	1	9163	6109	4209	10318
LIMA-CPA-sp-1018	1	9163	6109	3825	9934
LEDAkem-1-4	1	1036	6408	2136	8544
Round2-uround2-n1-fn0-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn1-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn2-l2	2	1160	6413	6428	12841
DAGS 1	1	432640	6760	552	7312
FrodoKEM 640	1	19872	9616	9736	19352
LIMA-CCA-sp-1306	2	15673	10449	6763	17212
LIMA-CPA-sp-1306	2	15673	10449	6251	16700
Titanium CCA standard	1	16384	16352	3552	19904
Titanium CCA medium	1	18304	18272	4544	22816
BIG QUAKE 1	1	14772	25482	201	25683
Ramstake 216091	1	54056	27044	28064	55108
RLCE-KEM-128A	1	179946	118441	785	119226
RLCE-KEM-128B	1	310116	188001	988	188989
NTS-KEM (12, 6)	1	9216	319488	128	319616
LOTUS-128	1	714240	658944	1160	660104
Lizard-CATEGORY1-N536	1	8608	1130496	17696	1148192
Lizard-CATEGORY1-N663	1	10640	1390592	10896	1401488
PQRSA-KEM-30	2	3221225472	1073741824	1073741824	2147483648

Table 4.14: NIST security category 1 and 2 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for ciphertexts.

Submission Implementation	Sec	$k_{private}$[B]	k_{public}[B]	c[B]	pk+c[B]
NTS-KEM (12, 6)	1	9216	319488	128	319616
DME-144	1	144	1152	144	1296
BIG QUAKE 1	1	14772	25482	201	25683
SIKEp503	1	434	378	402	780
LAKE I	1	40	423	423	846
Round2-nround2-nd-l1	1	100	417	464	881
Round2-uround2-nd-l1	1	105	435	482	917
DAGS 1	1	432640	6760	552	7312
NTRUEncrypt-443	1	701	611	611	1222
Round2-nround2-nd-l2	2	122	519	614	1133
Round2-uround2-nd-l2	2	131	555	618	1173
CFPKM128	1	128	696	729	1425
SABER light	1	1568	672	736	1408
RLCE-KEM-128A	1	179946	118441	785	119226
CRYSTALS-KYBER 512	1	1632	736	800	1536
LOCKER I	1	787	747	875	1622
Three Bears BabyBear	2	40	804	917	1721
Three Bears BabyBear Ephem	2	40	804	917	1721
RLCE-KEM-128B	1	310116	188001	988	188989
DING Key Exchange 512	1	1536	1040	1088	2128
NewHope-CPA-512	1	869	928	1088	2016
LAC-CCA-128	1	1056	544	1024	1568
NewHope-CCA-512	1	1120	928	1120	2048
LOTUS-128	1	714240	658944	1160	660104
LOCKER IV	1	1050	1010	1138	2148
BIKE-2 1	1	267	1272	1272	2544
Ouroboros-R-128	1	40	676	1272	1948
NTRU-HRSS-KEM-701	1	1418	1138	1278	2416
RQC-128	1	826	786	1556	2342
LOCKER VII	1	1679	1639	1767	3406
KINDI-256-3-4-2	2	1472	1184	1792	2976
Lepton.CPA Light II	1	40	1045	1966	3011
Lepton-CCA Light II	1	1085	1045	1998	3043
RLizard-CATEGORY1	1	385	4096	2080	6176
DING Key Exchange 1024	3/5	3072	2064	2176	4240
LEDAkem-1-4	1	1036	6408	2136	8544
LEDAkem-1-3	1	844	4688	2344	7032
Lepton.CPA Moderate I	1	38	2052	2465	4517
Lepton-CCA Moderate I	1	2090	2052	2497	4549
BIKE-1 1	1	267	2542	2542	5084
Lepton.CPA Moderate II	1	48	2052	2719	4771
Lepton-CCA Moderate II	1	2100	2052	2751	4803

BIKE-3 1	1	252	2758	2758	5516
LEDAkem-1-2	1	668	3480	3480	6960
Titanium CCA standard	1	16384	16352	3552	19904
LIMA-CPA-sp-1018	1	9163	6109	3825	9934
LIMA-CCA-sp-1018	1	9163	6109	4209	10318
Titanium CCA medium	1	18304	18272	4544	22816
Round2-uround2-n1-fn0-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn1-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn2-l1	1	625	3455	4837	8292
HQC Basic I	1	2859	2819	5622	8441
HQC Basic II	1	3049	3009	6002	9011
HQC Basic III	1	3165	3125	6234	9359
LIMA-CPA-sp-1306	2	15673	10449	6251	16700
Round2-uround2-n1-fn0-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn1-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn2-l2	2	1160	6413	6428	12841
LIMA-CCA-sp-1306	2	15673	10449	6763	17212
FrodoKEM 640	1	19872	9616	9736	19352
Lizard-CATEGORY1-N663	1	10640	1390592	10896	1401488
Lizard-CATEGORY1-N536	1	8608	1130496	17696	1148192
Ramstake 216091	1	54056	27044	28064	55108
PQRSA-KEM-30	2	3221225472	1073741824	1073741824	2147483648

Table 4.15: NIST security category 1 and 2 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for the combined length of public keys and ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
SIKEp503	1	434	378	402	780
LAKE I	1	40	423	423	846
Round2-nround2-nd-l1	1	100	417	464	881
Round2-uround2-nd-l1	1	105	435	482	917
Round2-nround2-nd-l2	2	122	519	614	1133
Round2-uround2-nd-l2	2	131	555	618	1173
NTRUEncrypt-443	1	701	611	611	1222
DME-144	1	144	1152	144	1296
SABER light	1	1568	672	736	1408
CFPKM128	1	128	696	729	1425
CRYSTALS-KYBER 512	1	1632	736	800	1536
LAC-CCA-128	1	1056	544	1024	1568
LOCKER I	1	787	747	875	1622
Three Bears BabyBear	2	40	804	917	1721
Three Bears BabyBear Ephem	2	40	804	917	1721
Ouroboros-R-128	1	40	676	1272	1948
NewHope-CPA-512	1	869	928	1088	2016
NewHope-CCA-512	1	1120	928	1120	2048
DING Key Exchange 512	1	1536	1040	1088	2128
LOCKER IV	1	1050	1010	1138	2148
RQC-128	1	826	786	1556	2342
NTRU-HRSS-KEM-701	1	1418	1138	1278	2416
BIKE-2 1	1	267	1272	1272	2544
KINDI-256-3-4-2	2	1472	1184	1792	2976
Lepton.CPA Light II	1	40	1045	1966	3011
Lepton-CCA Light II	1	1085	1045	1998	3043
LOCKER VII	1	1679	1639	1767	3406
DING Key Exchange 1024	3/5	3072	2064	2176	4240
Lepton.CPA Moderate I	1	38	2052	2465	4517
Lepton-CCA Moderate I	1	2090	2052	2497	4549
Lepton.CPA Moderate II	1	48	2052	2719	4771
Lepton-CCA Moderate II	1	2100	2052	2751	4803
BIKE-1 1	1	267	2542	2542	5084
BIKE-3 1	1	252	2758	2758	5516
RLizard-CATEGORY1	1	385	4096	2080	6176
LEDAkem-1-2	1	668	3480	3480	6960
LEDAkem-1-3	1	844	4688	2344	7032
DAGS 1	1	432640	6760	552	7312
Round2-uround2-n1-fn0-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn1-l1	1	625	3455	4837	8292
Round2-uround2-n1-fn2-l1	1	625	3455	4837	8292
HQC Basic I	1	2859	2819	5622	8441

LEDAkem-1-4	1	1036	6408	2136	8544
HQC Basic II	1	3049	3009	6002	9011
HQC Basic III	1	3165	3125	6234	9359
LIMA-CPA-sp-1018	1	9163	6109	3825	9934
LIMA-CCA-sp-1018	1	9163	6109	4209	10318
Round2-uround2-n1-fn0-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn1-l2	2	1160	6413	6428	12841
Round2-uround2-n1-fn2-l2	2	1160	6413	6428	12841
LIMA-CPA-sp-1306	2	15673	10449	6251	16700
LIMA-CCA-sp-1306	2	15673	10449	6763	17212
FrodoKEM 640	1	19872	9616	9736	19352
Titanium CCA standard	1	16384	16352	3552	19904
Titanium CCA medium	1	18304	18272	4544	22816
BIG QUAKE 1	1	14772	25482	201	25683
Ramstake 216091	1	54056	27044	28064	55108
RLCE-KEM-128A	1	179946	118441	785	119226
RLCE-KEM-128B	1	310116	188001	988	188989
NTS-KEM (12, 6)	1	9216	319488	128	319616
LOTUS-128	1	714240	658944	1160	660104
Lizard-CATEGORY1-N536	1	8608	1130496	17696	1148192
Lizard-CATEGORY1-N663	1	10640	1390592	10896	1401488
PQRSA-KEM-30	2	3221225472	1073741824	1073741824	2147483648

Table 4.16: NIST security category 3 and 4 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
LAKE II	3	40	636	636	1272
Ouroboros-R-192	3	40	807	1534	2341
Three Bears MamaBear	4	40	1194	1307	2501
Three Bears MamaBear Ephem	4	40	1194	1307	2501
Lepton.CPA Moderate III	3	56	2052	2973	5025
Round2-uround2-nd-l3	3	135	565	636	1201
Round2-nround2-nd-l3	3	139	581	652	1233
Round2-nround2-nd-l4	4	165	707	898	1605
Round2-uround2-nd-l4	4	175	749	940	1689
CFPKM182	3	182	928	729	1657
Compact LWE	3	232	2064	360	2424
BIKE-1 3	3	287	5474	5474	10948
KCL AKCN-MLWE	4	288	991	1120	2111
KCL OKCN-MLWE	4	288	992	1120	2112
BIKE-3 3	3	396	5422	5422	10844
BIKE-2 3	3	412	1744	1744	3488
RLizard-CATEGORY3-N2048	3	625	8192	8240	16432
RLizard-CATEGORY3-N1024	3	641	4096	4144	8240
SIKEp751	3	644	564	596	1160
Round2-uround2-n1-fn1-l3	3	945	5223	6972	12195
Round2-uround2-n1-fn2-l3	3	945	5223	6972	12195
LEDAkem-3-2	3	972	7200	7200	14400
LOCKER II	3	1119	1079	1207	2286
NTRUEncrypt-734	4	1173	1023	1023	2046
LEDAkem-3-3	3	1196	10384	5192	15576
LOCKER V	3	1279	1339	1467	2806
LEDAkem-3-4	3	1364	13152	4384	17536
RQC-192	3	1451	1411	2806	4217
Round2-uround2-n1-fn0-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn1-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn2-l4	4	1965	10857	10904	21761
LOCKER VIII	3	1977	1937	2065	4002
LAC-CCA-192	3	2080	1056	1536	2592
Lepton-CCA Moderate III	3	2108	2052	3005	5057
KINDI-512-2-2-2	4	2112	1728	2688	4416
KINDI-512-2-4-1	4	2304	1984	2688	4672
SABER	3	2304	992	1088	2080
CRYSTALS-KYBER 768	3	2400	1088	1152	2240
DING Key Exchange 1024	3/5	3072	2064	2176	4240
HQC Advanced I	3	5155	5155	10214	15369
HQC Advanced II	3	5539	5499	10982	16481
HQC Advanced III	3	5924	5884	11752	17636

LIMA-CCA-2p-1024	3	9217	6145	4227	10372
LIMA-CPA-2p-1024	3	9217	6145	3843	9988
NTS-KEM (13, 80)	3	17524	929760	162	929922
Lizard-CATEGORY3-N816	3	19632	1720320	26928	1747248
Titanium CCA high	3	20544	20512	6048	26560
LIMA-CCA-sp-1822	3	21865	14577	8827	23404
LIMA-CPA-sp-1822	3	21865	14577	8315	22892
Lizard-CATEGORY3-N925	3	22896	1998848	31280	2030128
BIG QUAKE 3	3	30860	84132	406	84538
FrodoKEM 976	3	31272	15632	15768	31400
RLCE-KEM-192A	3	440008	287371	1238	288609
RLCE-KEM-192B	3	747393	450761	1545	452306
LOTUS-192	3	1126400	1025024	1480	1026504
DAGS 3	3	1284096	8448	944	9392

Table 4.17: NIST security category 3 and 4 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
SIKEp751	3	644	564	596	1160
Round2-uround2-nd-l3	3	135	565	636	1201
Round2-nround2-nd-l3	3	139	581	652	1233
LAKE II	3	40	636	636	1272
Round2-nround2-nd-l4	4	165	707	898	1605
Round2-uround2-nd-l4	4	175	749	940	1689
Ouroboros-R-192	3	40	807	1534	2341
CFPKM182	3	182	928	729	1657
KCL AKCN-MLWE	4	288	991	1120	2111
KCL OKCN-MLWE	4	288	992	1120	2112
SABER	3	2304	992	1088	2080
NTRUEncrypt-734	4	1173	1023	1023	2046
LAC-CCA-192	3	2080	1056	1536	2592
LOCKER II	3	1119	1079	1207	2286
CRYSTALS-KYBER 768	3	2400	1088	1152	2240
Three Bears MamaBear	4	40	1194	1307	2501
Three Bears MamaBear Ephem	4	40	1194	1307	2501
LOCKER V	3	1279	1339	1467	2806
RQC-192	3	1451	1411	2806	4217
KINDI-512-2-2-2	4	2112	1728	2688	4416
BIKE-2 3	3	412	1744	1744	3488
LOCKER VIII	3	1977	1937	2065	4002
KINDI-512-2-4-1	4	2304	1984	2688	4672
Lepton-CCA Moderate III	3	2108	2052	3005	5057
Lepton.CPA Moderate III	3	56	2052	2973	5025
Compact LWE	3	232	2064	360	2424
DING Key Exchange 1024	3/5	3072	2064	2176	4240
RLizard-CATEGORY3-N1024	3	641	4096	4144	8240
HQC Advanced I	3	5155	5155	10214	15369
Round2-uround2-n1-fn1-l3	3	945	5223	6972	12195
Round2-uround2-n1-fn2-l3	3	945	5223	6972	12195
BIKE-3 3	3	396	5422	5422	10844
HQC Advanced II	3	5539	5499	10982	16481
BIKE-1 3	3	287	5474	5474	10948
HQC Advanced III	3	5924	5884	11752	17636
LIMA-CCA-2p-1024	3	9217	6145	4227	10372
LIMA-CPA-2p-1024	3	9217	6145	3843	9988
LEDAkem-3-2	3	972	7200	7200	14400
RLizard-CATEGORY3-N2048	3	625	8192	8240	16432
DAGS 3	3	1284096	8448	944	9392
LEDAkem-3-3	3	1196	10384	5192	15576
Round2-uround2-n1-fn0-l4	4	1965	10857	10904	21761

Round2-uround2-n1-fn1-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn2-l4	4	1965	10857	10904	21761
LIMA-CCA-sp-1822	3	21865	14577	8827	23404
LIMA-CPA-sp-1822	3	21865	14577	8315	22892
FrodoKEM 976	3	31272	15632	15768	31400
LEDAkem-3-4	3	1364	13152	4384	17536
Titanium CCA high	3	20544	20512	6048	26560
BIG QUAKE 3	3	30860	84132	406	84538
RLCE-KEM-192A	3	440008	287371	1238	288609
RLCE-KEM-192B	3	747393	450761	1545	452306
NTS-KEM (13, 80)	3	17524	929760	162	929922
LOTUS-192	3	1126400	1025024	1480	1026504
Lizard-CATEGORY3-N816	3	19632	1720320	26928	1747248
Lizard-CATEGORY3-N925	3	22896	1998848	31280	2030128

Table 4.18: NIST security category 3 and 4 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	**$k_{private}$[B]**	**k_{public}[B]**	**c[B]**	**pk+c[B]**
NTS-KEM (13, 80)	3	17524	929760	162	929922
Compact LWE	3	232	2064	360	2424
BIG QUAKE 3	3	30860	84132	406	84538
SIKEp751	3	644	564	596	1160
Round2-uround2-nd-l3	3	135	565	636	1201
LAKE II	3	40	636	636	1272
Round2-nround2-nd-l3	3	139	581	652	1233
CFPKM182	3	182	928	729	1657
Round2-nround2-nd-l4	4	165	707	898	1605
Round2-uround2-nd-l4	4	175	749	940	1689
DAGS 3	3	1284096	8448	944	9392
NTRUEncrypt-734	4	1173	1023	1023	2046
SABER	3	2304	992	1088	2080
KCL OKCN-MLWE	4	288	992	1120	2112
KCL AKCN-MLWE	4	288	991	1120	2111
CRYSTALS-KYBER 768	3	2400	1088	1152	2240
LOCKER II	3	1119	1079	1207	2286
RLCE-KEM-192A	3	440008	287371	1238	288609
Three Bears MamaBear	4	40	1194	1307	2501
Three Bears MamaBear Ephem	4	40	1194	1307	2501
LOCKER V	3	1279	1339	1467	2806
LOTUS-192	3	1126400	1025024	1480	1026504
Ouroboros-R-192	3	40	807	1534	2341
LAC-CCA-192	3	2080	1056	1536	2592
RLCE-KEM-192B	3	747393	450761	1545	452306
BIKE-2 3	3	412	1744	1744	3488
LOCKER VIII	3	1977	1937	2065	4002
DING Key Exchange 1024	3/5	3072	2064	2176	4240
KINDI-512-2-4-1	4	2304	1984	2688	4672
KINDI-512-2-2-2	4	2112	1728	2688	4416
RQC-192	3	1451	1411	2806	4217
Lepton.CPA Moderate III	3	56	2052	2973	5025
Lepton-CCA Moderate III	3	2108	2052	3005	5057
LIMA-CPA-2p-1024	3	9217	6145	3843	9988
RLizard-CATEGORY3-N1024	3	641	4096	4144	8240
LIMA-CCA-2p-1024	3	9217	6145	4227	10372
LEDAkem-3-4	3	1364	13152	4384	17536
LEDAkem-3-3	3	1196	10384	5192	15576
BIKE-3 3	3	396	5422	5422	10844
BIKE-1 3	3	287	5474	5474	10948
Titanium CCA high	3	20544	20512	6048	26560
Round2-uround2-n1-fn1-l3	3	945	5223	6972	12195

Round2-uround2-n1-fn2-l3	3	945	5223	6972	12195
LEDAkem-3-2	3	972	7200	7200	14400
RLizard-CATEGORY3-N2048	3	625	8192	8240	16432
LIMA-CPA-sp-1822	3	21865	14577	8315	22892
LIMA-CCA-sp-1822	3	21865	14577	8827	23404
HQC Advanced I	3	5155	5155	10214	15369
Round2-uround2-n1-fn0-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn1-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn2-l4	4	1965	10857	10904	21761
HQC Advanced II	3	5539	5499	10982	16481
HQC Advanced III	3	5924	5884	11752	17636
FrodoKEM 976	3	31272	15632	15768	31400
Lizard-CATEGORY3-N816	3	19632	1720320	26928	1747248
Lizard-CATEGORY3-N925	3	22896	1998848	31280	2030128

Table 4.19: NIST security category 3 and 4 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for the combined length of public keys and ciphertexts.

Submission Implementation	Sec	$k_{private}$[B]	k_{public}[B]	c[B]	pk+c[B]
SIKEp751	3	644	564	596	1160
Round2-uround2-nd-l3	3	135	565	636	1201
Round2-nround2-nd-l3	3	139	581	652	1233
LAKE II	3	40	636	636	1272
Round2-nround2-nd-l4	4	165	707	898	1605
CFPKM182	3	182	928	729	1657
Round2-uround2-nd-l4	4	175	749	940	1689
NTRUEncrypt-734	4	1173	1023	1023	2046
SABER	3	2304	992	1088	2080
KCL AKCN-MLWE	4	288	991	1120	2111
KCL OKCN-MLWE	4	288	992	1120	2112
CRYSTALS-KYBER 768	3	2400	1088	1152	2240
LOCKER II	3	1119	1079	1207	2286
Ouroboros-R-192	3	40	807	1534	2341
Compact LWE	3	232	2064	360	2424
Three Bears MamaBear	4	40	1194	1307	2501
Three Bears MamaBear Ephem	4	40	1194	1307	2501
LAC-CCA-192	3	2080	1056	1536	2592
LOCKER V	3	1279	1339	1467	2806
BIKE-2 3	3	412	1744	1744	3488
LOCKER VIII	3	1977	1937	2065	4002
RQC-192	3	1451	1411	2806	4217
DING Key Exchange 1024	3/5	3072	2064	2176	4240
KINDI-512-2-2-2	4	2112	1728	2688	4416
KINDI-512-2-4-1	4	2304	1984	2688	4672
Lepton.CPA Moderate III	3	56	2052	2973	5025
Lepton-CCA Moderate III	3	2108	2052	3005	5057
RLizard-CATEGORY3-N1024	3	641	4096	4144	8240
DAGS 3	3	1284096	8448	944	9392
LIMA-CPA-2p-1024	3	9217	6145	3843	9988
LIMA-CCA-2p-1024	3	9217	6145	4227	10372
BIKE-3 3	3	396	5422	5422	10844
BIKE-1 3	3	287	5474	5474	10948
Round2-uround2-n1-fn1-l3	3	945	5223	6972	12195
Round2-uround2-n1-fn2-l3	3	945	5223	6972	12195
LEDAkem-3-2	3	972	7200	7200	14400
HQC Advanced I	3	5155	5155	10214	15369
LEDAkem-3-3	3	1196	10384	5192	15576
RLizard-CATEGORY3-N2048	3	625	8192	8240	16432
HQC Advanced II	3	5539	5499	10982	16481
LEDAkem-3-4	3	1364	13152	4384	17536
HQC Advanced III	3	5924	5884	11752	17636

Round2-uround2-n1-fn0-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn1-l4	4	1965	10857	10904	21761
Round2-uround2-n1-fn2-l4	4	1965	10857	10904	21761
LIMA-CPA-sp-1822	3	21865	14577	8315	22892
LIMA-CCA-sp-1822	3	21865	14577	8827	23404
Titanium CCA high	3	20544	20512	6048	26560
FrodoKEM 976	3	31272	15632	15768	31400
BIG QUAKE 3	3	30860	84132	406	84538
RLCE-KEM-192A	3	440008	287371	1238	288609
RLCE-KEM-192B	3	747393	450761	1545	452306
NTS-KEM (13, 80)	3	17524	929760	162	929922
LOTUS-192	3	1126400	1025024	1480	1026504
Lizard-CATEGORY3-N816	3	19632	1720320	26928	1747248
Lizard-CATEGORY3-N925	3	22896	1998848	31280	2030128

Table 4.20: NIST security category 5 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
Mersenne-756839	5	32	189248	160160	349408
LAKE III	5	40	826	826	1652
Ouroboros-R-256	5	40	1112	2144	3256
Three Bears PapaBear	5	40	1584	1697	3281
Three Bears PapaBear Ephem	5	40	1584	1697	3281
Lepton.CPA Paranoid I	5	70	4128	5303	9431
Lepton.CPA Moderate IV	5	74	2052	3989	6041
Lepton.CPA Paranoid II	5	80	4128	5557	9685
Round2-nround2-nd-l5	5	165	691	818	1509
Round2-uround2-nd-l5	5	169	709	868	1577
DME-288	5	288	2304	288	2592
BIKE-1 5	5	548	8188	8188	16376
BIKE-2 5	5	548	4096	4096	8192
QC-MDPC KEM	5	548	4097	8226	12323
BIKE-3 5	5	566	9034	9034	18068
RLizard-CATEGORY5	5	769	8192	8256	16448
SIKEp964	5	826	726	766	1492
NTRU Prime ntrulpr4591761	5	1238	1047	1175	2222
LEDAkem-5-2	5	1244	12384	12384	24768
LOCKER III	5	1286	1246	1374	2620
LOCKER VI	5	1482	1442	1570	3012
LEDAkem-5-3	5	1548	18016	9008	27024
Round2-uround2-n1-fn0-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn1-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn2-l5	5	1572	8679	8710	17389
NTRU Prime sntrup4591761	5	1600	1218	1047	2265
KCL AKCN-RLWE	5	1664	1696	2083	3779
KCL OKCN-RLWE	5	1664	1696	1995	3691
KINDI-256-5-2-2	5	1712	1456	2496	3952
LEDAkem-5-4	5	1772	22704	7568	30272
NewHope-CPA-1024	5	1792	1824	2176	4000
HILA5	5	1824	1824	2012	3836
RQC-256	5	1835	1795	2574	4369
Lepton-CCA Moderate IV	5	2126	2052	4021	6073
LOCKER IX	5	2238	2198	2326	4524
KINDI-512-3-2-1	5	2752	2368	3328	5696
LAC-CCA-256	5	2080	1056	2048	3104
SABER fire	5	3040	1312	1472	2784
DING Key Exchange 1024	3/5	3072	2064	2176	4240
CRYSTALS-KYBER 1024	5	3168	1440	1504	2944
NewHope-CCA-1024	5	3680	1824	2208	4032
Lepton-CCA Paranoid I	5	4198	4128	5335	9463

Lepton-CCA Paranoid II	5	4208	4128	5589	9717
HQC Paranoiac I	5	7457	7417	14818	22235
HQC Paranoiac II	5	8029	7989	15962	23951
NTRUEncrypt-1024	5	8194	4097	4097	8194
HQC Paranoiac III	5	8543	8503	16990	25493
HQC Paranoiac IV	5	8937	8897	17778	26675
Classic McEliece mceliece6960119	5	13908	1047319	226	1047545
Classic McEliece mceliece8192128	5	14080	1357824	240	1358064
LIMA-CCA-2p-2048	5	18433	12289	7299	19588
LIMA-CPA-2p-2048	5	18433	12289	6915	19204
NTS-KEM (13, 136)	5	19890	1419704	253	1419957
LIMA-CCA-sp-2062	5	24745	16497	9787	26284
LIMA-CPA-sp-2062	5	24745	16497	9275	25772
Titanium CCA super	5	26944	26912	8352	35264
Lizard-CATEGORY5-N1088	5	34880	4587520	35904	4623424
Lizard-CATEGORY5-N1300	5	41664	2727936	42688	2770624
BIG QUAKE 5	5	41804	149800	492	150292
Ramstake 756839	5	189242	94637	96167	190804
RLCE-KEM-256A	5	1048176	742089	2023	744112
LOTUS-256	5	1630720	1470976	1800	1472776
RLCE-KEM-256B	5	1773271	1232001	2640	1234641
DAGS 5	5	2230272	11616	1616	13232

Table 4.21: NIST security category 5 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
Round2-nround2-nd-l5	5	165	691	818	1509
Round2-uround2-nd-l5	5	169	709	868	1577
SIKEp964	5	826	726	766	1492
LAKE III	5	40	826	826	1652
NTRU Prime ntrulpr4591761	5	1238	1047	1175	2222
LAC-CCA-256	5	2080	1056	2048	3104
Ouroboros-R-256	5	40	1112	2144	3256
NTRU Prime sntrup4591761	5	1600	1218	1047	2265
LOCKER III	5	1286	1246	1374	2620
SABER fire	5	3040	1312	1472	2784
LOCKER VI	5	1482	1442	1570	3012
Three Bears PapaBear	5	40	1584	1697	3281
Three Bears PapaBear Ephem	5	40	1584	1697	3281
CRYSTALS-KYBER 1024	5	3168	1440	1504	2944
KINDI-256-5-2-2	5	1712	1456	2496	3952
KCL AKCN-RLWE	5	1664	1696	2083	3779
KCL OKCN-RLWE	5	1664	1696	1995	3691
RQC-256	5	1835	1795	2574	4369
HILA5	5	1824	1824	2012	3836
NewHope-CCA-1024	5	3680	1824	2208	4032
NewHope-CPA-1024	5	1792	1824	2176	4000
Lepton-CCA Moderate IV	5	2126	2052	4021	6073
Lepton.CPA Moderate IV	5	74	2052	3989	6041
DING Key Exchange 1024	3/5	3072	2064	2176	4240
LOCKER IX	5	2238	2198	2326	4524
DME-288	5	288	2304	288	2592
KINDI-512-3-2-1	5	2752	2368	3328	5696
BIKE-2 5	5	548	4096	4096	8192
NTRUEncrypt-1024	5	8194	4097	4097	8194
QC-MDPC KEM	5	548	4097	8226	12323
Lepton-CCA Paranoid I	5	4198	4128	5335	9463
Lepton-CCA Paranoid II	5	4208	4128	5589	9717
Lepton.CPA Paranoid I	5	70	4128	5303	9431
Lepton.CPA Paranoid II	5	80	4128	5557	9685
HQC Paranoiac I	5	7457	7417	14818	22235
HQC Paranoiac II	5	8029	7989	15962	23951
BIKE-1 5	5	548	8188	8188	16376
RLizard-CATEGORY5	5	769	8192	8256	16448
HQC Paranoiac III	5	8543	8503	16990	25493
Round2-uround2-n1-fn0-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn1-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn2-l5	5	1572	8679	8710	17389

HQC Paranoiac IV	5	8937	8897	17778	26675
BIKE-3 5	5	566	9034	9034	18068
DAGS 5	5	2230272	11616	1616	13232
LIMA-CCA-2p-2048	5	18433	12289	7299	19588
LIMA-CPA-2p-2048	5	18433	12289	6915	19204
LEDAkem-5-2	5	1244	12384	12384	24768
LIMA-CCA-sp-2062	5	24745	16497	9787	26284
LIMA-CPA-sp-2062	5	24745	16497	9275	25772
LEDAkem-5-3	5	1548	18016	9008	27024
LEDAkem-5-4	5	1772	22704	7568	30272
Titanium CCA super	5	26944	26912	8352	35264
Ramstake 756839	5	189242	94637	96167	190804
BIG QUAKE 5	5	41804	149800	492	150292
Mersenne-756839	5	32	189248	160160	349408
RLCE-KEM-256A	5	1048176	742089	2023	744112
Classic McEliece mceliece6960119	5	13908	1047319	226	1047545
RLCE-KEM-256B	5	1773271	1232001	2640	1234641
Classic McEliece mceliece8192128	5	14080	1357824	240	1358064
NTS-KEM (13, 136)	5	19890	1419704	253	1419957
LOTUS-256	5	1630720	1470976	1800	1472776
Lizard-CATEGORY5-N1300	5	41664	2727936	42688	2770624
Lizard-CATEGORY5-N1088	5	34880	4587520	35904	4623424

Table 4.22: NIST security category 5 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
Classic McEliece mceliece6960119	5	13908	1047319	226	1047545
Classic McEliece mceliece8192128	5	14080	1357824	240	1358064
NTS-KEM (13, 136)	5	19890	1419704	253	1419957
DME-288	5	288	2304	288	2592
BIG QUAKE 5	5	41804	149800	492	150292
SIKEp964	5	826	726	766	1492
Round2-nround2-nd-l5	5	165	691	818	1509
LAKE III	5	40	826	826	1652
Round2-uround2-nd-l5	5	169	709	868	1577
NTRU Prime sntrup4591761	5	1600	1218	1047	2265
NTRU Prime ntrulpr4591761	5	1238	1047	1175	2222
LOCKER III	5	1286	1246	1374	2620
SABER fire	5	3040	1312	1472	2784
CRYSTALS-KYBER 1024	5	3168	1440	1504	2944
LOCKER VI	5	1482	1442	1570	3012
DAGS 5	5	2230272	11616	1616	13232
Three Bears PapaBear	5	40	1584	1697	3281
Three Bears PapaBear Ephem	5	40	1584	1697	3281
LOTUS-256	5	1630720	1470976	1800	1472776
KCL OKCN-RLWE	5	1664	1696	1995	3691
HILA5	5	1824	1824	2012	3836
RLCE-KEM-256A	5	1048176	742089	2023	744112
LAC-CCA-256	5	2080	1056	2048	3104
KCL AKCN-RLWE	5	1664	1696	2083	3779
Ouroboros-R-256	5	40	1112	2144	3256
DING Key Exchange 1024	3/5	3072	2064	2176	4240
NewHope-CPA-1024	5	1792	1824	2176	4000
NewHope-CCA-1024	5	3680	1824	2208	4032
LOCKER IX	5	2238	2198	2326	4524
KINDI-256-5-2-2	5	1712	1456	2496	3952
RQC-256	5	1835	1795	2574	4369
RLCE-KEM-256B	5	1773271	1232001	2640	1234641
KINDI-512-3-2-1	5	2752	2368	3328	5696
Lepton.CPA Moderate IV	5	74	2052	3989	6041
Lepton-CCA Moderate IV	5	2126	2052	4021	6073
BIKE-2 5	5	548	4096	4096	8192
NTRUEncrypt-1024	5	8194	4097	4097	8194
Lepton.CPA Paranoid I	5	70	4128	5303	9431
Lepton-CCA Paranoid I	5	4198	4128	5335	9463
Lepton.CPA Paranoid II	5	80	4128	5557	9685
Lepton-CCA Paranoid II	5	4208	4128	5589	9717
LIMA-CPA-2p-2048	5	18433	12289	6915	19204

LIMA-CCA-2p-2048	5	18433	12289	7299	19588
LEDAkem-5-4	5	1772	22704	7568	30272
BIKE-1 5	5	548	8188	8188	16376
QC-MDPC KEM	5	548	4097	8226	12323
RLizard-CATEGORY5	5	769	8192	8256	16448
Titanium CCA super	5	26944	26912	8352	35264
Round2-uround2-n1-fn0-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn1-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn2-l5	5	1572	8679	8710	17389
LEDAkem-5-3	5	1548	18016	9008	27024
BIKE-3 5	5	566	9034	9034	18068
LIMA-CPA-sp-2062	5	24745	16497	9275	25772
LIMA-CCA-sp-2062	5	24745	16497	9787	26284
LEDAkem-5-2	5	1244	12384	12384	24768
HQC Paranoiac I	5	7457	7417	14818	22235
HQC Paranoiac II	5	8029	7989	15962	23951
HQC Paranoiac III	5	8543	8503	16990	25493
HQC Paranoiac IV	5	8937	8897	17778	26675
Lizard-CATEGORY5-N1088	5	34880	4587520	35904	4623424
Lizard-CATEGORY5-N1300	5	41664	2727936	42688	2770624
Ramstake 756839	5	189242	94637	96167	190804
Mersenne-756839	5	32	189248	160160	349408

Table 4.23: NIST security category 5 KEM implementation security levels, key lengths, ciphertext lengths, and the combined length of public keys and ciphertexts, sorted after space requirements for the combined length of public keys and ciphertexts.

Submission Implementation	Sec	$k_{private}$[B]	k_{public}[B]	c[B]	pk+c[B]
SIKEp964	5	826	726	766	1492
Round2-nround2-nd-l5	5	165	691	818	1509
Round2-uround2-nd-l5	5	169	709	868	1577
LAKE III	5	40	826	826	1652
NTRU Prime ntrulpr4591761	5	1238	1047	1175	2222
NTRU Prime sntrup4591761	5	1600	1218	1047	2265
DME-288	5	288	2304	288	2592
LOCKER III	5	1286	1246	1374	2620
SABER fire	5	3040	1312	1472	2784
CRYSTALS-KYBER 1024	5	3168	1440	1504	2944
LOCKER VI	5	1482	1442	1570	3012
LAC-CCA-256	5	2080	1056	2048	3104
Ouroboros-R-256	5	40	1112	2144	3256
Three Bears PapaBear	5	40	1584	1697	3281
Three Bears PapaBear Ephem	5	40	1584	1697	3281
KCL OKCN-RLWE	5	1664	1696	1995	3691
KCL AKCN-RLWE	5	1664	1696	2083	3779
HILA5	5	1824	1824	2012	3836
KINDI-256-5-2-2	5	1712	1456	2496	3952
NewHope-CPA-1024	5	1792	1824	2176	4000
NewHope-CCA-1024	5	3680	1824	2208	4032
DING Key Exchange 1024	3/5	3072	2064	2176	4240
RQC-256	5	1835	1795	2574	4369
LOCKER IX	5	2238	2198	2326	4524
KINDI-512-3-2-1	5	2752	2368	3328	5696
Lepton.CPA Moderate IV	5	74	2052	3989	6041
Lepton-CCA Moderate IV	5	2126	2052	4021	6073
BIKE-2 5	5	548	4096	4096	8192
NTRUEncrypt-1024	5	8194	4097	4097	8194
Lepton.CPA Paranoid I	5	70	4128	5303	9431
Lepton-CCA Paranoid I	5	4198	4128	5335	9463
Lepton.CPA Paranoid II	5	80	4128	5557	9685
Lepton-CCA Paranoid II	5	4208	4128	5589	9717
QC-MDPC KEM	5	548	4097	8226	12323
DAGS 5	5	2230272	11616	1616	13232
BIKE-1 5	5	548	8188	8188	16376
RLizard-CATEGORY5	5	769	8192	8256	16448
Round2-uround2-n1-fn0-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn1-l5	5	1572	8679	8710	17389
Round2-uround2-n1-fn2-l5	5	1572	8679	8710	17389
BIKE-3 5	5	566	9034	9034	18068
LIMA-CPA-2p-2048	5	18433	12289	6915	19204

LIMA-CCA-2p-2048	5	18433	12289	7299	19588
HQC Paranoiac I	5	7457	7417	14818	22235
HQC Paranoiac II	5	8029	7989	15962	23951
LEDAkem-5-2	5	1244	12384	12384	24768
HQC Paranoiac III	5	8543	8503	16990	25493
LIMA-CPA-sp-2062	5	24745	16497	9275	25772
LIMA-CCA-sp-2062	5	24745	16497	9787	26284
HQC Paranoiac IV	5	8937	8897	17778	26675
LEDAkem-5-3	5	1548	18016	9008	27024
LEDAkem-5-4	5	1772	22704	7568	30272
Titanium CCA super	5	26944	26912	8352	35264
BIG QUAKE 5	5	41804	149800	492	150292
Ramstake 756839	5	189242	94637	96167	190804
Mersenne-756839	5	32	189248	160160	349408
RLCE-KEM-256A	5	1048176	742089	2023	744112
Classic McEliece mceliece6960119	5	13908	1047319	226	1047545
RLCE-KEM-256B	5	1773271	1232001	2640	1234641
Classic McEliece mceliece8192128	5	14080	1357824	240	1358064
NTS-KEM (13, 136)	5	19890	1419704	253	1419957
LOTUS-256	5	1630720	1470976	1800	1472776
Lizard-CATEGORY5-N1300	5	41664	2727936	42688	2770624
Lizard-CATEGORY5-N1088	5	34880	4587520	35904	4623424

4.1.3 Signatures Space Requirements

For all Tables containing space requirements for signature implementations, *sec* denotes the claimed NIST level of security for the implementation (See Section 2.5.1), and $k_{private}$ (private key), *pk* (public key), and signature entries all denote length in bytes. If left blank, the implementation does not fulfil any of the NIST security levels' requirements, or lacks information about the given property.

Table 4.24 is a Table containing all signature algorithm implementations, and is sorted alphabetically after submission implementation names. This Table is meant as a referencing and look-up Table for all implementations. Following this Table, there are $3 \cdot 3$ Tables containing the same algorithm implementations, divided into the NIST security levels 1 and 2, 3 and 4, and 5. There are 3 different Tables for each level, each sorted after space requirements each of the previously mentioned property lengths. Below is an overview of the Tables for each level.

- Levels 1 and 2:
  - Table 4.25, sorted by private key length.
  - Table 4.26, sorted by public key length.
  - Table 4.27, sorted by ciphertext length.
- Levels 3 and 4:
  - Table 4.28, sorted by private key length.
  - Table 4.29, sorted by public key length.
  - Table 4.30, sorted by ciphertext length.
- Level 5:
  - Table 4.31, sorted by private key length.
  - Table 4.32, sorted by public key length.
  - Table 4.33, sorted by ciphertext length.

Table 4.24: Signature implementation security levels, key lengths and signature lengths, sorted alphabetically after submission implementation names.

Submission Implementation	Sec	$k_{private}$[B]	k_{public}[B]	Signature[B]
CRYSTALS-DILITHIUM weak	-	2393	896	1487
CRYSTALS-DILITHIUM medium	1	2800	1184	2044
CRYSTALS-DILITHIUM high	2	3504	1472	2701
CRYSTALS-DILITHIUM very high	3	3856	1760	3366
DRS 128	1	51274	5094433	8550
DRS 192	3	84060	8410001	11020
DRS 256	5	144527	14402026	14421
DualModeMS 128	1	18038184	528	32002
DualModeMS 192	3	-	1560	79315
DualModeMS 256	5	-	2112	149029
DualModeMS Inner 128	1	-	1139060	35
DualModeMS Inner 192	3	-	4243730	53
DualModeMS Inner 256	5	-	10635320	72
FALCON 512	1	4097	897	690
FALCON 768	2/3	6145	1441	1077
FALCON 1024	4/5	8193	1793	1330
GeMSS 128	1	14208	417408	48
GeMSS 192	3	39440	1304192	88
GeMSS 256	5	82056	3603792	104
Gravity-SPHINCS S	2	65568	32	12540
Gravity-SPHINCS M	2	2097184	32	28929
Gravity-SPHINCS L	2	1048608	32	35168
Gui-184	1	19100	416300	45
Gui-312	3	59300	1955100	63
Gui-448	5	155900	5789200	83
HiMQ-3	1	12074	128744	75
HiMQ-3F	1	14878	100878	67
HiMQ-3P	1	32	100878	67
LUOV-8-63-256	2	32	15500	319
LUOV-8-90-351	4	32	45000	441
LUOV-8-117-404	5	32	98600	521
LUOV-49-49-242	2	32	7300	1700
LUOV-64-68-330	4	32	19500	3100
LUOV-80-86-399	5	32	39300	4700
MQDSS-48	2	32	62	32882
MQDSS-64	4	48	88	67800
Picnic-L1-FS	1	16	32	34000
Picnic-L1-UR	1	16	32	53929
Picnic-L3-FS	3	24	48	76740
Picnic-L3-UR	3	24	48	121813
Picnic-L5-FS	5	32	64	132824
Picnic-L5-UR	5	32	64	209474
PQRSA-SIGN-15	-	98304	32768	32800

PQRSA-SIGN-20	-	3145728	1048576	1048608
PQRSA-SIGN-25	-	100663296	33554432	33554464
PQRSA-SIGN-30	2	3221225472	1073741824	1073741856
pqNTRUsign Gaussian-1024	5	2503	2065	2065
pqNTRUsign Uniform-1024	5	2604	2065	2065
pqsigRM-4-12	1	1382118	336804	337064
pqsigRM-6-12	3	334006	501176	501692
pqsigRM-6-13	5	2144166	2105344	2106372
qTESLA-128	1	1856	2976	2720
qTESLA-192	3	4160	6176	5664
qTESLA-256	5	4128	6432	5920
RaCoSS	1	173056	305	203980
Rainbow Ia	1	100209	152097	64
Rainbow Ib	1	114308	163185	78
Rainbow Ic	1	143385	192241	104
Rainbow IIIb	3	409463	564535	112
Rainbow IIIc	3	537781	720793	156
Rainbow IVa	4	376141	565489	92
Rainbow Vc	5	1274317	1723681	204
Rainbow VIa	5	892079	1351361	118
Rainbow VIb	5	1016868	1456225	147
SPHINCS+ haraka-128s	1	64	32	8080
SPHINCS+ haraka-128f	1	64	32	16976
SPHINCS+ haraka-192s	3	96	48	17064
SPHINCS+ haraka-192f	3	96	48	35664
SPHINCS+ haraka-256s	5	128	64	29792
SPHINCS+ haraka-256f	5	128	64	49216
SPHINCS+ SHA256-128s	1	64	32	8080
SPHINCS+ SHA256-128f	1	64	32	16976
SPHINCS+ SHA256-192s	3	96	48	17064
SPHINCS+ SHA256-192f	3	96	48	35664
SPHINCS+ SHA256-256s	5	128	64	29792
SPHINCS+ SHA256-256f	5	128	64	49216
SPHINCS+ shake256-128s	1	64	32	8080
SPHINCS+ shake256-128f	1	64	32	16976
SPHINCS+ shake256-192s	3	96	48	17064
SPHINCS+ shake256-192f	3	96	48	35664
SPHINCS+ shake256-256s	5	128	64	29792
SPHINCS+ shake256-256f	5	128	64	49216
WalnutDSA BKL-128	1	136	83	1100
WalnutDSA BKL-256	5	291	128	1800
WalnutDSA STOC-128	1	136	83	1200
WalnutDSA STOC-256	5	291	128	2100
WalnutDSA STOC-wo-DEH-128	1	291	128	2000
WalnutDSA STOC-wo-DEH-256	5	136	83	3400

Table 4.25: NIST security category 1 and 2 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for private keys ($\mathbf{k_{private}}$).

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	Signature[B]
Picnic-L1-FS	1	16	32	34000
Picnic-L1-UR	1	16	32	53929
LUOV-49-49-242	2	32	7300	1700
LUOV-8-63-256	2	32	15500	319
HiMQ-3P	1	32	100878	67
MQDSS-48	2	32	62	32882
SPHINCS+ haraka-128s	1	64	32	8080
SPHINCS+ haraka-128f	1	64	32	16976
SPHINCS+ SHA256-128s	1	64	32	8080
SPHINCS+ SHA256-128f	1	64	32	16976
SPHINCS+ shake256-128s	1	64	32	8080
SPHINCS+ shake256-128f	1	64	32	16976
WalnutDSA BKL-128	1	136	83	1100
WalnutDSA STOC-128	1	136	83	1200
WalnutDSA STOC-wo-DEH-128	1	291	128	2000
qTESLA-128	1	1856	2976	2720
CRYSTALS-DILITHIUM medium	1	2800	1184	2044
CRYSTALS-DILITHIUM high	2	3504	1472	2701
FALCON 512	1	4097	897	690
FALCON 768	2/3	6145	1441	1077
HiMQ-3	1	12074	128744	75
HiMQ-3F	1	14878	100878	67
Gui-184	1	19100	416300	45
GeMSS 128	1	14208	417408	48
DRS 128	1	51274	5094433	8550
Gravity-SPHINCS S	2	65568	32	12540
Rainbow Ia	1	100209	152097	64
Rainbow Ib	1	114308	163185	78
Rainbow Ic	1	143385	192241	104
RaCoSS	1	173056	305	203980
Gravity-SPHINCS L	2	1048608	32	35168
pqsigRM-4-12	1	1382118	336804	337064
Gravity-SPHINCS M	2	2097184	32	28929
DualModeMS 128	1	18038184	528	32002
PQRSA-SIGN-30	2	3221225472	1073741824	1073741856

Table 4.26: NIST security category 1 and 2 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**Signature[B]**
Gravity-SPHINCS S	2	65568	32	12540
Gravity-SPHINCS M	2	2097184	32	28929
Gravity-SPHINCS L	2	1048608	32	35168
Picnic-L1-FS	1	16	32	34000
Picnic-L1-UR	1	16	32	53929
SPHINCS+ haraka-128s	1	64	32	8080
SPHINCS+ haraka-128f	1	64	32	16976
SPHINCS+ SHA256-128s	1	64	32	8080
SPHINCS+ SHA256-128f	1	64	32	16976
SPHINCS+ shake256-128s	1	64	32	8080
SPHINCS+ shake256-128f	1	64	32	16976
MQDSS-48	2	32	62	32882
WalnutDSA BKL-128	1	136	83	1100
WalnutDSA STOC-128	1	136	83	1200
WalnutDSA STOC-wo-DEH-128	1	291	128	2000
RaCoSS	1	173056	305	203980
DualModeMS 128	1	18038184	528	32002
FALCON 512	1	4097	897	690
CRYSTALS-DILITHIUM medium	1	2800	1184	2044
FALCON 768	2/3	6145	1441	1077
CRYSTALS-DILITHIUM high	2	3504	1472	2701
qTESLA-128	1	1856	2976	2720
LUOV-49-49-242	2	32	7300	1700
LUOV-8-63-256	2	32	15500	319
HiMQ-3P	1	32	100878	67
HiMQ-3F	1	14878	100878	67
HiMQ-3	1	12074	128744	75
Rainbow Ia	1	100209	152097	64
Rainbow Ib	1	114308	163185	78
Rainbow Ic	1	143385	192241	104
pqsigRM-4-12	1	1382118	336804	337064
Gui-184	1	19100	416300	45
GeMSS 128	1	14208	417408	48
DualModeMS Inner 128	1	-	1139060	35
DRS 128	1	51274	5094433	8550
PQRSA-SIGN-30	2	3221225472	1073741824	1073741856

Table 4.27: NIST security category 1 and 2 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**Signature[B]**
DualModeMS Inner 128	1	-	1139060	35
Gui-184	1	19100	416300	45
GeMSS 128	1	14208	417408	48
Rainbow Ia	1	100209	152097	64
HiMQ-3P	1	32	100878	67
HiMQ-3F	1	14878	100878	67
HiMQ-3	1	12074	128744	75
Rainbow Ib	1	114308	163185	78
Rainbow Ic	1	143385	192241	104
LUOV-8-63-256	2	32	15500	319
FALCON 512	1	4097	897	690
FALCON 768	2/3	6145	1441	1077
WalnutDSA BKL-128	1	136	83	1100
WalnutDSA STOC-128	1	136	83	1200
LUOV-49-49-242	2	32	7300	1700
WalnutDSA STOC-wo-DEH-128	1	291	128	2000
CRYSTALS-DILITHIUM medium	1	2800	1184	2044
CRYSTALS-DILITHIUM high	2	3504	1472	2701
qTESLA-128	1	1856	2976	2720
SPHINCS+ haraka-128s	1	64	32	8080
SPHINCS+ SHA256-128s	1	64	32	8080
SPHINCS+ shake256-128s	1	64	32	8080
DRS 128	1	51274	5094433	8550
Gravity-SPHINCS S	2	65568	32	12540
SPHINCS+ haraka-128f	1	64	32	16976
SPHINCS+ SHA256-128f	1	64	32	16976
SPHINCS+ shake256-128f	1	64	32	16976
Gravity-SPHINCS M	2	2097184	32	28929
DualModeMS 128	1	18038184	528	32002
MQDSS-48	2	32	62	32882
Picnic-L1-FS	1	16	32	34000
Gravity-SPHINCS L	2	1048608	32	35168
Picnic-L1-UR	1	16	32	53929
RaCoSS	1	173056	305	203980
pqsigRM-4-12	1	1382118	336804	337064
PQRSA-SIGN-30	2	3221225472	1073741824	1073741856

Table 4.28: NIST security category 3 and 4 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**Signature[B]**
Picnic-L3-FS	3	24	48	76740
Picnic-L3-UR	3	24	48	121813
LUOV-8-90-351	4	32	45000	441
LUOV-64-68-330	4	32	19500	3100
MQDSS-64	4	48	88	67800
SPHINCS+ haraka-192s	3	96	48	17064
SPHINCS+ haraka-192f	3	96	48	35664
SPHINCS+ SHA256-192s	3	96	48	17064
SPHINCS+ SHA256-192f	3	96	48	35664
SPHINCS+ shake256-192s	3	96	48	17064
SPHINCS+ shake256-192f	3	96	48	35664
CRYSTALS-DILITHIUM very high	3	3856	1760	3366
qTESLA-192	3	4160	6176	5664
FALCON 768	2/3	6145	1441	1077
FALCON 1024	4/5	8193	1793	1330
GeMSS 192	3	39440	1304192	88
Gui-312	3	59300	1955100	63
DRS 192	3	84060	8410001	11020
pqsigRM-6-12	3	334006	501176	501692
Rainbow IVa	4	376141	565489	92
Rainbow IIIb	3	409463	564535	112
Rainbow IIIc	3	537781	720793	156

Table 4.29: NIST security category 3 and 4 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	Signature[B]
Picnic-L3-FS	3	24	48	76740
Picnic-L3-UR	3	24	48	121813
MQDSS-64	4	48	88	67800
SPHINCS+ haraka-192s	3	96	48	17064
SPHINCS+ haraka-192f	3	96	48	35664
SPHINCS+ SHA256-192s	3	96	48	17064
SPHINCS+ SHA256-192f	3	96	48	35664
SPHINCS+ shake256-192s	3	96	48	17064
SPHINCS+ shake256-192f	3	96	48	35664
FALCON 768	2/3	6145	1441	1077
DualModeMS 192	3	-	1560	79315
CRYSTALS-DILITHIUM very high	3	3856	1760	3366
FALCON 1024	4/5	8193	1793	1330
qTESLA-192	3	4160	6176	5664
LUOV-64-68-330	4	32	19500	3100
LUOV-8-90-351	4	32	45000	441
pqsigRM-6-12	3	334006	501176	501692
Rainbow IVa	4	376141	565489	92
Rainbow IIIb	3	409463	564535	112
Rainbow IIIc	3	537781	720793	156
GeMSS 192	3	39440	1304192	88
Gui-312	3	59300	1955100	63
DualModeMS Inner 192	3	-	4243730	53
DRS 192	3	84060	8410001	11020

Table 4.30: NIST security category 3 and 4 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**Signature[B]**
DualModeMS Inner 192	3	-	4243730	53
Gui-312	3	59300	1955100	63
GeMSS 192	3	39440	1304192	88
Rainbow IVa	4	376141	565489	92
Rainbow IIIb	3	409463	564535	112
Rainbow IIIc	3	537781	720793	156
LUOV-8-90-351	4	32	45000	441
FALCON 768	2/3	6145	1441	1077
FALCON 1024	4/5	8193	1793	1330
CRYSTALS-DILITHIUM very high	3	3856	1760	3366
LUOV-64-68-330	4	32	19500	3100
qTESLA-192	3	4160	6176	5664
DRS 192	3	84060	8410001	11020
SPHINCS+ haraka-192s	3	96	48	17064
SPHINCS+ SHA256-192s	3	96	48	17064
SPHINCS+ shake256-192s	3	96	48	17064
SPHINCS+ haraka-192f	3	96	48	35664
SPHINCS+ SHA256-192f	3	96	48	35664
SPHINCS+ shake256-192f	3	96	48	35664
MQDSS-64	4	48	88	67800
Picnic-L3-FS	3	24	48	76740
DualModeMS 192	3	-	1560	79315
Picnic-L3-UR	3	24	48	121813
pqsigRM-6-12	3	334006	501176	501692

Table 4.31: NIST security category 5 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for private keys $\mathbf{k_{private}}$.

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	Signature[B]
LUOV-8-117-404	5	32	98600	521
LUOV-80-86-399	5	32	39300	4700
Picnic-L5-FS	5	32	64	132824
Picnic-L5-UR	5	32	64	209474
SPHINCS+ haraka-256s	5	128	64	29792
SPHINCS+ haraka-256f	5	128	64	49216
SPHINCS+ SHA256-256s	5	128	64	29792
SPHINCS+ SHA256-256f	5	128	64	49216
SPHINCS+ shake256-256s	5	128	64	29792
SPHINCS+ shake256-256f	5	128	64	49216
WalnutDSA STOC-wo-DEH-256	5	136	83	3400
WalnutDSA BKL-256	5	291	128	1800
WalnutDSA STOC-256	5	291	128	2100
pqNTRUsign Gaussian-1024	5	2503	2065	2065
pqNTRUsign Uniform-1024	5	2604	2065	2065
qTESLA-256	5	4128	6432	5920
FALCON 1024	4/5	8193	1793	1330
GeMSS 256	5	82056	3603792	104
DRS 256	5	144527	14402026	14421
Gui-448	5	155900	5789200	83
Rainbow VIa	5	892079	1351361	118
Rainbow VIb	5	1016868	1456225	147
Rainbow Vc	5	1274317	1723681	204
pqsigRM-6-13	5	2144166	2105344	2106372

Table 4.32: NIST security category 5 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for public keys ($\mathbf{k_{public}}$).

Submission Implementation	Sec	$\mathbf{k_{private}}$[B]	$\mathbf{k_{public}}$[B]	Signature[B]
Picnic-L5-FS	5	32	64	132824
Picnic-L5-UR	5	32	64	209474
SPHINCS+ haraka-256s	5	128	64	29792
SPHINCS+ haraka-256f	5	128	64	49216
SPHINCS+ SHA256-256s	5	128	64	29792
SPHINCS+ SHA256-256f	5	128	64	49216
SPHINCS+ shake256-256s	5	128	64	29792
SPHINCS+ shake256-256f	5	128	64	49216
WalnutDSA STOC-wo-DEH-256	5	136	83	3400
WalnutDSA BKL-256	5	291	128	1800
WalnutDSA STOC-256	5	291	128	2100
FALCON 1024	4/5	8193	1793	1330
pqNTRUsign Gaussian-1024	5	2503	2065	2065
pqNTRUsign Uniform-1024	5	2604	2065	2065
DualModeMS 256	5	-	2112	149029
qTESLA-256	5	4128	6432	5920
LUOV-80-86-399	5	32	39300	4700
LUOV-8-117-404	5	32	98600	521
Rainbow VIa	5	892079	1351361	118
Rainbow VIb	5	1016868	1456225	147
Rainbow Vc	5	1274317	1723681	204
pqsigRM-6-13	5	2144166	2105344	2106372
GeMSS 256	5	82056	3603792	104
Gui-448	5	155900	5789200	83
DualModeMS Inner 256	5	-	10635320	72
DRS 256	5	144527	14402026	14421

Table 4.33: NIST security category 5 signature implementation security levels, key lengths and ciphertext lengths, sorted after space requirements for ciphertexts.

Submission Implementation	**Sec**	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**Signature[B]**
DualModeMS Inner 256	5	-	10635320	72
Gui-448	5	155900	5789200	83
GeMSS 256	5	82056	3603792	104
Rainbow VIa	5	892079	1351361	118
Rainbow VIb	5	1016868	1456225	147
Rainbow Vc	5	1274317	1723681	204
LUOV-8-117-404	5	32	98600	521
FALCON 1024	4/5	8193	1793	1330
WalnutDSA BKL-256	5	291	128	1800
pqNTRUsign Gaussian-1024	5	2503	2065	2065
pqNTRUsign Uniform-1024	5	2604	2065	2065
WalnutDSA STOC-256	5	291	128	2100
WalnutDSA STOC-wo-DEH-256	5	136	83	3400
LUOV-80-86-399	5	32	39300	4700
qTESLA-256	5	4128	6432	5920
DRS 256	5	144527	14402026	14421
SPHINCS+ haraka-256s	5	128	64	29792
SPHINCS+ SHA256-256s	5	128	64	29792
SPHINCS+ shake256-256s	5	128	64	29792
SPHINCS+ haraka-256f	5	128	64	49216
SPHINCS+ SHA256-256f	5	128	64	49216
SPHINCS+ shake256-256f	5	128	64	49216
Picnic-L5-FS	5	32	64	132824
DualModeMS 256	5	-	2112	149029
Picnic-L5-UR	5	32	64	209474
pqsigRM-6-13	5	2144166	2105344	2106372

4.2 Execution Times

This section contains execution times for key generation, encryption, decryption, encapsulation, decapsulation, signature generation, and verification for all implementations. Each of the subsections 4.2.1, 4.2.2, and 4.2.3 contains all Tables for running times for all encryption algorithms implementations, KEM algorithms implementations, and signature algorithm implementations, respectively. All numbers are retrieved from the original submission documentation and reference implementation for each submission. For references, see Chapter 3.

Execution times are given as CPU cycles per operation (encryption, decryption, key generation, etc.).

Where only time for execution is given, cycles are attempted to be calculated using the given information about the computer, namely Hz and core number. It is worth noting that these numbers are estimations only, as calculating the cycles per seconds for each processor is not an accurate measurement. These numbers are therefore marked with an asterisk (*). All numbers calculated using numbers marked with an asterisk are subsequently also marked as such.

4.2.1 Encryption Running Times

For all Tables containing running times for encryption implementations, *sec* denotes the claimed NIST level of security for the implementation (See Section 2.5.1), and *Key.Gen* (key generation), encrypt, and decrypt entries all denote number of cycles needed to complete each process. If left blank, the implementation does not fulfil any of the NIST security levels' requirements, or the data is not available.

Table 4.34 is a Table containing all encryption algorithm implementations, and is sorted alphabetically after submission implementation names. This Table is meant as a referencing and look-up Table for all implementations. Following this Table, there are $3 \cdot 3$ Tables containing the same algorithm implementations, divided into the NIST security levels 1 and 2, 3 and 4, and 5. There are 3 different Tables for each level, each sorted after space requirements each of the previously mentioned property lengths. Below is an overview of the Tables for each level. Note that all encryption and decryption is compared at 32-bit message lengths unless otherwise specified.

- Levels 1 and 2:
  - Table 4.35, sorted by the number of needed cycles for key generation.
  - Table 4.36, sorted by the number of needed cycles for encryption.
  - Table 4.37, sorted by the number of needed cycles for decryption.
- Levels 3 and 4:
  - Table 4.38, sorted by the number of needed cycles for key generation.
  - Table 4.39, sorted by the number of needed cycles for encryption.
  - Table 4.40, sorted by the number of needed cycles for decryption.

- Level 5:
  - Table 4.41, sorted by the number of needed cycles for key generation.
  - Table 4.42, sorted by the number of needed cycles for encryption.
  - Table 4.43, sorted by the number of needed cycles for decryption.

Table 4.34: Encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted alphabetically after submission implementation names.

Submission Implementation	Sec	Key.Gen	Encrypt	Decrypt
Giophantus 602	1	92909566	178456036	335353573
Giophantus 868	3	160497017	378860493	716243384
Giophantus 1134	5	239510004	626677271	1186128486
Guess Again	5	-	5324800000*	-
KCL AKCN-MLWE-CCA	4	481881	568461	630891
KINDI-256-3-4-2	2	203096	247793	312211
KINDI-256-5-2-2	5	519010	595043	701763
KINDI-512-2-2-2	4	214064	280420	377962
KINDI-512-2-4-1	4	215542	285832	382958
KINDI-512-3-2-1	5	723922	530173	672720
LAC-CPA-128	1	90686	152575	68285
LAC-CPA-192	3	309216	410469	238268
LAC-CPA-256	5	269827	513753	336207
LEDApkc-1-2	1	129344098	8597636	50484279
LEDApkc-1-3	1	60872340	8268701	61401087
LEDApkc-1-4	1	48451595	10481941	69485795
LEDApkc-3-2	3	502395419	31957109	136053636
LEDApkc-3-3	3	263377457	34482262	140245281
LEDApkc-3-4	3	215603578	42121566	151476731
LEDApkc-5-2	5	1599125506	92839822	264937652
LEDApkc-5-3	5	802604872	99518907	262949682
LEDApkc-5-4	5	560609350	107088891	366509289
LIMA-CCA-sp-1018	1	1429742	1239122	1610046
LIMA-CCA-sp-1306	2	2600237	2360157	3091742
LIMA-CCA-sp-1822	3	2853857	2513027	3264289
LIMA-CCA-sp-2062	5	5114770	4728991	6235608
LIMA-CCA-2p-1024	3	654921	482597	615220
LIMA-CCA-2p-2048	5	1325909	977969	1242139
LIMA-CPA-sp-1018	1	1429742	1236494	395944
LIMA-CPA-sp-1306	2	2600237	2354094	770324
LIMA-CPA-sp-1822	3	2853857	2506669	813067
LIMA-CPA-sp-2062	5	5114770	4726471	1549057
LIMA-CPA-2p-1024	3	654921	480700	156880
LIMA-CPA-2p-2048	5	1325909	971660	310484
Lizard-CATEGORY1-N536	1	105700625	326246	144127
Lizard-CATEGORY1-N663	1	119955905	318857	146639
Lizard-CATEGORY3-N816	3	180072866	375555	265405
Lizard-CATEGORY3-N925	3	273990136	533156	867432
RLizard-CATEGORY1	1	914860	347269	96689
RLizard-CATEGORY3-N1024	3	1492114	422050	183654
RLizard-CATEGORY3-N2048	3	1244631	774805	237206
RLizard-CATEGORY5	5	1084552	804982	568140

LOTUS-128	1	26820400	316252	382582
LOTUS-192	3	46658849	443667	587417
LOTUS-256	5	72229496	626112	882523
McNie-3Q-128-1	1	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	5980389
McNie-3Q-128-2	1	$1.84466{\cdot}10^{17}$	4959808	6686612
McNie-3Q-192-1	3	$1.84466{\cdot}10^{17}$	6053836	8257663
McNie-3Q-192-2	3	$1.84466{\cdot}10^{17}$	7161772	9664007
McNie-3Q-256-1	5	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	11301899
McNie-3Q-256-2	5	$1.84466{\cdot}10^{17}$	11992894	15703174
McNie-4Q-128-1	1	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	4229606
McNie-4Q-128-2	1	$1.84466{\cdot}10^{17}$	2479921	5236511
McNie-4Q-192-1	3	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	6349456
McNie-4Q-192-2	3	$1.84466{\cdot}10^{17}$	3327255	7504623
McNie-4Q-256-1	5	$1.84466{\cdot}10^{17}$	3503615	7707171
McNie-4Q-256-2	5	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	9416644
NTRUEncrypt-443	1	2454400*	436800*	566800*
NTRUEncrypt-734	4	5148000*	629200*	1014000*
NTRUEncrypt-1024	5	224640000*	348400000*	598000000*
Odd Manhatten-128	1	201062400*	71794800*	77884400*
Odd Manhatten-192	3	327738400*	138855200*	152196000*
Odd Manhatten-256	5	593124400*	283250000*	310604800*
pqrsa-ENCRYPT-30	2	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$
Titanium CPA toy	-	1458062	1061298	176191
Titanium CPA lite	-	1507082	1317313	218207
Titanium CPA standard	1	1981835	1508258	261583
Titanium CPA medium	1	2221874	2009472	301930
Titanium CPA high	3	2179991	2041861	376220
Titanium CPA super	5	3054311	2917708	534948

Table 4.35: NIST security category 1 and 2 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for key generation.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
LAC-CPA-128	1	90686	152575	68285
KINDI-256-3-4-2	2	203096	247793	312211
RLizard-CATEGORY1	1	914860	347269	96689
LIMA-CCA-sp-1018	1	1429742	1239122	1610046
LIMA-CPA-sp-1018	1	1429742	1236494	395944
Titanium CPA standard	1	1981835	1508258	261583
Titanium CPA medium	1	2221874	2009472	301930
NTRUEncrypt-443	1	2454400*	436800*	566800*
LIMA-CCA-sp-1306	2	2600237	2360157	3091742
LIMA-CPA-sp-1306	2	2600237	2354094	770324
LOTUS-128	1	26820400	316252	382582
LEDApkc-1-4	1	48451595	10481941	69485795
LEDApkc-1-3	1	60872340	8268701	61401087
Giophantus 602	1	92909566	178456036	335353573
Lizard-CATEGORY1-N536	1	105700625	326246	144127
Lizard-CATEGORY1-N663	1	119955905	318857	146639
LEDApkc-1-2	1	129344098	8597636	50484279
Odd Manhatten-128	1	201062400*	71794800*	77884400*
McNie-3Q-128-1	1	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	5980389
McNie-3Q-128-2	1	$1.84466 \cdot 10^{17}$	4959808	6686612
McNie-4Q-128-1	1	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	4229606
McNie-4Q-128-2	1	$1.84466 \cdot 10^{17}$	2479921	5236511
pqrsa-ENCRYPT-30	2	$5.87 \cdot 10^{14}$	$1.82 \cdot 10^{13}$	$2.23 \cdot 10^{13}$

Table 4.36: NIST security category 1 and 2 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for encryption.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
LAC-CPA-128	1	90686	152575	68285
KINDI-256-3-4-2	2	203096	247793	312211
LOTUS-128	1	26820400	316252	382582
Lizard-CATEGORY1-N663	1	119955905	318857	146639
Lizard-CATEGORY1-N536	1	105700625	326246	144127
RLizard-CATEGORY1	1	914860	347269	96689
NTRUEncrypt-443	1	2454400*	436800*	566800*
LIMA-CPA-sp-1018	1	1429742	1236494	395944
LIMA-CCA-sp-1018	1	1429742	1239122	1610046
Titanium CPA standard	1	1981835	1508258	261583
Titanium CPA medium	1	2221874	2009472	301930
LIMA-CPA-sp-1306	2	2600237	2354094	770324
LIMA-CCA-sp-1306	2	2600237	2360157	3091742
McNie-4Q-128-2	1	$1.84466 \cdot 10^{17}$	2479921	5236511
McNie-3Q-128-2	1	$1.84466 \cdot 10^{17}$	4959808	6686612
LEDApkc-1-3	1	60872340	8268701	61401087
LEDApkc-1-2	1	129344098	8597636	50484279
LEDApkc-1-4	1	48451595	10481941	69485795
Giophantus 602	1	92909566	178456036	335353573
Odd Manhatten-128	1	201062400*	71794800*	77884400*
McNie-3Q-128-1	1	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	5980389
McNie-4Q-128-1	1	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	4229606
pqrsa-ENCRYPT-30	2	$5.87 \cdot 10^{14}$	$1.82 \cdot 10^{13}$	$2.23 \cdot 10^{13}$

Table 4.37: NIST security category 1 and 2 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for decryption.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
LAC-CPA-128	1	90686	152575	68285
RLizard-CATEGORY1	1	914860	347269	96689
Lizard-CATEGORY1-N536	1	105700625	326246	144127
Lizard-CATEGORY1-N663	1	119955905	318857	146639
Titanium CPA standard	1	1981835	1508258	261583
Titanium CPA medium	1	2221874	2009472	301930
KINDI-256-3-4-2	2	203096	247793	312211
LOTUS-128	1	26820400	316252	382582
LIMA-CPA-sp-1018	1	1429742	1236494	395944
NTRUEncrypt-443	1	2454400*	436800*	566800*
LIMA-CPA-sp-1306	2	2600237	2354094	770324
LIMA-CCA-sp-1018	1	1429742	1239122	1610046
LIMA-CCA-sp-1306	2	2600237	2360157	3091742
McNie-4Q-128-1	1	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	4229606
McNie-4Q-128-2	1	$1.84466{\cdot}10^{17}$	2479921	5236511
McNie-3Q-128-1	1	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	5980389
McNie-3Q-128-2	1	$1.84466{\cdot}10^{17}$	4959808	6686612
LEDApkc-1-2	1	129344098	8597636	50484279
LEDApkc-1-3	1	60872340	8268701	61401087
LEDApkc-1-4	1	48451595	10481941	69485795
Odd Manhatten-128	1	201062400*	71794800*	77884400*
Giophantus 602	1	92909566	178456036	335353573
pqrsa-ENCRYPT-30	2	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$

Table 4.38: NIST security category 3 and 4 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for key generation.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
KINDI-512-2-2-2	4	214064	280420	377962
KINDI-512-2-4-1	4	215542	285832	382958
LAC-CPA-192	3	309216	410469	238268
KCL AKCN-MLWE-CCA	4	481881	568461	630891
LIMA-CCA-2p-1024	3	654921	482597	615220
LIMA-CPA-2p-1024	3	654921	480700	156880
RLizard-CATEGORY3-N2048	3	1244631	774805	237206
RLizard-CATEGORY3-N1024	3	1492114	422050	183654
Titanium CPA high	3	2179991	2041861	376220
LIMA-CCA-sp-1822	3	2853857	2513027	3264289
LIMA-CPA-sp-1822	3	2853857	2506669	813067
NTRUEncrypt-734	4	5148000*	629200*	1014000*
LOTUS-192	3	46658849	443667	587417
Giophantus 868	3	160497017	378860493	716243384
Lizard-CATEGORY3-N816	3	180072866	375555	265405
LEDApkc-3-4	3	215603578	42121566	151476731
LEDApkc-3-3	3	263377457	34482262	140245281
Lizard-CATEGORY3-N925	3	273990136	533156	867432
Odd Manhatten-192	3	327738400*	138855200*	152196000*
LEDApkc-3-2	3	502395419	31957109	136053636
McNie-3Q-192-1	3	$1.84466 \cdot 10^{17}$	6053836	8257663
McNie-3Q-192-2	3	$1.84466 \cdot 10^{17}$	7161772	9664007
McNie-4Q-192-1	3	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	6349456
McNie-4Q-192-2	3	$1.84466 \cdot 10^{17}$	3327255	7504623

Table 4.39: NIST security category 3 and 4 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for encryption.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
KINDI-512-2-2-2	4	214064	280420	377962
KINDI-512-2-4-1	4	215542	285832	382958
Lizard-CATEGORY3-N816	3	180072866	375555	265405
LAC-CPA-192	3	309216	410469	238268
RLizard-CATEGORY3-N1024	3	1492114	422050	183654
LOTUS-192	3	46658849	443667	587417
LIMA-CCA-2p-1024	3	654921	482597	615220
LIMA-CPA-2p-1024	3	654921	480700	156880
Lizard-CATEGORY3-N925	3	273990136	533156	867432
KCL AKCN-MLWE-CCA	4	481881	568461	630891
NTRUEncrypt-734	4	5148000*	629200*	1014000*
RLizard-CATEGORY3-N2048	3	1244631	774805	237206
Titanium CPA high	3	2179991	2041861	376220
LIMA-CCA-sp-1822	3	2853857	2513027	3264289
LIMA-CPA-sp-1822	3	2853857	2506669	813067
McNie-4Q-192-2	3	$1.84466{\cdot}10^{17}$	3327255	7504623
McNie-3Q-192-1	3	$1.84466{\cdot}10^{17}$	6053836	8257663
McNie-3Q-192-2	3	$1.84466{\cdot}10^{17}$	7161772	9664007
LEDApkc-3-2	3	502395419	31957109	136053636
LEDApkc-3-3	3	263377457	34482262	140245281
LEDApkc-3-4	3	215603578	42121566	151476731
Odd Manhatten-192	3	327738400*	138855200*	152196000*
Giophantus 868	3	160497017	378860493	716243384
McNie-4Q-192-1	3	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	6349456

Table 4.40: NIST security category 3 and 4 encryption implementation security levels,key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for decryption.

Submission Implementation	Sec	Key.Gen	Encrypt	Decrypt
LIMA-CPA-2p-1024	3	654921	480700	156880
RLizard-CATEGORY3-N1024	3	1492114	422050	183654
RLizard-CATEGORY3-N2048	3	1244631	774805	237206
LAC-CPA-192	3	309216	410469	238268
Lizard-CATEGORY3-N816	3	180072866	375555	265405
Titanium CPA high	3	2179991	2041861	376220
KINDI-512-2-2-2	4	214064	280420	377962
KINDI-512-2-4-1	4	215542	285832	382958
LOTUS-192	3	46658849	443667	587417
LIMA-CCA-2p-1024	3	654921	482597	615220
KCL AKCN-MLWE-CCA	4	481881	568461	630891
LIMA-CPA-sp-1822	3	2853857	2506669	813067
Lizard-CATEGORY3-N925	3	273990136	533156	867432
NTRUEncrypt-734	4	5148000*	629200*	1014000*
LIMA-CCA-sp-1822	3	2853857	2513027	3264289
McNie-4Q-192-1	3	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	6349456
McNie-4Q-192-2	3	$1.84466 \cdot 10^{17}$	3327255	7504623
McNie-3Q-192-1	3	$1.84466 \cdot 10^{17}$	6053836	8257663
McNie-3Q-192-2	3	$1.84466 \cdot 10^{17}$	7161772	9664007
LEDApkc-3-2	3	502395419	31957109	136053636
LEDApkc-3-3	3	263377457	34482262	140245281
LEDApkc-3-4	3	215603578	42121566	151476731
Odd Manhatten-192	3	327738400*	138855200*	152196000*
Giophantus 868	3	160497017	378860493	716243384

Table 4.41: NIST security category 5 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for key generation.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
LAC-CPA-256	5	269827	513753	336207
KINDI-256-5-2-2	5	519010	595043	701763
KINDI-512-3-2-1	5	723922	530173	672720
RLizard-CATEGORY5	5	1084552	804982	568140
LIMA-CCA-2p-2048	5	1325909	977969	1242139
LIMA-CPA-2p-2048	5	1325909	971660	310484
Titanium CPA super	5	3054311	2917708	534948
LIMA-CCA-sp-2062	5	5114770	4728991	6235608
LIMA-CPA-sp-2062	5	5114770	4726471	1549057
LOTUS-256	5	72229496	626112	882523
NTRUEncrypt-1024	5	224640000*	348400000*	598000000*
Giophantus 1134	5	239510004	626677271	1186128486
LEDApkc-5-4	5	560609350	107088891	366509289
Odd Manhatten-256	5	593124400*	283250000*	310604800*
LEDApkc-5-3	5	802604872	99518907	262949682
LEDApkc-5-2	5	1599125506	92839822	264937652
McNie-3Q-256-1	5	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	11301899
McNie-3Q-256-2	5	$1.84466 \cdot 10^{17}$	11992894	15703174
McNie-4Q-256-1	5	$1.84466 \cdot 10^{17}$	3503615	7707171
McNie-4Q-256-2	5	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	9416644

Table 4.42: NIST security category 5 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for encryption.

Submission Implementation	**Sec**	**Key.Gen**	**Encrypt**	**Decrypt**
LAC-CPA-256	5	269827	513753	336207
KINDI-256-5-2-2	5	519010	595043	701763
KINDI-512-3-2-1	5	723922	530173	672720
LOTUS-256	5	72229496	626112	882523
RLizard-CATEGORY5	5	1084552	804982	568140
LIMA-CPA-2p-2048	5	1325909	971660	310484
LIMA-CCA-2p-2048	5	1325909	977969	1242139
Titanium CPA super	5	3054311	2917708	534948
McNie-4Q-256-1	5	$1.84466{\cdot}10^{17}$	3503615	7707171
LIMA-CPA-sp-2062	5	5114770	4726471	1549057
LIMA-CCA-sp-2062	5	5114770	4728991	6235608
McNie-3Q-256-2	5	$1.84466{\cdot}10^{17}$	11992894	15703174
LEDApkc-5-2	5	1599125506	92839822	264937652
LEDApkc-5-3	5	802604872	99518907	262949682
LEDApkc-5-4	5	560609350	107088891	366509289
Odd Manhatten-256	5	593124400*	283250000*	310604800*
NTRUEncrypt-1024	5	224640000*	348400000*	598000000*
Giophantus 1134	5	239510004	626677271	1186128486
Guess Again	5	-	5324800000*	-
McNie-3Q-256-1	5	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	11301899
McNie-4Q-256-2	5	$1.84466{\cdot}10^{17}$	$1.84466{\cdot}10^{17}$	9416644

Table 4.43: NIST security category 5 encryption implementation security levels, key generation, encryption, and decryption given in number of needed CPU cycles, sorted after running times for decryption.

Submission Implementation	Sec	Key.Gen	Encrypt	Decrypt
LIMA-CPA-2p-2048	5	1325909	971660	310484
LAC-CPA-256	5	269827	513753	336207
Titanium CPA super	5	3054311	2917708	534948
RLizard-CATEGORY5	5	1084552	804982	568140
KINDI-512-3-2-1	5	723922	530173	672720
KINDI-256-5-2-2	5	519010	595043	701763
LOTUS-256	5	72229496	626112	882523
LIMA-CCA-2p-2048	5	1325909	977969	1242139
LIMA-CPA-sp-2062	5	5114770	4726471	1549057
LIMA-CCA-sp-2062	5	5114770	4728991	6235608
McNie-4Q-256-1	5	$1.84466 \cdot 10^{17}$	3503615	7707171
McNie-4Q-256-2	5	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	9416644
McNie-3Q-256-1	5	$1.84466 \cdot 10^{17}$	$1.84466 \cdot 10^{17}$	11301899
McNie-3Q-256-2	5	$1.84466 \cdot 10^{17}$	11992894	15703174
LEDApkc-5-3	5	802604872	99518907	262949682
LEDApkc-5-2	5	1599125506	92839822	264937652
Odd Manhatten-256	5	593124400*	283250000*	310604800*
LEDApkc-5-4	5	560609350	107088891	366509289
NTRUEncrypt-1024	5	224640000*	348400000*	598000000*
Giophantus 1134	5	239510004	626677271	1186128486

4.2.2 KEM Running Times

For all Tables containing running times for KEM implementations, *sec* denotes the claimed NIST level of security for the implementation (See Section 2.5.1), and *Key.Gen* (key generation), *encap* (encapsulation), *decap* (decapsulation), and *Sum*(key generation + encapsulation + decapsulation) entries all denote the number of cycles needed to complete each process. The latter of these properties is especially important, as this is the sum of all procedures which have to be performed by both parties when producing ephemeral keys. If left blank, the implementation does not fulfil any of the NIST security levels' requirements, or the data is not available.

Table 4.44 is a Table containing all KEM algorithm implementations, and is sorted alphabetically after submission implementation names. This Table is meant as a referencing and look-up Table for all implementations. Following this Table, there are $3 \cdot 4$ Tables containing the same algorithm implementations, divided into the NIST security levels 1 and 2, 3 and 4, and 5. There are 3 different Tables for each level, each sorted after space requirements each of the previously mentioned property lengths. Below is an overview of the Tables for each level.

- Levels 1 and 2:
  - Table 4.45, sorted by the number of needed cycles for key generation.
  - Table 4.46, sorted by the number of needed cycles for encapsulation.
  - Table 4.47, sorted by the number of needed cycles for decapsulation.
  - Table 4.48, sorted by the number of needed cycles for the sum of key generation + encapsulation + decapsulation.
- Levels 3 and 4:
  - Table 4.49, sorted by the number of needed cycles for key generation.
  - Table 4.50, sorted by the number of needed cycles for encapsulation.
  - Table 4.51, sorted by the number of needed cycles for decapsulation.
  - Table 4.52, sorted by the number of needed cycles for the sum of key generation + encapsulation + decapsulation.
- Level 5:
  - Table 4.53, sorted by the number of needed cycles for key generation.
  - Table 4.54, sorted by the number of needed cycles for encapsulation.
  - Table 4.55, sorted by the number of needed cycles for decapsulation.
  - Table 4.56, sorted by the number of needed cycles for the sum of key generation + encapsulation + decapsulation.

Table 4.44: KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted alphabetically after submission implementation names

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
BIG QUAKE 1	1	1047893031	3519702	4223050	1055635783
BIG QUAKE 3	3	8786966684	10857644	49593688	8847418016
BIG QUAKE 5	5	16528607297	12772072	51333539	16592712908
BIKE-1 1	1	730025	689193	2901203	4320421
BIKE-1 3	3	1709921	1850425	7666855	11227201
BIKE-1 5	5	2986647	3023816	17486906	23497369
BIKE-2 1	1	6383408	281755	2674115	9339278
BIKE-2 3	3	22205901	710970	7114241	30031112
BIKE-2 5	5	58806046	1201161	16485956	76493163
BIKE-3 1	1	433258	575237	3437956	4446451
BIKE-3 3	3	1100372	1460866	7732167	10293405
BIKE-3 5	5	2300332	3257675	18047493	23605500
CFPKM128	1	748800000*	1123200000*	1487200000*	2685280000
Classic McEliece mceliece8192128	5	6008245724	296036	458556	6009000316
CRYSTALS-KYBER 512	1	141872	205468	246040	593380
CRYSTALS-KYBER 768	3	243004	332616	394424	970044
CRYSTALS-KYBER 1024	5	368564	481042	558740	1408346
DAGS 1	1	49394032811	20109354	23639371	49437781536
DAGS 3	3	106876000000	26109354	24639371	106927000000
DAGS 5	5	136498000000	49029613	260829051	136808000000
DING Key Exchange 512	1	4399965	5735092	4774104	14909161
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
DME-144	1	-	0.122	0.593	-
DME-288	5	-	0.847	4.191	-
EMBLEM	1	776300	24700000	24700	25501000
FrodoKEM 640 AES	1	1287000	1810000	1811000	4908000
FrodoKEM 976 AES	3	2715000	3572000	3588000	9875000
FrodoKEM 640 cSHAKE	1	8297000	9082000	9077000	26456000
FrodoKEM 976 cSHAKE	3	17798000	19285000	19299000	56382000
HILA5	5	934320*	1222640*	229840*	2386800*
HQC Basic I	1	570000	1220000	1950000	3740000
HQC Basic II	1	610000	1280000	2070000	3960000
HQC Basic III	1	630000	1350000	2150000	4130000
HQC Advanced I	3	1260000	2610000	3820000	7690000
HQC Advanced II	3	1370000	2810000	3820000	8000000
HQC Advanced III	3	1470000	3020000	2350000	6840000
HQC Paranoiac I	5	2210000	4670000	6670000	13550000
HQC Paranoiac II	5	2520000	5370000	7510000	15400000
HQC Paranoiac III	5	2660000	5620000	8030000	16310000
HQC Paranoiac IV	5	2810000	5950000	8460000	17220000
KCL AKCN-MLWE	4	343023	411204	85215	839442
KCL AKCN-RLWE	5	338215	395116	83455	816786

KCL OKCN-MLWE	4	428257	703104	176481	1307842
KCL OKCN-RLWE	5	433536	715307	192306	1341149
KINDI-256-3-4-2	2	203096	260137	323947	787180
KINDI-256-5-2-2	5	519010	623436	723922	1866368
KINDI-512-2-2-2	4	214064	306043	397147	917254
KINDI-512-2-4-1	4	215542	307999	402041	925582
KINDI-512-3-2-1	5	723922	562640	698041	1984603
LAC-CCA-128	1	90411	160314	216957	467682
LAC-CCA-192	3	281324	421439	647030	1349793
LAC-CCA-256	5	267831	526915	874742	1669488
LAKE I	1	1580000	300000	1270000	3150000
LAKE II	3	1740000	310000	2090000	4140000
LAKE III	5	1790000	350000	2890000	5030000
LEDAkem-1-2	1	144232627	10458134	57583495	212274256
LEDAkem-1-3	1	60662499	8504380	60127907	129294786
LEDAkem-1-4	1	71709151	15873687	102280186	189863024
LEDAkem-3-2	3	303783612	38379599	154693170	496856381
LEDAkem-3-3	3	540278763	34559868	145110314	719948945
LEDAkem-3-4	3	225248772	41904122	145089675	412242569
LEDAkem-5-2	5	902318487	112652298	280835007	1295805792
LEDAkem-5-3	5	1858107259	106848011	298953313	2263908583
LEDAkem-5-4	5	623140012	115437155	386274394	1124851561
Lepton.CPA Light I	-	33625	78808	33400	145833
Lepton.CPA Light II	1	34912	85347	42462	162721
Lepton.CPA Moderate I	1	48932	117275	45519	211726
Lepton.CPA Moderate II	1	51519	125178	51353	228050
Lepton.CPA Moderate III	3	51508	130057	60289	241854
Lepton.CPA Moderate IV	5	57861	152431	72564	282856
Lepton.CPA Paranoid I	5	96602	237722	97757	432081
Lepton.CPA Paranoid II	5	97884	247932	105200	451016
Lepton-CCA Light I	-	34308	79152	87043	200503
Lepton-CCA Light II	1	34536	86584	100141	221261
Lepton-CCA Moderate I	1	49943	121564	132708	304215
Lepton-CCA Moderate II	1	51658	124426	141988	318072
Lepton-CCA Moderate III	3	52699	130631	151185	334515
Lepton-CCA Moderate IV	5	59450	154473	179520	393443
Lepton-CCA Paranoid I	5	94454	234441	264881	593776
Lepton-CCA Paranoid II	5	97569	244706	282199	624474
LIMA-CCA-sp-1018	1	1429742	1241867	1612433	4284042
LIMA-CCA-sp-1306	2	2600237	2361683	3085679	8047599
LIMA-CCA-sp-1822	3	2853857	2512619	3263201	8629677
LIMA-CCA-sp-2062	5	5114770	4738128	6237127	16090025
LIMA-CCA-2p-1024	3	654921	486938	611232	1753091
LIMA-CCA-2p-2048	5	1325909	1262893	1229593	3818395
LIMA-CPA-sp-1018	1	1429742	1233953	396764	3060459
LIMA-CPA-sp-1306	2	2600237	2355666	770710	5726613

LIMA-CPA-sp-1822	3	2853857	2505937	812501	6172295
LIMA-CPA-sp-2062	5	5114770	4729707	1553638	11398115
LIMA-CPA-2p-1024	3	654921	480913	156297	1292131
LIMA-CPA-2p-2048	5	1325909	1117377	481230	2924516
Lizard-CATEGORY1-N536	1	45807345	1382731	1359913	48549989
Lizard-CATEGORY1-N663	1	58381975	2124422	1828766	62335163
Lizard-CATEGORY3-N816	3	92361739	2757373	2160853	97279965
Lizard-CATEGORY3-N925	3	108118130	2951099	2959018	114028247
Lizard-CATEGORY5-N1088	5	6372454400*	17596800*	18824000*	6408875200
Lizard-CATEGORY5-N1300	5	3810518400*	17180800*	18636800*	3846336000
RLizard-CATEGORY1	1	939058	533152	122781	1594991
RLizard-CATEGORY3-N1024	3	916915	334678	217213	1468806
RLizard-CATEGORY3-N2048	3	1806966	343911	963863	3114740
RLizard-CATEGORY5	5	1336795	1060163	660404	3057362
LOCKER I	1	2710000	550000	2570000	5830000
LOCKER II	3	3190000	540000	1080000	4810000
LOCKER III	5	3580000	600000	3770000	7950000
LOCKER IV	1	3720000	710000	2860000	7290000
LOCKER V	3	4360000	860000	4320000	9540000
LOCKER VI	5	4360000	750000	4060000	9170000
LOCKER VII	1	8440000	1350000	4780000	14570000
LOCKER VIII	3	9480000	1390000	5000000	15870000
LOCKER IX	5	10400000	1490000	6600000	18490000
LOTUS-128	1	26825276	315611	3786920	30927807
LOTUS-192	3	46095015	462842	598836	47156693
LOTUS-256	5	71846095	584915	867464	73298474
Mersenne-756839	5	17090755	25367142	56778896	99236793
NewHope-CPA-512	1	106820	155840	40988	303648
NewHope-CPA-1024	5	117128	180648	206244	504020
NewHope-CCA-512	1	222922	330828	87080	640830
NewHope-CCA-1024	5	244944	377092	437056	1059092
NTRU-HRSS-KEM-701	1	18151998	1208946	3578538	22939482
NTRU Prime ntrulpr4591761	5	14060919	44116905	71245370	129423194
NTRU Prime sntrup4591761	5	6000000	59456	97684	6157140
NTRUEncrypt-443	1	1257307	394406	363281	2014994
NTRUEncrypt-734	4	3031086	579527	767267	4377880
NTRUEncrypt-1024	5	135483043	224147211	385916996	745547250
NTS-KEM (12, 6)	1	41746373	172463	689087	42607923
NTS-KEM (13, 80)	3	135813837	429301	1300102	137543240
NTS-KEM (13, 136)	5	249939545	544406	2911120	253395071
Ouroboros-R-128	1	600000	980000	1780000	3360000
Ouroboros-R-192	3	650000	1120000	3260000	5030000
Ouroboros-R-256	5	820000	1390000	4730000	6940000
pqrsa-KEM-30	2	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$	$6.11{\cdot}10^{14}$
QC-MDPC KEM	5	131540379	20180017	229002269	380722665
Ramstake 216091	1	9445009	17700978	36706919	63852906

Ramstake 756839	5	43148424	79342014	154721609	277212047
RLCE-KEM-128A	1	465481183	1040629	3589491	470111303
RLCE-KEM-128B	1	1011071617	1805010	4646941	1017523568
RLCE-KEM-192A	3	1962533052	2361787	7160709	1972055548
RLCE-KEM-192B	3	3829675407	3331234	8668186	3841674827
RLCE-KEM-256A	5	5057459034	5362174	24174369	5086995577
RLCE-KEM-256B	5	9612380645	8184051	36705481	9657270177
Round2-nround2-nd-l1	1	5490000	10680000	5220000	21390000
Round2-nround2-nd-l2	2	7990000	15640000	7660000	31290000
Round2-nround2-nd-l3	3	10350000	20300000	9990000	40640000
Round2-nround2-nd-l4	4	14350000	28250000	13960000	56560000
Round2-nround2-nd-l5	5	12370000	28260000	13	40630013
Round2-uround2-nd-l1	1	330000	360000	50000	740000
Round2-uround2-nd-l2	2	440000	500000	80000	1020000
Round2-uround2-nd-l3	3	460000	530000	70000	1060000
Round2-uround2-nd-l4	4	640000	760000	130000	1530000
Round2-uround2-nd-l5	5	630000	720000	100000	1450000
Round2-uround2-n1-fn0-l1	1	29109913	33185468	448417	62743798
Round2-uround2-n1-fn0-l2	2	35716914	37238338	344978	73300230
Round2-uround2-n1-fn0-l3	3	38444351	40440272	345716	79230339
Round2-uround2-n1-fn0-l4	4	56941220	56565761	551546	114058527
Round2-uround2-n1-fn0-l5	5	55115889	56034085	433575	111583549
Round2-uround2-n1-fn1-l1	1	1865144	3025972	220117	5111233
Round2-uround2-n1-fn1-l2	2	3969339	4861933	319314	9150586
Round2-uround2-n1-fn1-l3	3	3491466	5689054	308673	9489193
Round2-uround2-n1-fn1-l4	4	7703134	9707152	575705	17985991
Round2-uround2-n1-fn1-l5	5	6589401	8665781	402090	15657272
Round2-uround2-n1-fn2-l1	1	3430000	4300000	180000	7910000
Round2-uround2-n1-fn2-l2	2	7080000	7590000	250000	14920000
Round2-uround2-n1-fn2-l3	3	6650000	8140000	260000	15050000
Round2-uround2-n1-fn2-l4	4	10820000	11830000	430000	23080000
Round2-uround2-n1-fn2-l5	5	9360000	10110000	340000	19810000
RQC-128	1	790000	1970000	5300000	8060000
RQC-192	3	1760000	5600000	14460000	21820000
RQC-256	5	2820000	6460000	18000000	27280000
SABER light	1	105881	155131	179415	440427
SABER	3	216597	267841	318785	803223
SABER fire	5	360539	400817	472366	1233722
SIKEp503	1	1561680	2207324	2663521	6432525
SIKEp751	3	4735527	6485322	7996219	19217068
SIKEp964	5	10563749	14995526	17957283	43516558
Three Bears BabyBear	2	41000	60000	101000	202000
Three Bears BabyBear Ephem	2	41000	62000	34000	137000
Three Bears MamaBear	4	79000	97000	152000	328000
Three Bears MamaBear Ephem	4	84000	103000	34000	221000
Three Bears PapaBear	5	119000	145000	213000	477000

Three Bears PapaBear Ephem	5	125000	154000	40000	319000
Titanium CCA toy	-	60	1061298	176191	1237549
Titanium CCA lite	-	1507082	1317313	218207	3042602
Titanium CCA standard	1	1981835	1508258	261583	3751676
Titanium CCA medium	1	2221874	1009472	301930	3533276
Titanium CCA high	3	2179991	2041861	376220	4598072
Titanium CCA super	5	3054311	2917708	534948	6506967

Table 4.45: NIST security category 1 and 2 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for key generation

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Lepton-CCA Light II	1	34536	86584	100141	221261
Lepton.CPA Light II	1	34912	85347	42462	162721
Three Bears BabyBear	2	41000	60000	101000	202000
Three Bears BabyBear Ephem	2	41000	62000	34000	137000
Lepton.CPA Moderate I	1	48932	117275	45519	211726
Lepton-CCA Moderate I	1	49943	121564	132708	304215
Lepton.CPA Moderate II	1	51519	125178	51353	228050
LAC-CCA-128	1	90411	160314	216957	467682
SABER light	1	105881	155131	179415	440427
NewHope-CPA-512	1	106820	155840	40988	303648
CRYSTALS-KYBER 512	1	141872	205468	246040	593380
KINDI-256-3-4-2	2	203096	260137	323947	787180
NewHope-CCA-512	1	222922	330828	87080	640830
Round2-uround2-nd-l1	1	330000	360000	50000	740000
BIKE-3 1	1	433258	575237	3437956	4446451
Round2-uround2-nd-l2	2	440000	500000	80000	1020000
HQC Basic I	1	570000	1220000	1950000	3740000
Ouroboros-R-128	1	600000	980000	1780000	3360000
HQC Basic II	1	610000	1280000	2070000	3960000
HQC Basic III	1	630000	1350000	2150000	4130000
BIKE-1 1	1	730025	689193	2901203	4320421
EMBLEM	1	776300	24700000	24700	25501000
RQC-128	1	790000	1970000	5300000	8060000
RLizard-CATEGORY1	1	939058	533152	122781	1594991
NTRUEncrypt-443	1	1257307	394406	363281	2014994
FrodoKEM 640 AES	1	1287000	1810000	1811000	4908000
LIMA-CCA-sp-1018	1	1429742	1241867	1612433	4284042
LIMA-CPA-sp-1018	1	1429742	1233953	396764	3060459
SIKEp503	1	1561680	2207324	2663521	6432525
LAKE I	1	1580000	300000	1270000	3150000
Round2-uround2-n1-fn1-l1	1	1865144	3025972	220117	5111233
Titanium CCA standard	1	1981835	1508258	261583	3751676
Titanium CCA medium	1	2221874	1009472	301930	3533276
LIMA-CCA-sp-1306	2	2600237	2361683	3085679	8047599
LIMA-CPA-sp-1306	2	2600237	2355666	770710	5726613
LOCKER I	1	2710000	550000	2570000	5830000
Round2-uround2-n1-fn2-l1	1	3430000	4300000	180000	7910000
LOCKER IV	1	3720000	710000	2860000	7290000
Round2-uround2-n1-fn1-l2	2	3969339	4861933	319314	9150586
DING Key Exchange 512	1	4399965	5735092	4774104	14909161
Round2-nround2-nd-l1	1	5490000	10680000	5220000	21390000
BIKE-2 1	1	6383408	281755	2674115	9339278

Round2-uround2-n1-fn2-l2	2	7080000	7590000	250000	14920000
Round2-nround2-nd-l2	2	7990000	15640000	7660000	31290000
FrodoKEM 640 cSHAKE	1	8297000	9082000	9077000	26456000
LOCKER VII	1	8440000	1350000	4780000	14570000
Ramstake 216091	1	9445009	17700978	36706919	63852906
NTRU-HRSS-KEM-701	1	18151998	1208946	3578538	22939482
LOTUS-128	1	26825276	315611	3786920	30927807
Round2-uround2-n1-fn0-l1	1	29109913	33185468	448417	62743798
Round2-uround2-n1-fn0-l2	2	35716914	37238338	344978	73300230
NTS-KEM (12, 6)	1	41746373	172463	689087	42607923
Lizard-CATEGORY1-N536	1	45807345	1382731	1359913	48549989
Lizard-CATEGORY1-N663	1	58381975	2124422	1828766	62335163
LEDAkem-1-3	1	60662499	8504380	60127907	129294786
LEDAkem-1-4	1	71709151	15873687	102280186	189863024
LEDAkem-1-2	1	144232627	10458134	57583495	212274256
RLCE-KEM-128A	1	465481183	1040629	3589491	470111303
CFPKM128	1	748800000*	1123200000*	1487200000*	3359200000
RLCE-KEM-128B	1	1011071617	1805010	4646941	1017523568
BIG QUAKE 1	1	1047893031	3519702	4223050	1055635783
DAGS 1	1	49394032811	20109354	23639371	49437781536
pqrsa-KEM-30	2	$5.87 \cdot 10^{14}$	$1.82 \cdot 10^{13}$	$2.23 \cdot 10^{13}$	$6.11 \cdot 10^{14}$

Table 4.46: NIST security category 1 and 2 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for encapsulation.

Submisssion Implementation	Sec	Key.Gen	Encap	Decap	Sum
Three Bears BabyBear	2	41000	60000	101000	202000
Three Bears BabyBear Ephem	2	41000	62000	34000	137000
Lepton.CPA Light II	1	34912	85347	42462	162721
Lepton-CCA Light II	1	34536	86584	100141	221261
Lepton.CPA Moderate I	1	48932	117275	45519	211726
Lepton-CCA Moderate I	1	49943	121564	132708	304215
Lepton.CPA Moderate II	1	51519	125178	51353	228050
SABER light	1	105881	155131	179415	440427
NewHope-CPA-512	1	106820	155840	40988	303648
LAC-CCA-128	1	90411	160314	216957	467682
NTS-KEM (12, 6)	1	41746373	172463	689087	42607923
CRYSTALS-KYBER 512	1	141872	205468	246040	593380
KINDI-256-3-4-2	2	203096	260137	323947	787180
BIKE-2 1	1	6383408	281755	2674115	9339278
LAKE I	1	1580000	300000	1270000	3150000
LOTUS-128	1	26825276	315611	3786920	30927807
NewHope-CCA-512	1	222922	330828	87080	640830
Round2-uround2-nd-l1	1	330000	360000	50000	740000
NTRUEncrypt-443	1	1257307	394406	363281	2014994
Round2-uround2-nd-l2	2	440000	500000	80000	1020000
RLizard-CATEGORY1	1	939058	533152	122781	1594991
LOCKER I	1	2710000	550000	2570000	5830000
BIKE-3 1	1	433258	575237	3437956	4446451
BIKE-1 1	1	730025	689193	2901203	4320421
LOCKER IV	1	3720000	710000	2860000	7290000
Ouroboros-R-128	1	600000	980000	1780000	3360000
Titanium CCA medium	1	2221874	1009472	301930	3533276
RLCE-KEM-128A	1	465481183	1040629	3589491	470111303
NTRU-HRSS-KEM-701	1	18151998	1208946	3578538	22939482
HQC Basic I	1	570000	1220000	1950000	3740000
LIMA-CPA-sp-1018	1	1429742	1233953	396764	3060459
LIMA-CCA-sp-1018	1	1429742	1241867	1612433	4284042
HQC Basic II	1	610000	1280000	2070000	3960000
HQC Basic III	1	630000	1350000	2150000	4130000
LOCKER VII	1	8440000	1350000	4780000	14570000
Lizard-CATEGORY1-N536	1	45807345	1382731	1359913	48549989
Titanium CCA standard	1	1981835	1508258	261583	3751676
RLCE-KEM-128B	1	1011071617	1805010	4646941	1017523568
FrodoKEM 640 AES	1	1287000	1810000	1811000	4908000
RQC-128	1	790000	1970000	5300000	8060000
Lizard-CATEGORY1-N663	1	58381975	2124422	1828766	62335163
SIKEp503	1	1561680	2207324	2663521	6432525

LIMA-CPA-sp-1306	2	2600237	2355666	770710	5726613
LIMA-CCA-sp-1306	2	2600237	2361683	3085679	8047599
Round2-uround2-n1-fn1-l1	1	1865144	3025972	220117	5111233
BIG QUAKE 1	1	1047893031	3519702	4223050	1055635783
Round2-uround2-n1-fn2-l1	1	3430000	4300000	180000	7910000
Round2-uround2-n1-fn1-l2	2	3969339	4861933	319314	9150586
DING Key Exchange 512	1	4399965	5735092	4774104	14909161
Round2-uround2-n1-fn2-l2	2	7080000	7590000	250000	14920000
LEDAkem-1-3	1	60662499	8504380	60127907	129294786
FrodoKEM 640 cSHAKE	1	8297000	9082000	9077000	26456000
Round2-nround2-nd-l1	1	5490000	10680000	5220000	21390000
LEDAkem-1-2	1	144232627	10458134	57583495	212274256
Round2-nround2-nd-l2	2	7990000	15640000	7660000	31290000
LEDAkem-1-4	1	71709151	15873687	102280186	189863024
Ramstake 216091	1	9445009	17700978	36706919	63852906
DAGS 1	1	49394032811	20109354	23639371	49437781536
EMBLEM	1	776300	24700000	24700	25501000
Round2-uround2-n1-fn0-l1	1	29109913	33185468	448417	62743798
Round2-uround2-n1-fn0-l2	2	35716914	37238338	344978	73300230
CFPKM128	1	748800000*	1123200000*	1487200000*	3359200000
pqrsa-KEM-30	2	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$	$6.11{\cdot}10^{14}$

Table 4.47: NIST security category 1 and 2 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for decapsulation.

Submission Implementation	**Sec**	**Key.Gen**	**Encap**	**Decap**	**Sum**
EMBLEM	1	776300	24700000	24700	25501000
Three Bears BabyBear Ephem	2	41000	62000	34000	137000
NewHope-CPA-512	1	106820	155840	40988	303648
Lepton.CPA Light II	1	34912	85347	42462	162721
Lepton.CPA Moderate I	1	48932	117275	45519	211726
Round2-uround2-nd-l1	1	330000	360000	50000	740000
Lepton.CPA Moderate II	1	51519	125178	51353	228050
Round2-uround2-nd-l2	2	440000	500000	80000	1020000
NewHope-CCA-512	1	222922	330828	87080	640830
Three Bears BabyBear	2	41000	60000	101000	202000
Lepton-CCA Light II	1	34536	86584	100141	221261
RLizard-CATEGORY1	1	939058	533152	122781	1594991
Lepton-CCA Moderate I	1	49943	121564	132708	304215
SABER light	1	105881	155131	179415	440427
Round2-uround2-n1-fn2-l1	1	3430000	4300000	180000	7910000
LAC-CCA-128	1	90411	160314	216957	467682
Round2-uround2-n1-fn1-l1	1	1865144	3025972	220117	5111233
CRYSTALS-KYBER 512	1	141872	205468	246040	593380
Round2-uround2-n1-fn2-l2	2	7080000	7590000	250000	14920000
Titanium CCA standard	1	1981835	1508258	261583	3751676
Titanium CCA medium	1	2221874	1009472	301930	3533276
Round2-uround2-n1-fn1-l2	2	3969339	4861933	319314	9150586
KINDI-256-3-4-2	2	203096	260137	323947	787180
Round2-uround2-n1-fn0-l2	2	35716914	37238338	344978	73300230
NTRUEncrypt-443	1	1257307	394406	363281	2014994
LIMA-CPA-sp-1018	1	1429742	1233953	396764	3060459
Round2-uround2-n1-fn0-l1	1	29109913	33185468	448417	62743798
NTS-KEM (12, 6)	1	41746373	172463	689087	42607923
LIMA-CPA-sp-1306	2	2600237	2355666	770710	5726613
LAKE I	1	1580000	300000	1270000	3150000
Lizard-CATEGORY1-N536	1	45807345	1382731	1359913	48549989
LIMA-CCA-sp-1018	1	1429742	1241867	1612433	4284042
Ouroboros-R-128	1	600000	980000	1780000	3360000
FrodoKEM 640 AES	1	1287000	1810000	1811000	4908000
Lizard-CATEGORY1-N663	1	58381975	2124422	1828766	62335163
HQC Basic I	1	570000	1220000	1950000	3740000
HQC Basic II	1	610000	1280000	2070000	3960000
HQC Basic III	1	630000	1350000	2150000	4130000
LOCKER I	1	2710000	550000	2570000	5830000
SIKEp503	1	1561680	2207324	2663521	6432525
BIKE-2 1	1	6383408	281755	2674115	9339278
LOCKER IV	1	3720000	710000	2860000	7290000

BIKE-1 1	1	730025	689193	2901203	4320421
LIMA-CCA-sp-1306	2	2600237	2361683	3085679	8047599
BIKE-3 1	1	433258	575237	3437956	4446451
NTRU-HRSS-KEM-701	1	18151998	1208946	3578538	22939482
RLCE-KEM-128A	1	465481183	1040629	3589491	470111303
LOTUS-128	1	26825276	315611	3786920	30927807
BIG QUAKE 1	1	1047893031	3519702	4223050	1055635783
DING Key Exchange 512	1	4399965	5735092	4774104	14909161
RLCE-KEM-128B	1	1011071617	1805010	4646941	1017523568
LOCKER VII	1	8440000	1350000	4780000	14570000
Round2-nround2-nd-l1	1	5490000	10680000	5220000	21390000
RQC-128	1	790000	1970000	5300000	8060000
Round2-nround2-nd-l2	2	7990000	15640000	7660000	31290000
FrodoKEM 640 cSHAKE	1	8297000	9082000	9077000	26456000
DAGS 1	1	49394032811	20109354	23639371	49437781536
Ramstake 216091	1	9445009	17700978	36706919	63852906
LEDAkem-1-2	1	144232627	10458134	57583495	212274256
LEDAkem-1-3	1	60662499	8504380	60127907	129294786
LEDAkem-1-4	1	71709151	15873687	102280186	189863024
CFPKM128	1	748800000*	1123200000*	1487200000*	3359200000
pqrsa-KEM-30	2	$5.87 \cdot 10^{14}$	$1.82 \cdot 10^{13}$	$2.23 \cdot 10^{13}$	$6.11 \cdot 10^{14}$

Table 4.48: NIST security category 1 and 2 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for the sum of key generation + encapsulation + decapsulation.

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Three Bears BabyBear Ephem	2	41000	62000	34000	137000
Lepton.CPA Light II	1	34912	85347	42462	162721
Three Bears BabyBear	2	41000	60000	101000	202000
Lepton.CPA Moderate I	1	48932	117275	45519	211726
Lepton-CCA Light II	1	34536	86584	100141	221261
Lepton.CPA Moderate II	1	51519	125178	51353	228050
NewHope-CPA-512	1	106820	155840	40988	303648
Lepton-CCA Moderate I	1	49943	121564	132708	304215
SABER light	1	105881	155131	179415	440427
LAC-CCA-128	1	90411	160314	216957	467682
CRYSTALS-KYBER 512	1	141872	205468	246040	593380
NewHope-CCA-512	1	222922	330828	87080	640830
Round2-uround2-nd-l1	1	330000	360000	50000	740000
KINDI-256-3-4-2	2	203096	260137	323947	787180
Round2-uround2-nd-l2	2	440000	500000	80000	1020000
RLizard-CATEGORY1	1	939058	533152	122781	1594991
NTRUEncrypt-443	1	1257307	394406	363281	2014994
LIMA-CPA-sp-1018	1	1429742	1233953	396764	3060459
LAKE I	1	1580000	300000	1270000	3150000
Ouroboros-R-128	1	600000	980000	1780000	3360000
Titanium CCA medium	1	2221874	1009472	301930	3533276
HQC Basic I	1	570000	1220000	1950000	3740000
Titanium CCA standard	1	1981835	1508258	261583	3751676
HQC Basic II	1	610000	1280000	2070000	3960000
HQC Basic III	1	630000	1350000	2150000	4130000
LIMA-CCA-sp-1018	1	1429742	1241867	1612433	4284042
BIKE-1 1	1	730025	689193	2901203	4320421
BIKE-3 1	1	433258	575237	3437956	4446451
FrodoKEM 640 AES	1	1287000	1810000	1811000	4908000
Round2-uround2-n1-fn1-l1	1	1865144	3025972	220117	5111233
LIMA-CPA-sp-1306	2	2600237	2355666	770710	5726613
LOCKER I	1	2710000	550000	2570000	5830000
SIKEp503	1	1561680	2207324	2663521	6432525
LOCKER IV	1	3720000	710000	2860000	7290000
Round2-uround2-n1-fn2-l1	1	3430000	4300000	180000	7910000
LIMA-CCA-sp-1306	2	2600237	2361683	3085679	8047599
RQC-128	1	790000	1970000	5300000	8060000
Round2-uround2-n1-fn1-l2	2	3969339	4861933	319314	9150586
BIKE-2 1	1	6383408	281755	2674115	9339278
LOCKER VII	1	8440000	1350000	4780000	14570000
DING Key Exchange 512	1	4399965	5735092	4774104	14909161
Round2-uround2-n1-fn2-l2	2	7080000	7590000	250000	14920000

Round2-nround2-nd-l1	1	5490000	10680000	5220000	21390000
NTRU-HRSS-KEM-701	1	18151998	1208946	3578538	22939482
EMBLEM	1	776300	24700000	24700	25501000
EMBLEM	1	776300	24700000	24700	25501000
FrodoKEM 640 cSHAKE	1	8297000	9082000	9077000	26456000
LOTUS-128	1	26825276	315611	3786920	30927807
Round2-nround2-nd-l2	2	7990000	15640000	7660000	31290000
NTS-KEM (12, 6)	1	41746373	172463	689087	42607923
Lizard-CATEGORY1-N536	1	45807345	1382731	1359913	48549989
Lizard-CATEGORY1-N663	1	58381975	2124422	1828766	62335163
Round2-uround2-n1-fn0-l1	1	29109913	33185468	448417	62743798
Ramstake 216091	1	9445009	17700978	36706919	63852906
Round2-uround2-n1-fn0-l2	2	35716914	37238338	344978	73300230
LEDAkem-1-3	1	60662499	8504380	60127907	129294786
LEDAkem-1-4	1	71709151	15873687	102280186	189863024
LEDAkem-1-2	1	144232627	10458134	57583495	212274256
RLCE-KEM-128A	1	465481183	1040629	3589491	470111303
RLCE-KEM-128B	1	1011071617	1805010	4646941	1017523568
BIG QUAKE 1	1	1047893031	3519702	4223050	1055635783
CFPKM128	1	748800000*	1123200000*	1487200000*	3359200000
DAGS 1	1	49394032811	20109354	23639371	49437781536
pqrsa-KEM-30	2	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$	$6.11{\cdot}10^{14}$

Table 4.49: NIST security category 3 and 4 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for key generation

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Lepton.CPA Moderate III	3	51508	130057	60289	241854
Lepton-CCA Moderate III	3	52699	130631	151185	334515
Three Bears MamaBear	4	79000	97000	152000	328000
Three Bears MamaBear Ephem	4	84000	103000	34000	221000
KINDI-512-2-2-2	4	214064	306043	397147	917254
KINDI-512-2-4-1	4	215542	307999	402041	925582
SABER	3	216597	267841	318785	803223
CRYSTALS-KYBER 768	3	243004	332616	394424	970044
LAC-CCA-192	3	281324	421439	647030	1349793
KCL AKCN-MLWE	4	343023	411204	85215	839442
KCL OKCN-MLWE	4	428257	703104	176481	1307842
Round2-uround2-nd-l3	3	460000	530000	70000	1060000
Round2-uround2-nd-l4	4	640000	760000	130000	1530000
Ouroboros-R-192	3	650000	1120000	3260000	5030000
LIMA-CCA-2p-1024	3	654921	486938	611232	1753091
LIMA-CPA-2p-1024	3	654921	480913	156297	1292131
RLizard-CATEGORY3-N1024	3	916915	334678	217213	1468806
BIKE-3 3	3	1100372	1460866	7732167	10293405
HQC Advanced I	3	1260000	2610000	3820000	7690000
HQC Advanced II	3	1370000	2810000	3820000	8000000
HQC Advanced III	3	1470000	3020000	2350000	6840000
BIKE-1 3	3	1709921	1850425	7666855	11227201
LAKE II	3	1740000	310000	2090000	4140000
RQC-192	3	1760000	5600000	14460000	21820000
RLizard-CATEGORY3-N2048	3	1806966	343911	963863	3114740
Titanium CCA high	3	2179991	2041861	376220	4598072
FrodoKEM 976 AES	3	2715000	3572000	3588000	9875000
LIMA-CCA-sp-1822	3	2853857	2512619	3263201	8629677
LIMA-CPA-sp-1822	3	2853857	2505937	812501	6172295
NTRUEncrypt-734	4	3031086	579527	767267	4377880
LOCKER II	3	3190000	540000	1080000	4810000
Round2-uround2-n1-fn1-l3	3	3491466	5689054	308673	9489193
LOCKER V	3	4360000	860000	4320000	9540000
SIKEp751	3	4735527	6485322	7996219	19217068
Round2-uround2-n1-fn2-l3	3	6650000	8140000	260000	15050000
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
Round2-uround2-n1-fn1-l4	4	7703134	9707152	575705	17985991
LOCKER VIII	3	9480000	1390000	5000000	15870000
Round2-uround2-n1-fn2-l4	4	10820000	11830000	430000	23080000
Round2-nround2-nd-l3	3	10350000	20300000	9990000	40640000
Round2-nround2-nd-l4	4	14350000	28250000	13960000	56560000
FrodoKEM 976 cSHAKE	3	17798000	19285000	19299000	56382000

BIKE-2 3	3	22205901	710970	7114241	30031112
Round2-uround2-n1-fn0-l3	3	38444351	40440272	345716	79230339
LOTUS-192	3	46095015	462842	598836	47156693
Round2-uround2-n1-fn0-l4	4	56941220	56565761	551546	114058527
Lizard-CATEGORY3-N816	3	92361739	2757373	2160853	97279965
Lizard-CATEGORY3-N925	3	108118130	2951099	2959018	114028247
NTS-KEM (13, 80)	3	135813837	429301	1300102	137543240
LEDAkem-3-4	3	225248772	41904122	145089675	412242569
LEDAkem-3-2	3	303783612	38379599	154693170	496856381
LEDAkem-3-3	3	540278763	34559868	145110314	719948945
RLCE-KEM-192A	3	1962533052	2361787	7160709	1972055548
RLCE-KEM-192B	3	3829675407	3331234	8668186	3841674827
BIG QUAKE 3	3	8786966684	10857644	49593688	8847418016
DAGS 3	3	106876000000	26109354	24639371	106927000000

Table 4.50: NIST security category 3 and 4 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for encapsulation.

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Three Bears MamaBear	4	79000	97000	152000	328000
Three Bears MamaBear Ephem	4	84000	103000	34000	221000
Lepton.CPA Moderate III	3	51508	130057	60289	241854
Lepton-CCA Moderate III	3	52699	130631	151185	334515
SABER	3	216597	267841	318785	803223
LAKE II	3	1740000	310000	2090000	4140000
KINDI-512-2-2-2	4	214064	306043	397147	917254
KINDI-512-2-4-1	4	215542	307999	402041	925582
CRYSTALS-KYBER 768	3	243004	332616	394424	970044
RLizard-CATEGORY3-N1024	3	916915	334678	217213	1468806
RLizard-CATEGORY3-N2048	3	1806966	343911	963863	3114740
KCL AKCN-MLWE	4	343023	411204	85215	839442
LAC-CCA-192	3	281324	421439	647030	1349793
NTS-KEM (13, 80)	3	135813837	429301	1300102	137543240
LOTUS-192	3	46095015	462842	598836	47156693
LIMA-CPA-2p-1024	3	654921	480913	156297	1292131
LIMA-CCA-2p-1024	3	654921	486938	611232	1753091
Round2-uround2-nd-l3	3	460000	530000	70000	1060000
NTRUEncrypt-734	4	3031086	579527	767267	4377880
KCL OKCN-MLWE	4	428257	703104	176481	1307842
BIKE-2 3	3	22205901	710970	7114241	30031112
Round2-uround2-nd-l4	4	640000	760000	130000	1530000
LOCKER V	3	4360000	860000	4320000	9540000
Ouroboros-R-192	3	650000	1120000	3260000	5030000
LOCKER VIII	3	9480000	1390000	5000000	15870000
BIKE-3 3	3	1100372	1460866	7732167	10293405
BIKE-1 3	3	1709921	1850425	7666855	11227201
Titanium CCA high	3	2179991	2041861	376220	4598072
RLCE-KEM-192A	3	1962533052	2361787	7160709	1972055548
LIMA-CPA-sp-1822	3	2853857	2505937	812501	6172295
LIMA-CCA-sp-1822	3	2853857	2512619	3263201	8629677
HQC Advanced I	3	1260000	2610000	3820000	7690000
Lizard-CATEGORY3-N816	3	92361739	2757373	2160853	97279965
HQC Advanced II	3	1370000	2810000	3820000	8000000
Lizard-CATEGORY3-N925	3	108118130	2951099	2959018	114028247
HQC Advanced III	3	1470000	3020000	2350000	6840000
RLCE-KEM-192B	3	3829675407	3331234	8668186	3841674827
FrodoKEM 976 AES	3	2715000	3572000	3588000	9875000
RQC-192	3	1760000	5600000	14460000	21820000
LOCKER II	3	3190000	540000	1080000	4810000
Round2-uround2-n1-fn1-l3	3	3491466	5689054	308673	9489193
SIKEp751	3	4735527	6485322	7996219	19217068

Round2-uround2-n1-fn2-l3	3	6650000	8140000	260000	15050000
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
Round2-uround2-n1-fn1-l4	4	7703134	9707152	575705	17985991
BIG QUAKE 3	3	8786966684	10857644	49593688	8847418016
Round2-uround2-n1-fn2-l4	4	10820000	11830000	430000	23080000
FrodoKEM 976 cSHAKE	3	17798000	19285000	19299000	56382000
Round2-nround2-nd-l3	3	10350000	20300000	9990000	40640000
DAGS 3	3	106876000000	26109354	24639371	106927000000
Round2-nround2-nd-l4	4	14350000	28250000	13960000	56560000
LEDAkem-3-3	3	540278763	34559868	145110314	719948945
LEDAkem-3-2	3	303783612	38379599	154693170	496856381
Round2-uround2-n1-fn0-l3	3	38444351	40440272	345716	79230339
LEDAkem-3-4	3	225248772	41904122	145089675	412242569
Round2-uround2-n1-fn0-l4	4	56941220	56565761	551546	114058527

Table 4.51: NIST security category 3 and 4 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for decapsulation.

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Three Bears MamaBear Ephem	4	84000	103000	34000	221000
Lepton.CPA Moderate III	3	51508	130057	60289	241854
Round2-uround2-nd-l3	3	460000	530000	70000	1060000
KCL AKCN-MLWE	4	343023	411204	85215	839442
Round2-uround2-nd-l4	4	640000	760000	130000	1530000
Lepton-CCA Moderate III	3	52699	130631	151185	334515
Three Bears MamaBear	4	79000	97000	152000	328000
LIMA-CPA-2p-1024	3	654921	480913	156297	1292131
KCL OKCN-MLWE	4	428257	703104	176481	1307842
RLizard-CATEGORY3-N1024	3	916915	334678	217213	1468806
Round2-uround2-n1-fn2-l3	3	6650000	8140000	260000	15050000
Round2-uround2-n1-fn1-l3	3	3491466	5689054	308673	9489193
SABER	3	216597	267841	318785	803223
Round2-uround2-n1-fn0-l3	3	38444351	40440272	345716	79230339
Titanium CCA high	3	2179991	2041861	376220	4598072
CRYSTALS-KYBER 768	3	243004	332616	394424	970044
KINDI-512-2-2-2	4	214064	306043	397147	917254
KINDI-512-2-4-1	4	215542	307999	402041	925582
Round2-uround2-n1-fn2-l4	4	10820000	11830000	430000	23080000
Round2-uround2-n1-fn0-l4	4	56941220	56565761	551546	114058527
Round2-uround2-n1-fn1-l4	4	7703134	9707152	575705	17985991
LOTUS-192	3	46095015	462842	598836	47156693
LIMA-CCA-2p-1024	3	654921	486938	611232	1753091
LAC-CCA-192	3	281324	421439	647030	1349793
NTRUEncrypt-734	4	3031086	579527	767267	4377880
LIMA-CPA-sp-1822	3	2853857	2505937	812501	6172295
RLizard-CATEGORY3-N2048	3	1806966	343911	963863	3114740
LOCKER II	3	3190000	540000	1080000	4810000
NTS-KEM (13, 80)	3	135813837	429301	1300102	137543240
LAKE II	3	1740000	310000	2090000	4140000
Lizard-CATEGORY3-N816	3	92361739	2757373	2160853	97279965
HQC Advanced III	3	1470000	3020000	2350000	6840000
Lizard-CATEGORY3-N925	3	108118130	2951099	2959018	114028247
Ouroboros-R-192	3	650000	1120000	3260000	5030000
LIMA-CCA-sp-1822	3	2853857	2512619	3263201	8629677
FrodoKEM 976 AES	3	2715000	3572000	3588000	9875000
HQC Advanced I	3	1260000	2610000	3820000	7690000
HQC Advanced II	3	1370000	2810000	3820000	8000000
LOCKER V	3	4360000	860000	4320000	9540000
LOCKER VIII	3	9480000	1390000	5000000	15870000
BIKE-2 3	3	22205901	710970	7114241	30031112
RLCE-KEM-192A	3	1962533052	2361787	7160709	1972055548

DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
BIKE-1 3	3	1709921	1850425	7666855	11227201
BIKE-3 3	3	1100372	1460866	7732167	10293405
SIKEp751	3	4735527	6485322	7996219	19217068
RLCE-KEM-192B	3	3829675407	3331234	8668186	3841674827
Round2-nround2-nd-l3	3	10350000	20300000	9990000	40640000
RQC-192	3	1760000	5600000	14460000	21820000
Round2-nround2-nd-l4	4	14350000	28250000	13960000	56560000
FrodoKEM 976 cSHAKE	3	17798000	19285000	19299000	56382000
DAGS 3	3	106876000000	26109354	24639371	106927000000
BIG QUAKE 3	3	8786966684	10857644	49593688	8847418016
LEDAkem-3-4	3	225248772	41904122	145089675	412242569
LEDAkem-3-3	3	540278763	34559868	145110314	719948945
LEDAkem-3-2	3	303783612	38379599	154693170	496856381

Table 4.52: NIST security category 3 and 4 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for the sum of key generation + encapsulation + decapsulation.

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Three Bears MamaBear Ephem	4	84000	103000	34000	221000
Lepton.CPA Moderate III	3	51508	130057	60289	241854
Three Bears MamaBear	4	79000	97000	152000	328000
Lepton-CCA Moderate III	3	52699	130631	151185	334515
SABER	3	216597	267841	318785	803223
KCL AKCN-MLWE	4	343023	411204	85215	839442
KINDI-512-2-2-2	4	214064	306043	397147	917254
KINDI-512-2-4-1	4	215542	307999	402041	925582
CRYSTALS-KYBER 768	3	243004	332616	394424	970044
Round2-uround2-nd-l3	3	460000	530000	70000	1060000
LIMA-CPA-2p-1024	3	654921	480913	156297	1292131
KCL OKCN-MLWE	4	428257	703104	176481	1307842
LAC-CCA-192	3	281324	421439	647030	1349793
RLizard-CATEGORY3-N1024	3	916915	334678	217213	1468806
Round2-uround2-nd-l4	4	640000	760000	130000	1530000
LIMA-CCA-2p-1024	3	654921	486938	611232	1753091
RLizard-CATEGORY3-N2048	3	1806966	343911	963863	3114740
LAKE II	3	1740000	310000	2090000	4140000
NTRUEncrypt-734	4	3031086	579527	767267	4377880
Titanium CCA high	3	2179991	2041861	376220	4598072
LOCKER II	3	3190000	540000	1080000	4810000
Ouroboros-R-192	3	650000	1120000	3260000	5030000
LIMA-CPA-sp-1822	3	2853857	2505937	812501	6172295
HQC Advanced III	3	1470000	3020000	2350000	6840000
HQC Advanced I	3	1260000	2610000	3820000	7690000
HQC Advanced II	3	1370000	2810000	3820000	8000000
LIMA-CCA-sp-1822	3	2853857	2512619	3263201	8629677
Round2-uround2-n1-fn1-l3	3	3491466	5689054	308673	9489193
LOCKER V	3	4360000	860000	4320000	9540000
FrodoKEM 976 AES	3	2715000	3572000	3588000	9875000
BIKE-3 3	3	1100372	1460866	7732167	10293405
BIKE-1 3	3	1709921	1850425	7666855	11227201
Round2-uround2-n1-fn2-l3	3	6650000	8140000	260000	15050000
LOCKER VIII	3	9480000	1390000	5000000	15870000
Round2-uround2-n1-fn1-l4	4	7703134	9707152	575705	17985991
SIKEp751	3	4735527	6485322	7996219	19217068
RQC-192	3	1760000	5600000	14460000	21820000
Round2-uround2-n1-fn2-l4	4	10820000	11830000	430000	23080000
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
BIKE-2 3	3	22205901	710970	7114241	30031112
Round2-nround2-nd-l3	3	10350000	20300000	9990000	40640000
LOTUS-192	3	46095015	462842	598836	47156693

FrodoKEM 976 cSHAKE	3	17798000	19285000	19299000	56382000
Round2-nround2-nd-l4	4	14350000	28250000	13960000	56560000
Round2-uround2-n1-fn0-l3	3	38444351	40440272	345716	79230339
Lizard-CATEGORY3-N816	3	92361739	2757373	2160853	97279965
Lizard-CATEGORY3-N925	3	108118130	2951099	2959018	114028247
Round2-uround2-n1-fn0-l4	4	56941220	56565761	551546	114058527
NTS-KEM (13, 80)	3	135813837	429301	1300102	137543240
LEDAkem-3-4	3	225248772	41904122	145089675	412242569
LEDAkem-3-2	3	303783612	38379599	154693170	496856381
LEDAkem-3-3	3	540278763	34559868	145110314	719948945
RLCE-KEM-192A	3	1962533052	2361787	7160709	1972055548
RLCE-KEM-192B	3	3829675407	3331234	8668186	3841674827
BIG QUAKE 3	3	8786966684	10857644	49593688	8847418016
DAGS 3	3	106876000000	26109354	24639371	106927000000

Table 4.53: NIST security category 5 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for key generation

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Lepton.CPA Moderate IV	5	57861	152431	72564	282856
Lepton-CCA Moderate IV	5	59450	154473	179520	393443
Lepton-CCA Paranoid I	5	94454	234441	264881	593776
Lepton.CPA Paranoid I	5	96602	237722	97757	432081
Lepton-CCA Paranoid II	5	97569	244706	282199	624474
Lepton.CPA Paranoid II	5	97884	247932	105200	451016
NewHope-CPA-1024	5	117128	180648	206244	504020
Three Bears PapaBear	5	119000	145000	213000	477000
Three Bears PapaBear Ephem	5	125000	154000	40000	319000
NewHope-CCA-1024	5	244944	377092	437056	1059092
LAC-CCA-256	5	267831	526915	874742	1669488
KCL AKCN-RLWE	5	338215	395116	83455	816786
SABER fire	5	360539	400817	472366	1233722
CRYSTALS-KYBER 1024	5	368564	481042	558740	1408346
KCL OKCN-RLWE	5	433536	715307	192306	1341149
KINDI-256-5-2-2	5	519010	623436	723922	1866368
Round2-uround2-nd-l5	5	630000	720000	100000	1450000
KINDI-512-3-2-1	5	723922	562640	698041	1984603
Ouroboros-R-256	5	820000	1390000	4730000	6940000
HILA5	5	934320*	1222640*	229840*	2386800*
LIMA-CCA-2p-2048	5	1325909	1262893	1229593	3818395
LIMA-CPA-2p-2048	5	1325909	1117377	481230	2924516
RLizard-CATEGORY5	5	1336795	1060163	660404	3057362
LAKE III	5	1790000	350000	2890000	5030000
HQC Paranoiac I	5	2210000	4670000	6670000	13550000
BIKE-3 5	5	2300332	3257675	18047493	23605500
HQC Paranoiac II	5	2520000	5370000	7510000	15400000
HQC Paranoiac III	5	2660000	5620000	8030000	16310000
HQC Paranoiac IV	5	2810000	5950000	8460000	17220000
RQC-256	5	2820000	6460000	18000000	27280000
BIKE-1 5	5	2986647	3023816	17486906	23497369
Titanium CCA super	5	3054311	2917708	534948	6506967
LOCKER III	5	3580000	600000	3770000	7950000
LOCKER VI	5	4360000	750000	4060000	9170000
LIMA-CCA-sp-2062	5	5114770	4738128	6237127	16090025
LIMA-CPA-sp-2062	5	5114770	4729707	1553638	11398115
NTRU Prime sntrup4591761	5	6000000	59456	97684	6157140
Round2-uround2-n1-fn1-l5	5	6589401	8665781	402090	15657272
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
Round2-uround2-n1-fn2-l5	5	9360000	10110000	340000	19810000
LOCKER IX	5	10400000	1490000	6600000	18490000
SIKEp964	5	10563749	14995526	17957283	43516558

Round2-nround2-nd-l5	5	12370000	28260000	13	40630013
NTRU Prime ntrulpr4591761	5	14060919	44116905	71245370	129423194
Mersenne-756839	5	17090755	25367142	56778896	99236793
Ramstake 756839	5	43148424	79342014	154721609	277212047
Round2-uround2-n1-fn0-l5	5	55115889	56034085	433575	111583549
BIKE-2 5	5	58806046	1201161	16485956	76493163
LOTUS-256	5	71846095	584915	867464	73298474
QC-MDPC KEM	5	131540379	20180017	229002269	380722665
NTRUEncrypt-1024	5	135483043	224147211	385916996	745547250
NTS-KEM (13, 136)	5	249939545	544406	2911120	253395071
LEDAkem-5-4	5	623140012	115437155	386274394	1124851561
LEDAkem-5-2	5	902318487	112652298	280835007	1295805792
LEDAkem-5-3	5	1858107259	106848011	298953313	2263908583
Lizard-CATEGORY5-N1300	5	3810518400*	17180800*	18636800*	3846336000
RLCE-KEM-256A	5	5057459034	5362174	24174369	5086995577
Classic McEliece mceliece8192128	5	6008245724	296036	458556	6009000316
Lizard-CATEGORY5-N1088	5	6372454400*	17596800*	18824000*	6408875200
RLCE-KEM-256B	5	9612380645	8184051	36705481	9657270177
BIG QUAKE 5	5	16528607297	12772072	51333539	16592712908
DAGS 5	5	136498000000	49029613	260829051	136808000000

Table 4.54: NIST security category 5 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for encapsulation.

Submisssion Implementation	**Sec**	**Key.Gen**	**Encap**	**Decap**	**Sum**
NTRU Prime sntrup4591761	5	6000000	59456	97684	6157140
Three Bears PapaBear	5	119000	145000	213000	477000
Lepton.CPA Moderate IV	5	57861	152431	72564	282856
Three Bears PapaBear Ephem	5	125000	154000	40000	319000
Lepton-CCA Moderate IV	5	59450	154473	179520	393443
NewHope-CPA-1024	5	117128	180648	206244	504020
Lepton-CCA Paranoid I	5	94454	234441	264881	593776
Lepton.CPA Paranoid I	5	96602	237722	97757	432081
Lepton-CCA Paranoid II	5	97569	244706	282199	624474
Lepton.CPA Paranoid II	5	97884	247932	105200	451016
Classic McEliece mceliece8192128	5	6008245724	296036	458556	6009000316
LAKE III	5	1790000	350000	2890000	5030000
NewHope-CCA-1024	5	244944	377092	437056	1059092
KCL AKCN-RLWE	5	338215	395116	83455	816786
SABER fire	5	360539	400817	472366	1233722
CRYSTALS-KYBER 1024	5	368564	481042	558740	1408346
LAC-CCA-256	5	267831	526915	874742	1669488
NTS-KEM (13, 136)	5	249939545	544406	2911120	253395071
KINDI-512-3-2-1	5	723922	562640	698041	1984603
LOTUS-256	5	71846095	584915	867464	73298474
LOCKER III	5	3580000	600000	3770000	7950000
KINDI-256-5-2-2	5	519010	623436	723922	1866368
KCL OKCN-RLWE	5	433536	715307	192306	1341149
Round2-uround2-nd-l5	5	630000	720000	100000	1450000
LOCKER VI	5	4360000	750000	4060000	9170000
RLizard-CATEGORY5	5	1336795	1060163	660404	3057362
LIMA-CPA-2p-2048	5	1325909	1117377	481230	2924516
BIKE-2 5	5	58806046	1201161	16485956	76493163
HILA5	5	934320*	1222640*	229840*	2386800*
LIMA-CCA-2p-2048	5	1325909	1262893	1229593	3818395
Ouroboros-R-256	5	820000	1390000	4730000	6940000
LOCKER IX	5	10400000	1490000	6600000	18490000
Titanium CCA super	5	3054311	2917708	534948	6506967
BIKE-1 5	5	2986647	3023816	17486906	23497369
BIKE-3 5	5	2300332	3257675	18047493	23605500
HQC Paranoiac I	5	2210000	4670000	6670000	13550000
LIMA-CPA-sp-2062	5	5114770	4729707	1553638	11398115
LIMA-CCA-sp-2062	5	5114770	4738128	6237127	16090025
RLCE-KEM-256A	5	5057459034	5362174	24174369	5086995577
HQC Paranoiac II	5	2520000	5370000	7510000	15400000
HQC Paranoiac III	5	2660000	5620000	8030000	16310000
HQC Paranoiac IV	5	2810000	5950000	8460000	17220000

RQC-256	5	2820000	6460000	18000000	27280000
RLCE-KEM-256B	5	9612380645	8184051	36705481	9657270177
Round2-uround2-n1-fn1-l5	5	6589401	8665781	402090	15657272
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
Round2-uround2-n1-fn2-l5	5	9360000	10110000	340000	19810000
BIG QUAKE 5	5	16528607297	12772072	51333539	16592712908
SIKEp964	5	10563749	14995526	17957283	43516558
Lizard-CATEGORY5-N1300	5	3810518400*	17180800*	18636800*	3846336000
Lizard-CATEGORY5-N1088	5	6372454400*	17596800*	18824000*	6408875200
QC-MDPC KEM	5	131540379	20180017	229002269	380722665
Mersenne-756839	5	17090755	25367142	56778896	99236793
Round2-nround2-nd-l5	5	12370000	28260000	13	40630013
NTRU Prime ntrulpr4591761	5	14060919	44116905	71245370	129423194
DAGS 5	5	136498000000	49029613	260829051	136808000000
Round2-uround2-n1-fn0-l5	5	55115889	56034085	433575	111583549
Ramstake 756839	5	43148424	79342014	154721609	277212047
LEDAkem-5-3	5	1858107259	106848011	298953313	2263908583
LEDAkem-5-2	5	902318487	112652298	280835007	1295805792
LEDAkem-5-4	5	623140012	115437155	386274394	1124851561
NTRUEncrypt-1024	5	135483043	224147211	385916996	745547250

Table 4.55: NIST security category 5 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for decapsulation.

Submission Implementation	Sec	Key.Gen	Encap	Decap	Sum
Round2-nround2-nd-l5	5	12370000	28260000	13	40630013
Three Bears PapaBear Ephem	5	125000	154000	40000	319000
Lepton.CPA Moderate IV	5	57861	152431	72564	282856
KCL AKCN-RLWE	5	338215	395116	83455	816786
NTRU Prime sntrup4591761	5	6000000	59456	97684	6157140
Lepton.CPA Paranoid I	5	96602	237722	97757	432081
Round2-uround2-nd-l5	5	630000	720000	100000	1450000
Lepton.CPA Paranoid II	5	97884	247932	105200	451016
Lepton-CCA Moderate IV	5	59450	154473	179520	393443
KCL OKCN-RLWE	5	433536	715307	192306	1341149
NewHope-CPA-1024	5	117128	180648	206244	504020
Three Bears PapaBear	5	119000	145000	213000	477000
HILA5	5	934320*	1222640*	229840*	2386800*
Lepton-CCA Paranoid I	5	94454	234441	264881	593776
Lepton-CCA Paranoid II	5	97569	244706	282199	624474
Round2-uround2-n1-fn2-l5	5	9360000	10110000	340000	19810000
Round2-uround2-n1-fn1-l5	5	6589401	8665781	402090	15657272
Round2-uround2-n1-fn0-l5	5	55115889	56034085	433575	111583549
NewHope-CCA-1024	5	244944	377092	437056	1059092
Classic McEliece mceliece8192128	5	6008245724	296036	458556	6009000316
SABER fire	5	360539	400817	472366	1233722
LIMA-CPA-2p-2048	5	1325909	1117377	481230	2924516
RLizard-CATEGORY5	5	1336795	1060163	660404	3057362
KINDI-512-3-2-1	5	723922	562640	698041	1984603
Titanium CCA super	5	3054311	2917708	534948	6506967
CRYSTALS-KYBER 1024	5	368564	481042	558740	1408346
KINDI-256-5-2-2	5	519010	623436	723922	1866368
LOTUS-256	5	71846095	584915	867464	73298474
LAC-CCA-256	5	267831	526915	874742	1669488
LIMA-CCA-2p-2048	5	1325909	1262893	1229593	3818395
LIMA-CPA-sp-2062	5	5114770	4729707	1553638	11398115
LAKE III	5	1790000	350000	2890000	5030000
NTS-KEM (13, 136)	5	249939545	544406	2911120	253395071
LOCKER III	5	3580000	600000	3770000	7950000
LOCKER VI	5	4360000	750000	4060000	9170000
Ouroboros-R-256	5	820000	1390000	4730000	6940000
LIMA-CCA-sp-2062	5	5114770	4738128	6237127	16090025
LOCKER IX	5	10400000	1490000	6600000	18490000
HQC Paranoiac I	5	2210000	4670000	6670000	13550000
HQC Paranoiac II	5	2520000	5370000	7510000	15400000
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
HQC Paranoiac III	5	2660000	5620000	8030000	16310000

HQC Paranoiac IV	5	2810000	5950000	8460000	17220000
BIKE-2 5	5	58806046	1201161	16485956	76493163
BIKE-1 5	5	2986647	3023816	17486906	23497369
SIKEp964	5	10563749	14995526	17957283	43516558
RQC-256	5	2820000	6460000	18000000	27280000
BIKE-3 5	5	2300332	3257675	18047493	23605500
Lizard-CATEGORY5-N1300	5	3810518400*	17180800*	18636800*	3846336000
Lizard-CATEGORY5-N1088	5	6372454400*	17596800*	18824000*	6408875200
QC-MDPC KEM	5	131540379	20180017	229002269	380722665
RLCE-KEM-256A	5	5057459034	5362174	24174369	5086995577
RLCE-KEM-256B	5	9612380645	8184051	36705481	9657270177
BIG QUAKE 5	5	16528607297	12772072	51333539	16592712908
Mersenne-756839	5	17090755	25367142	56778896	99236793
NTRU Prime ntrulpr4591761	5	14060919	44116905	71245370	129423194
DAGS 5	5	136498000000	49029613	260829051	136808000000
LEDAkem-5-3	5	1858107259	106848011	298953313	2263908583
Ramstake 756839	5	43148424	79342014	154721609	277212047
LEDAkem-5-2	5	902318487	112652298	280835007	1295805792
NTRUEncrypt-1024	5	135483043	224147211	385916996	745547250
LEDAkem-5-4	5	623140012	115437155	386274394	1124851561

Table 4.56: NIST security category 5 KEM implementation security levels, key generation, encapsulation, and decapsulation given in number of needed CPU cycles, sorted after running times for the sum of key generation + encapsulation + decapsulation.

Submission Implementation	**Sec**	**Key.Gen**	**Encap**	**Decap**	**Sum**
Lepton.CPA Moderate IV	5	57861	152431	72564	282856
Three Bears PapaBear Ephem	5	125000	154000	40000	319000
Lepton-CCA Moderate IV	5	59450	154473	179520	393443
Lepton.CPA Paranoid I	5	96602	237722	97757	432081
Lepton.CPA Paranoid II	5	97884	247932	105200	451016
Three Bears PapaBear	5	119000	145000	213000	477000
NewHope-CPA-1024	5	117128	180648	206244	504020
Lepton-CCA Paranoid I	5	94454	234441	264881	593776
Lepton-CCA Paranoid II	5	97569	244706	282199	624474
KCL AKCN-RLWE	5	338215	395116	83455	816786
NewHope-CCA-1024	5	244944	377092	437056	1059092
SABER fire	5	360539	400817	472366	1233722
KCL OKCN-RLWE	5	433536	715307	192306	1341149
CRYSTALS-KYBER 1024	5	368564	481042	558740	1408346
Round2-uround2-nd-l5	5	630000	720000	100000	1450000
LAC-CCA-256	5	267831	526915	874742	1669488
KINDI-256-5-2-2	5	519010	623436	723922	1866368
KINDI-512-3-2-1	5	723922	562640	698041	1984603
HILA5	5	934320*	1222640*	229840*	2386800*
LIMA-CPA-2p-2048	5	1325909	1117377	481230	2924516
RLizard-CATEGORY5	5	1336795	1060163	660404	3057362
LIMA-CCA-2p-2048	5	1325909	1262893	1229593	3818395
LAKE III	5	1790000	350000	2890000	5030000
NTRU Prime sntrup4591761	5	6000000	59456	97684	6157140
Titanium CCA super	5	3054311	2917708	534948	6506967
Ouroboros-R-256	5	820000	1390000	4730000	6940000
LOCKER III	5	3580000	600000	3770000	7950000
LOCKER VI	5	4360000	750000	4060000	9170000
LIMA-CPA-sp-2062	5	5114770	4729707	1553638	11398115
HQC Paranoiac I	5	2210000	4670000	6670000	13550000
HQC Paranoiac II	5	2520000	5370000	7510000	15400000
Round2-uround2-n1-fn1-l5	5	6589401	8665781	402090	15657272
LIMA-CCA-sp-2062	5	5114770	4738128	6237127	16090025
HQC Paranoiac III	5	2660000	5620000	8030000	16310000
HQC Paranoiac IV	5	2810000	5950000	8460000	17220000
LOCKER IX	5	10400000	1490000	6600000	18490000
Round2-uround2-n1-fn2-l5	5	9360000	10110000	340000	19810000
BIKE-1 5	5	2986647	3023816	17486906	23497369
BIKE-3 5	5	2300332	3257675	18047493	23605500
DING Key Exchange 1024	3/5	6813691	9541851	7617506	23973048
RQC-256	5	2820000	6460000	18000000	27280000
Round2-nround2-nd-l5	5	12370000	28260000	13	40630013

SIKEp964	5	10563749	14995526	17957283	43516558
LOTUS-256	5	71846095	584915	867464	73298474
BIKE-2 5	5	58806046	1201161	16485956	76493163
Mersenne-756839	5	17090755	25367142	56778896	99236793
Round2-uround2-n1-fn0-l5	5	55115889	56034085	433575	111583549
NTRU Prime ntrulpr4591761	5	14060919	44116905	71245370	129423194
NTS-KEM (13 ,136)	5	249939545	544406	2911120	253395071
Ramstake 756839	5	43148424	79342014	154721609	277212047
QC-MDPC KEM	5	131540379	20180017	229002269	380722665
NTRUEncrypt-1024	5	135483043	224147211	385916996	745547250
LEDAkem-5-4	5	623140012	115437155	386274394	1124851561
LEDAkem-5-2	5	902318487	112652298	280835007	1295805792
LEDAkem-5-3	5	1858107259	106848011	298953313	2263908583
Lizard-CATEGORY5-N1300	5	3810518400*	17180800*	18636800*	3846336000
RLCE-KEM-256A	5	5057459034	5362174	24174369	5086995577
Classic McEliece mceliece8192128	5	6008245724	296036	458556	6009000316
Lizard-CATEGORY5-N1088	5	6372454400*	17596800*	18824000*	6408875200
RLCE-KEM-256B	5	9612380645	8184051	36705481	9657270177
BIG QUAKE 5	5	16528607297	12772072	51333539	16592712908
DAGS 5	5	136498000000	49029613	260829051	136808000000

4.2.3 Signatures Running Times

For all Tables containing running times for signature implementations, *sec* denotes the claimed NIST level of security for the implementation (See Section 2.5.1), and *Key.Gen* (key generation, *Sign*) (signature generation), and *Verify* (verification) entries all denote number of cycles needed to complete each process. If left blank, the implementation does not fulfil any of the NIST security levels' requirements, or the data is not available.

Table 4.57 is a Table containing all signature algorithm implementations, and is sorted alphabetically after submission implementation names. This Table is meant as a referencing and look-up Table for all implementations.

- Levels 1 and 2:
  - Table 4.58, sorted by the number of needed cycles for key generation.
  - Table 4.59, sorted by the number of needed cycles for signature generation.
  - Table 4.60, sorted by the number of needed cycles for verification.
- Levels 3 and 4:
  - Table 4.61, sorted by the number of needed cycles for key generation.
  - Table 4.62, sorted by the number of needed cycles for signature generation.
  - Table 4.63, sorted by the number of needed cycles for verification.
- Level 5:
  - Table 4.64, sorted by the number of needed cycles for key generation.
  - Table 4.65, sorted by the number of needed cycles for signature generation.
  - Table 4.66, sorted by the number of needed cycles for verification.

Table 4.57: Signature implementation security levels, key generation, signature, and verification given in number of needed CPU cycles.

Submission Implementation	Sec	Key.Gen	Sign	Verify
CRYSTALS-DILITHIUM weak	-	169972	765442	196048
CRYSTALS-DILITHIUM medium	1	269844	1285476	296920
CRYSTALS-DILITHIUM high	2	382756	1817902	395936
CRYSTALS-DILITHIUM very high	3	512116	1677782	548558
DRS 128	1	1001828786	62867536	505869989
DRS 192	3	1910198595	95622249	814640083
DRS 256	5	3208544675	148424947	1419704155
DualModeMS 128	1	2435532588733	12468307435	10893369
FALCON 512	1	300030872	19884364	1384574
FALCON 768	2/3	91009209	8359971	666108
FALCON 1024	4/5	157623028	13058641	1117624
GeMSS 128	1	114893999	1660770752	1254877
GeMSS 192	3	567250472	4620657460	950111
GeMSS 256	5	1245472262	5522622728	2202925
Gravity-SPHINCS S	2	781646000*	1196400*	104000*
Gravity-SPHINCS M	2	24229712000*	18900000*	252000*
Gravity-SPHINCS L	2	11789080000*	21054000*	338000*
Gui-184	1	2408000	10910000	152000
Gui-312	3	43817000	25436000	846000
Gui-448	5	239502	872949	1787000
HiMQ-3	1	50593934	21594	17960
HiMQ-3F	1	79256175	25613	14645
HiMQ-3P	1	641010138	66179	23942
LUOV-8-63-256	2	21000000	5870000	4930000
LUOV-8-90-351	4	81800000	21600000	17300000
LUOV-8-117-404	5	146000000	36500000	29700000
LUOV-49-49-242	2	14800000	34100000	23600000
LUOV-64-68-330	4	50800000	111000000	66100000
LUOV-80-86-399	5	96800000	216000000	124000000
MQDSS-48	2	1206730	52466398	38686506
MQDSS-64	4	2806750	169298364	123239874
Picnic-L1-FS	1	163850	131390415	86062091
Picnic-L1-UR	1	146193	158826399	106128443
Picnic-L3-FS	3	364079	442257404	291723398
Picnic-L3-UR	3	369536	521842013	350635309
Picnic-L5-FS	5	722494	1073183185	70865264744
Picnic-L5-UR	5	740633	1187481996	797249015
PQRSA-SIGN-15	-	110000	530	3700
PQRSA-SIGN-20	-	110000	1000	5800
PQRSA-SIGN-25	-	540000	1400	15000
PQRSA-SIGN-30	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$	$6.11{\cdot}10^{14}$
pqNTRUsign Gaussian-1024	5	259672814	349028118	2955494
pqNTRUsign Uniform-1024	5	268329761	202185303	2726230

pqsigRM-4-12	1	9641836	15194705	81178
pqsigRM-6-12	3	1983428	77735436	116906
pqsigRM-6-13	5	22668519	1557210	540378
qTESLA-128	1	3402000	5870005	12433000
qTESLA-192	3	2495000	9686000	26063000
qTESLA-256	5	520000	1065000	1310000
RaCoSS	1	99260000000	243600000	127400000
Rainbow Ia	1	1302000000	601000	350000
Rainbow Ib	1	4578000000	2044000	1944000
Rainbow Ic	1	4089000000	1521000	939000
Rainbow IIIb	3	26172000000	5471000	4908000
Rainbow IIIc	3	31612000000	4047000	2974000
Rainbow IVa	4	11176000000	1823000	1241000
Rainbow Vc	5	116046000000	8688000	6174000
Rainbow VIa	5	45064000000	3916000	2897000
Rainbow VIb	5	164689000000	16755000	11224000
SPHINCS+ haraka-128s	1	917405356	16992635344	19360272
SPHINCS+ haraka-128f	1	28814020	1056761824	45964624
SPHINCS+ haraka-192s	3	1244530184	38062259596	27243200
SPHINCS+ haraka-192f	3	42782840	1276694620	69760728
SPHINCS+ haraka-256s	5	1817324180	28860355888	42380420
SPHINCS+ haraka-256f	5	113876252	3172247452	76203004
SPHINCS+ SHA256-128s	1	307425484	4606958168	5514124
SPHINCS+ SHA256-128f	1	9625644	302359220	12901012
SPHINCS+ SHA256-192s	3	576727832	12239247980	10740192
SPHINCS+ SHA256-192f	3	17902436	487388724	26456352
SPHINCS+ SHA256-256s	5	1095050628	12893347756	19141296
SPHINCS+ SHA256-256f	5	68819608	1558148364	38316192
SPHINCS+ shake256-128s	1	617619732	8610599004	10222936
SPHINCS+ shake256-128f	1	19348784	580904788	24826884
SPHINCS+ shake256-192s	3	907587276	17586416344	15036680
SPHINCS+ shake256-192f	3	28200752	757001640	40338224
SPHINCS+ shake256-256s	5	1210939356	13842403104	20889204
SPHINCS+ shake256-256f	5	75031996	1664510764	41469276
WalnutDSA BKL-128	1	2086564	137691863	96962
WalnutDSA BKL-256	5	4456087	472468875	197243
WalnutDSA STOC-128	1	2271199	51244842	101529
WalnutDSA STOC-256	5	4519863	134509781	194916
WalnutDSA STOC-wo-DEH-128	1	2574824	48246052	147948
WalnutDSA STOC-wo-DEH-256	5	4836298	130993063	311116

Table 4.58: NIST security category 1 and 2 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for key generation

Submission Implementation	**Sec**	**Key.Gen**	**Sign**	**Verify**
Picnic-L1-UR	1	146193	158826399	106128443
Picnic-L1-FS	1	163850	131390415	86062091
CRYSTALS-DILITHIUM medium	1	269844	1285476	296920
CRYSTALS-DILITHIUM high	2	382756	1817902	395936
MQDSS-48	2	1206730	52466398	38686506
WalnutDSA BKL-128	1	2086564	137691863	96962
WalnutDSA STOC-128	1	2271199	51244842	101529
Gui-184	1	2408000	10910000	152000
WalnutDSA STOC-wo-DEH-128	1	2574824	48246052	147948
qTESLA-128	1	3402000	5870005	12433000
SPHINCS+ SHA256-128f	1	9625644	302359220	12901012
pqsigRM-4-12	1	9641836	15194705	81178
LUOV-49-49-242	2	14800000	34100000	23600000
SPHINCS+ shake256-128f	1	19348784	580904788	24826884
LUOV-8-63-256	2	21000000	5870000	4930000
SPHINCS+ haraka-128f	1	28814020	1056761824	45964624
HiMQ-3	1	50593934	21594	17960
HiMQ-3F	1	79256175	25613	14645
FALCON 768	2/3	91009209	8359971	666108
GeMSS 128	1	114893999	1660770752	1254877
FALCON 512	1	300030872	19884364	1384574
SPHINCS+ SHA256-128s	1	307425484	4606958168	5514124
SPHINCS+ shake256-128s	1	617619732	8610599004	10222936
HiMQ-3P	1	641010138	66179	23942
Gravity-SPHINCS S	2	781646000*	1196400*	104000*
SPHINCS+ haraka-128s	1	917405356	16992635344	19360272
DRS 128	1	1001828786	62867536	505869989
Rainbow Ia	1	1302000000	601000	350000
Rainbow Ic	1	4089000000	1521000	939000
Rainbow Ib	1	4578000000	2044000	1944000
Gravity-SPHINCS L	2	11789080000*	21054000*	338000*
Gravity-SPHINCS M	2	24229712000*	18900000*	252000*
RaCoSS	1	99260000000	243600000	127400000
DualModeMS 128	1	2435532588733	12468307435	10893369
PQRSA-SIG-30	$5.87 \cdot 10^{14}$	$1.82 \cdot 10^{13}$	$2.23 \cdot 10^{13}$	$6.11 \cdot 10^{14}$

Table 4.59: NIST security category 1 and 2 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for signature generation.

Submission Implementation	**Sec**	**Key.Gen**	**Sign**	**Verify**
HiMQ-3	1	50593934	21594	17960
HiMQ-3F	1	79256175	25613	14645
HiMQ-3P	1	641010138	66179	23942
Rainbow Ia	1	1302000000	601000	350000
Gravity-SPHINCS S	2	781646000*	1196400*	104000*
Rainbow Ic	1	4089000000	1521000	939000
CRYSTALS-DILITHIUM medium	1	269844	1285476	296920
CRYSTALS-DILITHIUM high	2	382756	1817902	395936
Rainbow Ib	1	4578000000	2044000	1944000
LUOV-8-63-256	2	21000000	5870000	4930000
qTESLA-128	1	3402000	5870005	12433000
FALCON 768	2/3	91009209	8359971	666108
Gui-184	1	2408000	10910000	152000
pqsigRM-4-12	1	9641836	15194705	81178
Gravity-SPHINCS M	2	24229712000*	18900000*	252000*
FALCON 512	1	300030872	19884364	1384574
Gravity-SPHINCS L	2	11789080000*	21054000*	338000*
LUOV-49-49-242	2	14800000	34100000	23600000
WalnutDSA STOC-wo-DEH-128	1	2574824	48246052	147948
WalnutDSA STOC-128	1	2271199	51244842	101529
MQDSS-48	2	1206730	52466398	38686506
DRS 128	1	1001828786	62867536	505869989
Picnic-L1-FS	1	163850	131390415	86062091
WalnutDSA BKL-128	1	2086564	137691863	96962
Picnic-L1-UR	1	146193	158826399	106128443
RaCoSS	1	99260000000	243600000	127400000
SPHINCS+ SHA256-128f	1	9625644	302359220	12901012
SPHINCS+ shake256-128f	1	19348784	580904788	24826884
SPHINCS+ haraka-128f	1	28814020	1056761824	45964624
GeMSS 128	1	114893999	1660770752	1254877
SPHINCS+ SHA256-128s	1	307425484	4606958168	5514124
SPHINCS+ shake256-128s	1	617619732	8610599004	10222936
DualModeMS 128	1	2435532588733	12468307435	10893369
SPHINCS+ haraka-128s	1	917405356	16992635344	19360272
PQRSA-SIGN-30	$5.87 \cdot 10^{14}$	$1.82 \cdot 10^{13}$	$2.23 \cdot 10^{13}$	$6.11 \cdot 10^{14}$

Table 4.60: NIST security category 1 and 2 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for verification.

Submission Implementation	Sec	Key.Gen	Sign	Verify
HiMQ-3F	1	79256175	25613	14645
HiMQ-3	1	50593934	21594	17960
HiMQ-3P	1	641010138	66179	23942
pqsigRM-4-12	1	9641836	15194705	81178
WalnutDSA BKL-128	1	2086564	137691863	96962
WalnutDSA STOC-128	1	2271199	51244842	101529
Gravity-SPHINCS S	2	781646000*	1196400*	104000*
WalnutDSA STOC-wo-DEH-128	1	2574824	48246052	147948
Gui-184	1	2408000	10910000	152000
Gravity-SPHINCS M	2	24229712000*	18900000*	252000*
CRYSTALS-DILITHIUM medium	1	269844	1285476	296920
Gravity-SPHINCS L	2	11789080000*	21054000*	338000*
Rainbow Ia	1	1302000000	601000	350000
CRYSTALS-DILITHIUM high	2	382756	1817902	395936
FALCON 768	2/3	91009209	8359971	666108
Rainbow Ic	1	4089000000	1521000	939000
GeMSS 128	1	114893999	1660770752	1254877
FALCON 512	1	300030872	19884364	1384574
Rainbow Ib	1	4578000000	2044000	1944000
LUOV-8-63-256	2	21000000	5870000	4930000
SPHINCS+ SHA256-128s	1	307425484	4606958168	5514124
SPHINCS+ shake256-128s	1	617619732	8610599004	10222936
DualModeMS 128	1	2435532588733	12468307435	10893369
qTESLA-128	1	3402000	5870005	12433000
SPHINCS+ SHA256-128f	1	9625644	302359220	12901012
SPHINCS+ haraka-128s	1	917405356	16992635344	19360272
LUOV-49-49-242	2	14800000	34100000	23600000
SPHINCS+ shake256-128f	1	19348784	580904788	24826884
MQDSS-48	2	1206730	52466398	38686506
SPHINCS+ haraka-128f	1	28814020	1056761824	45964624
Picnic-L1-FS	1	163850	131390415	86062091
Picnic-L1-UR	1	146193	158826399	106128443
RaCoSS	1	99260000000	243600000	127400000
DRS 128	1	1001828786	62867536	505869989
PQRSA-SIG-30	$5.87{\cdot}10^{14}$	$1.82{\cdot}10^{13}$	$2.23{\cdot}10^{13}$	$6.11{\cdot}10^{14}$

Table 4.61: NIST security category 3 and 4 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for key generation

Submission Implementation	**Sec**	**Key.Gen**	**Sign**	**Verify**
Picnic-L3-FS	3	364079	442257404	291723398
Picnic-L3-UR	3	369536	521842013	350635309
CRYSTALS-DILITHIUM very high	3	512116	1677782	548558
pqsigRM-6-12	3	1983428	77735436	116906
qTESLA-192	3	2495000	9686000	26063000
MQDSS-64	4	2806750	169298364	123239874
SPHINCS+ SHA256-192f	3	17902436	487388724	26456352
SPHINCS+ shake256-192f	3	28200752	757001640	40338224
SPHINCS+ haraka-192f	3	42782840	1276694620	69760728
Gui-312	3	43817000	25436000	846000
LUOV-64-68-330	4	50800000	111000000	66100000
LUOV-8-90-351	4	81800000	21600000	17300000
FALCON 768	2/3	91009209	8359971	666108
FALCON 1024	4/5	157623028	13058641	1117624
GeMSS 192	3	567250472	4620657460	950111
SPHINCS+ SHA256-192s	3	576727832	12239247980	10740192
SPHINCS+ shake256-192s	3	907587276	17586416344	15036680
SPHINCS+ haraka-192s	3	1244530184	38062259596	27243200
DRS 192	3	1910198595	95622249	814640083
Rainbow IVa	4	11176000000	1823000	1241000
Rainbow IIIb	3	26172000000	5471000	4908000
Rainbow IIIc	3	31612000000	4047000	2974000

Table 4.62: NIST security category 3 and 4 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for signature generation.

Submission Implementation	Sec	Key.Gen	Sign	Verify
CRYSTALS-DILITHIUM very high	3	512116	1677782	548558
Rainbow IVa	4	11176000000	1823000	1241000
Rainbow IIIc	3	31612000000	4047000	2974000
Rainbow IIIb	3	26172000000	5471000	4908000
FALCON 768	2/3	91009209	8359971	666108
qTESLA-192	3	2495000	9686000	26063000
FALCON 1024	4/5	157623028	13058641	1117624
LUOV-8-90-351	4	81800000	21600000	17300000
Gui-312	3	43817000	25436000	846000
pqsigRM-6-12	3	1983428	77735436	116906
DRS 192	3	1910198595	95622249	814640083
LUOV-64-68-330	4	50800000	111000000	66100000
MQDSS-64	4	2806750	169298364	123239874
Picnic-L3-FS	3	364079	442257404	291723398
SPHINCS+ SHA256-192f	3	17902436	487388724	26456352
Picnic-L3-UR	3	369536	521842013	350635309
SPHINCS+ shake256-192f	3	28200752	757001640	40338224
SPHINCS+ haraka-192f	3	42782840	1276694620	69760728
GeMSS 192	3	567250472	4620657460	950111
SPHINCS+ SHA256-192s	3	576727832	12239247980	10740192
SPHINCS+ shake256-192s	3	907587276	17586416344	15036680
SPHINCS+ haraka-192s	3	1244530184	38062259596	27243200

Table 4.63: NIST security category 3 and 4 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for verification.

Submission Implementation	**Sec**	**Key.Gen**	**Sign**	**Verify**
pqsigRM-6-12	3	1983428	77735436	116906
CRYSTALS-DILITHIUM very high	3	512116	1677782	548558
FALCON 768	2/3	91009209	8359971	666108
Gui-312	3	43817000	25436000	846000
GeMSS 192	3	567250472	4620657460	950111
FALCON 1024	4/5	157623028	13058641	1117624
Rainbow IVa	4	11176000000	1823000	1241000
Rainbow IIIc	3	31612000000	4047000	2974000
Rainbow IIIb	3	26172000000	5471000	4908000
SPHINCS+ SHA256-192s	3	576727832	12239247980	10740192
SPHINCS+ shake256-192s	3	907587276	17586416344	15036680
LUOV-8-90-351	4	81800000	21600000	17300000
qTESLA-192	3	2495000	9686000	26063000
SPHINCS+ SHA256-192f	3	17902436	487388724	26456352
SPHINCS+ haraka-192s	3	1244530184	38062259596	27243200
SPHINCS+ shake256-192f	3	28200752	757001640	40338224
LUOV-64-68-330	4	50800000	111000000	66100000
SPHINCS+ haraka-192f	3	42782840	1276694620	69760728
MQDSS-64	4	2806750	169298364	123239874
Picnic-L3-FS	3	364079	442257404	291723398
Picnic-L3-UR	3	369536	521842013	350635309
DRS 192	3	1910198595	95622249	814640083

Table 4.64: NIST security category 5 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for key generation

Submission Implementation	Sec	Key.Gen	Sign	Verify
Gui-448	5	239502	872949	1787000
qTESLA-256	5	520000	1065000	1310000
Picnic-L5-FS	5	722494	1073183185	70865264744
Picnic-L5-UR	5	740633	1187481996	797249015
WalnutDSA BKL-256	5	4456087	472468875	197243
WalnutDSA STOC-256	5	4519863	134509781	194916
WalnutDSA STOC-wo-DEH-256	5	4836298	130993063	311116
pqsigRM-6-13	5	22668519	1557210	540378
SPHINCS+ SHA256-256f	5	68819608	1558148364	38316192
SPHINCS+ shake256-256f	5	75031996	1664510764	41469276
LUOV-80-86-399	5	96800000	216000000	124000000
SPHINCS+ haraka-256f	5	113876252	3172247452	76203004
LUOV-8-117-404	5	146000000	36500000	29700000
FALCON 1024	4/5	157623028	13058641	1117624
pqNTRUsign Gaussian-1024	5	259672814	349028118	2955494
pqNTRUsign Uniform-1024	5	268329761	202185303	2726230
SPHINCS+ SHA256-256s	5	1095050628	12893347756	19141296
SPHINCS+ shake256-256s	5	1210939356	13842403104	20889204
GeMSS 256	5	1245472262	5522622728	2202925
DRS 256	5	3208544675	148424947	1419704155
SPHINCS+ haraka-256s	5	1817324180	28860355888	42380420
Rainbow VIa	5	45064000000	3916000	2897000
Rainbow Vc	5	116046000000	8688000	6174000
Rainbow VIb	5	164689000000	16755000	11224000

Table 4.65: NIST security category 5 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for signature generation.

Submission Implementation	Sec	Key.Gen	Sign	Verify
Gui-448	5	239502	872949	1787000
qTESLA-256	5	520000	1065000	1310000
pqsigRM-6-13	5	22668519	1557210	540378
Rainbow VIa	5	45064000000	3916000	2897000
Rainbow Vc	5	116046000000	8688000	6174000
FALCON 1024	4/5	157623028	13058641	1117624
Rainbow VIb	5	164689000000	16755000	11224000
LUOV-8-117-404	5	146000000	36500000	29700000
WalnutDSA STOC-wo-DEH-256	5	4836298	130993063	311116
WalnutDSA STOC-256	5	4519863	134509781	194916
DRS 256	5	3208544675	148424947	1419704155
LUOV-80-86-399	5	96800000	216000000	124000000
pqNTRUsign Uniform-1024	5	268329761	202185303	2726230
pqNTRUsign Gaussian-1024	5	259672814	349028118	2955494
WalnutDSA BKL-256	5	4456087	472468875	197243
Picnic-L5-FS	5	722494	1073183185	70865264744
Picnic-L5-UR	5	740633	1187481996	797249015
SPHINCS+ SHA256-256f	5	68819608	1558148364	38316192
SPHINCS+ shake256-256f	5	75031996	1664510764	41469276
SPHINCS+ haraka-256f	5	113876252	3172247452	76203004
GeMSS 256	5	1245472262	5522622728	2202925
SPHINCS+ SHA256-256s	5	1095050628	12893347756	19141296
SPHINCS+ shake256-256s	5	1210939356	13842403104	20889204
SPHINCS+ haraka-256s	5	1817324180	28860355888	42380420

Table 4.66: NIST security category 5 signature implementation security levels, key generation, signature generation, and verification given in number of needed CPU cycles, sorted after running times for verification.

Submission Implementation	Sec	Key.Gen	Sign	Verify
WalnutDSA STOC-256	5	4519863	134509781	194916
WalnutDSA BKL-256	5	4456087	472468875	197243
WalnutDSA STOC-wo-DEH-256	5	4836298	130993063	311116
pqsigRM-6-13	5	22668519	1557210	540378
FALCON 1024	4/5	157623028	13058641	1117624
qTESLA-256	5	520000	1065000	1310000
Gui-448	5	239502	872949	1787000
GeMSS 256	5	1245472262	5522622728	2202925
pqNTRUsign Uniform-1024	5	268329761	202185303	2726230
Rainbow VIa	5	45064000000	3916000	2897000
pqNTRUsign Gaussian-1024	5	259672814	349028118	2955494
Rainbow Vc	5	116046000000	8688000	6174000
Rainbow VIb	5	164689000000	16755000	11224000
SPHINCS+ SHA256-256s	5	1095050628	12893347756	19141296
SPHINCS+ shake256-256s	5	1210939356	13842403104	20889204
LUOV-8-117-404	5	146000000	36500000	29700000
SPHINCS+ SHA256-256f	5	68819608	1558148364	38316192
SPHINCS+ shake256-256f	5	75031996	1664510764	41469276
SPHINCS+ haraka-256s	5	1817324180	28860355888	42380420
SPHINCS+ haraka-256f	5	113876252	3172247452	76203004
LUOV-80-86-399	5	96800000	216000000	124000000
Picnic-L5-UR	5	740633	1187481996	797249015
DRS 256	5	3208544675	148424947	1419704155
Picnic-L5-FS	5	722494	1073183185	70865264744

Chapter 5

Results and Discussion

Seeing as the development, testing, and implementation of post-quantum cryptography is a long and arduous process, there are no definite answers to be given as of yet. This Chapter will contain a summation of the information presented and compared in the previous Chapters 2, 3, and 4, as well as discussions on these new developments as a whole. It also briefly touches upon the continuation of the development of post-quantum cryptography.

5.1 General Discussion on the Post-Quantum Standardisation Process

When NIST announced their call for submissions for the Post-Quantum Standardisation, it was the beginning of what is sure to be a lengthy process, which will continue far past the duration of this master's thesis. Even NIST has defined the plan set for evaluation of the submission as tentative, stating that: *"It should be noted that this schedule for the evaluation process is somewhat tentative, depending upon the type, quantity, and quality of the submissions."* [185]. This is part of the reason why it is not feasible to commit to any of the submissions as, nor claim any of them as the "best" in their respective categories. Further extensive testing and research is needed before one can claim any of the implementations as secure.

Even after this, due to the complex nature of post-quantum cryptography as compared to traditional cryptography, it is believed that many algorithms are to be selected for standardisation within different fields and for different uses. In their initial call for proposals, NIST stated that: *"It is intended that the new public-key cryptography standards will specify* ***one or more*** *additional unclassified, publicly disclosed digital signature, public-key encryption, and key-establishment algorithms that are available worldwide, and are capable of protecting sensitive government information well into the foreseeable future, including after the advent of quantum computers."* [186] Note the emboldened text in this statement, which specifies that they do not exclude the possibility of selecting more than one algorithm in every category mentioned. The tentative plan for the rest of the evaluation is to begin round 2 of the evaluation during this year or the next, with a second PQC conference in the autumn of 2019. The 3rd stage of the evaluation will begin during

2020/2021. A tentative estimate for drafts for the standards is set to sometime between 2022 and 2024.

5.2 About the Different Categories of Post-Quantum Cryptography

While there is no definite answer to the question of which of the different categories is the best, it is possible to look at emerging trends in which ones are favoured over others. Of all the submissions given to the NIST Post-Quantum Cryptography Standardisation, there were 26 lattice-based, 20 code-based, 9 multivariate, and 3 hash-based. Using no other data than this, we can see a clear bias towards lattice-based and code-based cryptography. With code-based cryptography being such a varied category, it is clear that lattice-based systems are preferred by many. As one of the older categories, it is also much more elaborately studied than younger categories such as isogeny-based cryptography, simply due to the difference in time used. At the same time, many of these algorithms utilise keys which are not significantly longer than those used by many non-quantum-resistant algorithms.

5.3 Submission Implementations Analysis Discussion

As the process of determining which algorithms are the most promising is still ongoing as of this master begin completed, there can be no definite answer to the comparative analysis, in regard to which algorithms are most suited for varying tasks within each category. This paper does, however, give an overview of the different categories, their mathematical groundwork, and which algorithms are designed for what general purposes.

The comparative analysis given in Chapter 4 is intended as an examination of all the currently proposed implementations for all algorithms, and gives insight into their general priorities and purposes. Algorithms intended for long time storage of secret information often use larger key sizes and are not as time-effective as algorithms which are intended for use in live, time-sensitive transmissions, IoT-devices, or less sensitive, short-lived data. These trends can be seen in all three main categories (Encryption, KEM, and signatures), and are often mirrors of the priorities or intended use given in the description of each submission (See Chapter 3).

To provide a better overview of the submissions, a list of some of the current top contenders in each of the three categories of submissions will be given. Only the best ranked submission implementations for the top performing submissions have been included in the rankings. For initial implementation and standardisation, NIST security level 1 and 2 algorithm implementations are given (See Sections 5.3.1.1, 5.3.2.1, and 5.3.3.1). For long-term implementation and standardisation, NIST security level 5 algorithm implementations are given (See sections 5.3.1.2, 5.3.2.2, and 5.3.3.2). All submissions are chosen and ranked tentatively according to the following characteristics:

- The submission have not been withdrawn

- There are no discovered attacks against the submission which break the claimed security of the original submission
- There are no discovered attacks against the submission which expose a flaw which may be exploited
- There exists reliable data on the submission implementations for the following parameters:
  * For an encryption submission:

    Size of private key, public key, and ciphertext

    Execution times for key generation, encryption, and decryption
  * For a KEM submission:

    Size of private key, public key, and ciphertext

    Execution times for key generation, encapsulation, and decapsulation
  * For a signature submission:

    Size of private key, public key, and signature

    Execution times for key generation, signature, and verification
- The submission's efficiency (cycles per byte) is compared to create a sorting within the submissions fulfilling these requirements

Detailed explanations of the comparative Tables and rankings can be found in each subsequent subsection.

It is worth noting, however, that measuring after efficiency as cycles per byte does favour implementations intended for short-term storage and lower security somewhat. The author acknowledges this, which is why this is meant as a tentative ranking. This is also the reason why the absence of attacks already found on the submissions is the main criteria for the chosen submissions.

Please note that this is only as it stands at the time of writing (June, 2018), and that any later attacks/mitigations/changes/weaknesses for these algorithms can not be taken into account. The methodology used to create the rankings presented in this thesis can however be utilised for any future versions of the submitted algorithms.

5.3.1 Encryption

5.3.1.1 NIST Security Level 1 and 2

The top 10 encryption submissions as sorted after the size of the sum of private keys, public keys, and ciphertexts are given below. Their best implementations according to the same requirements can be seen in Table 5.2.

1. McNie
2. NTRUEncrypt
3. SABER
4. LAC
5. KINDI
6. Lizard
7. LEDApkc
8. Titanium
9. LIMA
10. Giophantus

Table 5.2: Space requirements for the top 10 NIST level 1 and 2 encryption submissions' implementations sorted after the size of the sum of private key, public key, and ciphertext (Sum).

Submission Implementation	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**Sum**
McNie-4Q-128-1	340	347	390	1077
McNie-3Q-128-1	194	431	547	1172
McNie-4Q-128-2	401	417	473	1291
McNie-3Q-128-2	218	486	621	1325
NTRUEncrypt-443	701	611	611	1923
SABER light	832	672	736	2240
LAC-CPA-128	1056	544	1024	2624
KINDI-256-3-4-2	1472	1184	1792	4448
RLizard-CATEGORY1	257	4096	2208	6561
LEDApkc-1-2	668	3480	6960	11108
LEDApkc-1-3	844	4688	7032	12564
LEDApkc-1-4	1036	6408	8544	15988
Titanium CPA standard	32	14720	3520	18272
Titanium CPA medium	32	16448	4512	20992
LIMA-CCA-sp-1018	9163	6109	6105	21377
LIMA-CPA-sp-1018	9163	6109	6105	21377
LIMA-CCA-sp-1306	15673	10449	10443	36565
LIMA-CPA-sp-1306	15673	10449	10443	36565
Giophantus 602	602	14412	28824	43838

The top 10 encryption submissions as sorted after the sum of execution times (number of needed cycles) are given below. Their best implementations according to the same requirements can be seen in Table 5.4.

1. LAC
2. KINDI
3. Lizard
4. LIMA
5. NTRUEncrypt
6. Titanium
7. LOTUS
8. LEDApkc
9. Odd Manhatten
10. Giopantus

Table 5.4: Execution times for the top 10 NIST level 1 and 2 encryption submissions' implementations given in number of needed cycles, sorted after the sum of all execution times (Sum).

Submission Implementation	Key.Gen	Encrypt	Decrypt	Sum
LAC-CPA-128	90686	152575	68285	311546
KINDI-256-3-4-2	203096	247793	312211	763100
RLizard-CATEGORY1	914860	347269	96689	1358818
LIMA-CPA-sp-1018	1429742	1236494	395944	3062180
NTRUEncrypt-443	2454400*	436800*	566800*	3458000*
Titanium CPA standard	1981835	1508258	261583	3751676
LIMA-CCA-sp-1018	1429742	1239122	1610046	4278910
Titanium CPA medium	2221874	2009472	301930	4533276
LIMA-CPA-sp-1306	2600237	2354094	770324	5724655
LIMA-CCA-sp-1306	2600237	2360157	3091742	8052136
LOTUS-128	26820400	316252	382582	27519234
Lizard-CATEGORY1-N536	105700625	326246	144127	106170998
Lizard-CATEGORY1-N663	119955905	318857	146639	120421401
LEDApkc-1-4	48451595	10481941	69485795	128419331
LEDApkc-1-3	60872340	8268701	61401087	130542128
LEDApkc-1-2	129344098	8597636	50484279	188426013
Odd Manhatten-128	201062400*	71794800*	77884400*	350741600*
Giophantus 602	92909566	178456036	335353573	606719175

A sorting based on the intersection of the two previous rankings for space requirements and execution times (sum of space requirements and sum of cycles). The intersection is the calculated cycles per byte (cpb). Any submission implementations which are not present in both the top ranking implementations for space requirements and execution times have been removed.

The top 5 performers using this category is shown below in order of best to worst. The calculated cpbs for each implementation is shown in Table 5.6.

1. Lizard
2. LIMA
3. LAC
4. LIMA

5. KINDI

Table 5.6: Calculated cpb for all level 1 and 2 encryption submissions' implementations which qualify as top 10 in both space requirements and execution times, using the sum of space requirements[B] and the sum of cycles needed, sorted after the calculated cpb (cpb).

Submission Implementation	**space reqs.[B]**	**exec.times[cycles]**	**cpb[cycles/B]**
Lizard-CATEGORY1-N663	2052823	120421401	58.66
LIMA-CCA-sp-1306	36565	2600237	71.11
LAC-CPA-128	2624	311546	118.73
LIMA-CPA-sp-1018	21377	3062180	143.25
LIMA-CPA-sp-1306	36565	5724655	156.56
KINDI-256-3-4-2	4448	763100	171.56
LIMA-CCA-sp-1018	21377	4278910	200.16
Titanium CPA standard	18272	3751676	205.32
RLizard-CATEGORY1	6561	1358818	207.11
Titanium CPA medium	20992	4533276	215.95
Lizard-CATEGORY1-N536	300880	106170998	352.87
NTRUEncrypt-443	1923	3458000*	1798.23*
LEDApkc-1-4	15988	128419331	8032.23
LEDApkc-1-3	12564	130542128	10390.17
Giophantus 602	43838	606719175	13840.03
LEDApkc-1-2	11108	188426013	16963.09

5.3.1.2 NIST Security Level 5

The top 10 encryption submissions as sorted after the size of the sum of private keys, public keys, and ciphertexts are given below. Their best implementations according to the same requirements can be seen in Table 5.8.

1. McNie
2. SABER
3. LAC
4. KINDI
5. NTRUEncrypt
6. Lizard
7. Titanium
8. LEDApkc
9. LIMA
10. Giphantus

Table 5.8: Space requirements for the top 10 NIST level 5 encryption submissions' implementations sorted after the size of the sum of private key, public key, and ciphertext (Sum).

Submission Implementation	$k_{private}$[B]	k_{public}[B]	c[B]	Sum
McNie-4Q-256-1	584	630	619	1833
McNie-4Q-256-2	601	647	749	1997
McNie-3Q-256-1	337	819	1065	2221
McNie-3Q-256-2	348	829	1078	2255
SABER fire	1664	1312	1472	4448
LAC-CPA-256	2080	1056	2048	5184
KINDI-256-5-2-2	1712	1456	2496	5664
KINDI-512-3-2-1	2752	2368	3328	8448
NTRUEncrypt-1024	8194	4097	4097	16388
RLizard-CATEGORY5	513	8192	8512	17217
Titanium CPA super	32	23552	8320	31904
LEDApkc-5-2	1244	12384	24768	38396
LIMA-CCA-2p-2048	18433	12289	12291	43013
LIMA-CPA-2p-2048	18433	12289	12291	43013
LEDApkc-5-3	1548	18016	27024	46588
LEDApkc-5-4	1772	22704	30272	54748
LIMA-CCA-sp-2062	24745	16497	16475	57717
LIMA-CPA-sp-2062	24745	16497	16475	57717
Giophantus 1134	1134	27204	54408	82746

The top 10 encryption submissions as sorted after the sum of execution times (number of needed cycles) are given below. Their best implementations according to the same requirements can be seen in Table 5.10.

1. LAC
2. KINDI

3. Lizard
4. LIMA
5. Titanium
6. LOTUS
7. LEDApkc
8. NTRUEncrypt
9. Odd Manhatten
10. Giophantus

Table 5.10: Execution times for the top 10 NIST level 5 encryption submissions' implementations given in number of needed cycles, sorted after the sum of all execution times (Sum)

Submission Implementation	**Key.Gen**	**Encrypt**	**Decrypt**	**Sum**
LAC-CPA-256	269827	513753	336207	1119787
KINDI-256-5-2-2	519010	595043	701763	1815816
KINDI-512-3-2-1	723922	530173	672720	1926815
RLizard-CATEGORY5	1084552	804982	568140	2457674
LIMA-CPA-2p-2048	1325909	971660	310484	2608053
LIMA-CCA-2p-2048	1325909	977969	1242139	3546017
Titanium CPA super	3054311	2917708	534948	6506967
LIMA-CPA-sp-2062	5114770	4726471	1549057	11390298
LIMA-CCA-sp-2062	5114770	4728991	6235608	16079369
LOTUS-256	72229496	626112	882523	73738131
LEDApkc-5-4	560609350	107088891	366509289	1034207530
LEDApkc-5-3	802604872	99518907	262949682	1165073461
NTRUEncrypt-1024	224640000*	348400000*	598000000*	1171040000*
Odd Manhatten-256	593124400*	283250000*	310604800*	1186979200*
LEDApkc-5-2	1599125506	92839822	264937652	1956902980
Giophantus 1134	239510004	626677271	1186128486	2052315761

A sorting based on the intersection of the two previous rankings for space requirements and execution times (sum of space requirements and sum of cycles). The intersection is the calculated cycles per byte (cpb). Any submission implementations which are not present in both the top ranking implementations for space requirements and execution times have been removed.

The top 5 performers using this category is shown below in order of best to worst. The calculated cpbs for each implementation is shown in Table 5.12.

1. LOTUS
2. LIMA
3. Odd Manhatten
4. Lizard
5. Titanium

Table 5.12: Calculated cpb for all level 5 encryption submissions' implementations which qualify as top 10 in both space requirements and execution times, using the sum of space requirements[B] and the sum of cycles needed, sorted after the calculated cpb (cpb)

Submission Implementation	**space reqs.[B]**	**exec.times[cycles]**	**cpb[cycles/B]**
LOTUS-256	3103464	73738131	23.76
LIMA-CPA-2p-2048	43013	2608053	60.63
LIMA-CCA-2p-2048	43013	3546017	82.44
Odd Manhatten-256	9527595	1186979200*	124.58*
RLizard-CATEGORY5	17217	2457674	142.75
LIMA-CPA-sp-2062	57717	11390298	197.35
Titanium CPA super	31904	6506967	203.95
LAC-CPA-256	5184	1119787	216.01
KINDI-512-3-2-1	8448	1926815	228.08
LIMA-CCA-sp-2062	57717	16079369	278.59
KINDI-256-5-2-2	5664	1815816	320.59
LEDApkc-5-4	54748	1034207530	18890.33
Giophantus 1134	82746	2052315761	24802.60
LEDApkc-5-3	46588	1165073461	25008.02
LEDApkc-5-2	38396	1956902980	50966.32
NTRUEncrypt-1024	16388	1171040000*	71457.16*

5.3.2 KEM

5.3.2.1 NIST Security Level 1 and 2

The top 20 KEM submissions as sorted after the size of the sum of public keys and ciphertexts (pk+c[B]) are given below. Their best implementations according to the same requirements can be seen in Table 5.14.

1. SIKE
2. LAKE
3. Round2
4. NTRUEncrypt
5. DME
6. SABER
7. CRYSTALS-KRYBER
8. LAC
9. LOCKER
10. Three Bears
11. Oroboros-R
12. NewHope
13. DING Key Exchange
14. RQC
15. NTRU-HRSS-KEM
16. BIKE
17. KINDI
18. Lepton
19. Lizard
20. LEDAkem

Table 5.14: Space requirements for the top 20 NIST level 1 and 2 KEM submissions' implementations sorted after the size of the sum of public key and ciphertext (pk+c[B]).

Submission Implementation	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
SIKEp503	434	378	402	780
LAKE I	40	423	423	846
Round2-nround2-nd-l1	100	417	464	881
Round2-uround2-nd-l1	105	435	482	917
Round2-nround2-nd-l2	122	519	614	1133
Round2-uround2-nd-l2	131	555	618	1173
NTRUEncrypt-443	701	611	611	1222
DME-144	144	1152	144	1296
SABER light	1568	672	736	1408
CRYSTALS-KYBER 512	1632	736	800	1536
LAC-CCA-128	1056	544	1024	1568
LOCKER I	787	747	875	1622
Three Bears BabyBear	40	804	917	1721
Three Bears BabyBear Ephem	40	804	917	1721
Ouroboros-R-128	40	676	1272	1948
NewHope-CPA-512	869	928	1088	2016
NewHope-CCA-512	1120	928	1120	2048
DING Key Exchange 512	1536	1040	1088	2128
LOCKER IV	1050	1010	1138	2148
RQC-128	826	786	1556	2342
NTRU-HRSS-KEM-701	1418	1138	1278	2416
BIKE-2 1	267	1272	1272	2544
KINDI-256-3-4-2	1472	1184	1792	2976
Lepton.CPA Light II	40	1045	1966	3011
Lepton-CCA Light II	1085	1045	1998	3043

BIKE-1 1	267	2542	2542	5084
BIKE-3 1	252	2758	2758	5516
RLizard-CATEGORY1	385	4096	2080	6176
LEDAkem-1-2	668	3480	3480	6960
LEDAkem-1-3	844	4688	2344	7032

The top 20 KEM submissions as sorted after the sum of execution times (number of needed cycles) are given below. Their best implementations according to the same requirements can be seen in Table 5.16.

1. Three Bears
2. Lepton
3. NewHope
4. SABER
5. LAC
6. CRYSTALS-KYBER
7. Round2
8. KINDI
9. RLizard
10. NTRUENcrypt
11. LIMA
12. Ouroboros-R
13. Titanium
14. HQC
15. BIKE
16. FrodoKEM
17. LOCKER
18. SIKE
19. RQC
20. DING Key Exchange

Table 5.16: Execution times for the top 20 NIST level 1 and 2 KEM submissions' implementations given in number of needed cycles, sorted after the sum of all execution times (Sum).

Submission Implementation	**Key.Gen**	**Encap**	**Decap**	**Sum**
Three Bears BabyBear Ephem	41000	62000	34000	137000
Lepton.CPA Light II	34912	85347	42462	162721
Three Bears BabyBear	41000	60000	101000	202000
Lepton.CPA Moderate I	48932	117275	45519	211726
Lepton-CCA Light II	34536	86584	100141	221261
Lepton.CPA Moderate II	51519	125178	51353	228050
NewHope-CPA-512	106820	155840	40988	303648
Lepton-CCA Moderate I	49943	121564	132708	304215
SABER light	105881	155131	179415	440427
LAC-CCA-128	90411	160314	216957	467682
CRYSTALS-KYBER 512	141872	205468	246040	593380
NewHope-CCA-512	222922	330828	87080	640830
Round2-uround2-nd-l1	330000	360000	50000	740000
KINDI-256-3-4-2	203096	260137	323947	787180
Round2-uround2-nd-l2	440000	500000	80000	1020000
RLizard-CATEGORY1	939058	533152	122781	1594991
NTRUEncrypt-443	1257307	394406	363281	2014994
LIMA-CPA-sp-1018	1429742	1233953	396764	3060459
LAKE I	1580000	300000	1270000	3150000
Ouroboros-R-128	600000	980000	1780000	3360000
Titanium CCA medium	2221874	1009472	301930	3533276

HQC Basic I	570000	1220000	1950000	3740000
Titanium CCA standard	1981835	1508258	261583	3751676
HQC Basic II	610000	1280000	2070000	3960000
HQC Basic III	630000	1350000	2150000	4130000
LIMA-CCA-sp-1018	1429742	1241867	1612433	4284042
BIKE-1 1	730025	689193	2901203	4320421
BIKE-3 1	433258	575237	3437956	4446451
FrodoKEM 640 AES	1287000	1810000	1811000	4908000
Round2-uround2-n1-fn1-l1	1865144	3025972	220117	5111233
LIMA-CPA-sp-1306	2600237	2355666	770710	5726613
LOCKER I	2710000	550000	2570000	5830000
SIKEp503	1561680	2207324	2663521	6432525
LOCKER IV	3720000	710000	2860000	7290000
Round2-uround2-n1-fn2-l1	3430000	4300000	180000	7910000
LIMA-CCA-sp-1306	2600237	2361683	3085679	8047599
RQC-128	790000	1970000	5300000	8060000
Round2-uround2-n1-fn1-l2	3969339	4861933	319314	9150586
BIKE-2 1	6383408	281755	2674115	9339278
LOCKER VII	8440000	1350000	4780000	14570000
DING Key Exchange 512	4399965	5735092	4774104	14909161

A sorting is created based on the intersection of the two previous rankings for space requirements and execution times (pk+c[B] and sum of cycles). The intersection is the calculated cycles per byte (cpb). Any submission implementations which are not present in both the top ranking implementations for space requirements and execution times have been removed.

The top 10 performers using this category is shown below in order of best to worst. The calculated cpbs for each implementation is shown in Table 5.18.

1. Lepton
2. Three Bears
3. NewHope
4. Lizard
5. KINDI
6. LAC
7. SABER
8. CRYSTALS-KYBER
9. BIKE
10. Round2

Table 5.18: Calculated cpb for all level 1 and 2 KEM submissions' implementations which qualify as top 10 in both space requirements and execution times, using pk+c and the sum of cycles needed, sorted after the calculated cpb (cpb).

Submission Implementation	**pk+c[B]**	**exec.times[cycles]**	**cpb[cycles/B]**
Lepton.CPA Light II	3011	162721	54.04

Lepton-CCA Light II	3043	221261	72.71
Three Bears BabyBear Ephem	1721	137000	79.60
Three Bears BabyBear	1721	202000	117.37
NewHope-CPA-512	2016	303648	150.62
RLizard-CATEGORY1	6176	1594991	258.26
KINDI-256-3-4-2	2976	787180	264.51
LAC-CCA-128	1568	467682	298.27
SABER light	1408	440427	312.80
NewHope-CCA-512	2048	640830	312.91
CRYSTALS-KYBER 512	1536	593380	386.32
BIKE-3 1	5516	4446451	806.10
Round2-uround2-nd-l1	917	740000	806.98
BIKE-1 1	5084	4320421	849.81
Round2-uround2-nd-l2	1173	1020000	869.57
NTRUEncrypt-443	1222	2014994	1648.93
Ouroboros-R-128	1948	3360000	1724.85
LOCKER IV	2148	7290000	3393.85
RQC-128	2342	8060000	3441.50
LOCKER I	1622	5830000	3594.33
BIKE-2 1	2544	9339278	3671.10
LAKE I	846	3150000	3723.40
DING Key Exchange 512	2128	14909161	7006.18
SIKEp503	780	6432525	8246.83

5.3.2.2 NIST Security Level 5

The top 20 KEM submissions as sorted after the size of the sum of public keys and ciphertexts (pk+c[B]) are given below. Their best implementations according to the same requirements can be seen in Table 5.20.

1. SIKE
2. Round2
3. LAKE
4. NTRU Prime
5. DME
6. LOCKER
7. SABER
8. CRYSTALS-KYBER
9. LAC
10. Ouroboros
11. Three Bears
12. KCL
13. HILA
14. KINDI
15. NewHope
16. DING Key Exchange
17. RQC
18. Lepton
19. BIKE
20. NTRUEncrypt

Table 5.20: Space requirements for the top 20 NIST level 5 KEM submissions' implementations sorted after the size of the sum of public key and ciphertext (pk+c[B]).

Submission Implementation	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**c[B]**	**pk+c[B]**
SIKEp964	826	726	766	1492
Round2-nround2-nd-l5	165	691	818	1509
Round2-uround2-nd-l5	169	709	868	1577
LAKE III	40	826	826	1652
NTRU Prime ntrulpr4591761	1238	1047	1175	2222
NTRU Prime sntrup4591761	1600	1218	1047	2265
DME-288	288	2304	288	2592
LOCKER III	1286	1246	1374	2620
SABER fire	3040	1312	1472	2784
CRYSTALS-KYBER 1024	3168	1440	1504	2944
LOCKER VI	1482	1442	1570	3012
LAC-CCA-256	2080	1056	2048	3104
Ouroboros-R-256	40	1112	2144	3256
Three Bears PapaBear	40	1584	1697	3281
Three Bears PapaBear Ephem	40	1584	1697	3281
KCL OKCN-RLWE	1664	1696	1995	3691
KCL AKCN-RLWE	1664	1696	2083	3779
HILA5	1824	1824	2012	3836
KINDI-256-5-2-2	1712	1456	2496	3952
NewHope-CPA-1024	1792	1824	2176	4000
NewHope-CCA-1024	3680	1824	2208	4032
DING Key Exchange 1024	3072	2064	2176	4240
RQC-256	1835	1795	2574	4369
LOCKER IX	2238	2198	2326	4524
KINDI-512-3-2-1	2752	2368	3328	5696
Lepton.CPA Moderate IV	74	2052	3989	6041

Lepton-CCA Moderate IV	2126	2052	4021	6073
BIKE-2 5	548	4096	4096	8192
NTRUEncrypt-1024	8194	4097	4097	8194

The top 20 KEM submissions as sorted after the sum of execution times (number of needed cycles) are given below. Their best implementations according to the same requirements can be seen in Table 5.22.

1. Lepton
2. Three Bears
3. NewHope
4. KCL
5. SABER
6. CRYSTALS-KYBER
7. Round2
8. LAC
9. KINDI
10. HILA5
11. LIMA
12. Lizard
13. LAKE
14. NTRU Prime
15. Titanium
16. Ouroboros-R
17. LOCKER
18. HQC
19. BIKE
20. Ding Key Exchange

Table 5.22: Execution times for the top 20 NIST level 5 KEM submissions' implementations given in number of needed cycles, sorted after the sum of all execution times (Sum).

Submission Implementation	**Key.Gen**	**Encap**	**Decap**	**Sum**
Lepton.CPA Moderate IV	57861	152431	72564	282856
Three Bears PapaBear Ephem	125000	154000	40000	319000
Lepton-CCA Moderate IV	59450	154473	179520	393443
Lepton.CPA Paranoid I	96602	237722	97757	432081
Lepton.CPA Paranoid II	97884	247932	105200	451016
Three Bears PapaBear	119000	145000	213000	477000
NewHope-CPA-1024	117128	180648	206244	504020
Lepton-CCA Paranoid I	94454	234441	264881	593776
Lepton-CCA Paranoid II	97569	244706	282199	624474
KCL AKCN-RLWE	338215	395116	83455	816786
NewHope-CCA-1024	244944	377092	437056	1059092
SABER fire	360539	400817	472366	1233722
KCL OKCN-RLWE	433536	715307	192306	1341149
CRYSTALS-KYBER 1024	368564	481042	558740	1408346
Round2-uround2-nd-l5	630000	720000	100000	1450000
LAC-CCA-256	267831	526915	874742	1669488
KINDI-256-5-2-2	519010	623436	723922	1866368
KINDI-512-3-2-1	723922	562640	698041	1984603
HILA5	934320*	1222640*	229840*	2386800*
LIMA-CPA-2p-2048	1325909	1117377	481230	2924516
RLizard-CATEGORY5	1336795	1060163	660404	3057362
LIMA-CCA-2p-2048	1325909	1262893	1229593	3818395
LAKE III	1790000	350000	2890000	5030000
NTRU Prime sntrup4591761	6000000	59456	97684	6157140

Titanium CCA super	3054311	2917708	534948	6506967
Ouroboros-R-256	820000	1390000	4730000	6940000
LOCKER III	3580000	600000	3770000	7950000
LOCKER VI	4360000	750000	4060000	9170000
LIMA-CPA-sp-2062	5114770	4729707	1553638	11398115
HQC Paranoiac I	2210000	4670000	6670000	13550000
HQC Paranoiac II	2520000	5370000	7510000	15400000
Round2-uround2-n1-fn1-l5	6589401	8665781	402090	15657272
LIMA-CCA-sp-2062	5114770	4738128	6237127	16090025
HQC Paranoiac III	2660000	5620000	8030000	16310000
HQC Paranoiac IV	2810000	5950000	8460000	17220000
LOCKER IX	10400000	1490000	6600000	18490000
Round2-uround2-n1-fn2-l5	9360000	10110000	340000	19810000
BIKE-1 5	2986647	3023816	17486906	23497369
BIKE-3 5	2300332	3257675	18047493	23605500
DING Key Exchange 1024	6813691	9541851	7617506	23973048

A sorting is created based on the intersection of the two previous rankings for space requirements and execution times (pk+c[B] and sum of cycles). The intersection is the calculated cycles per byte (cpb). Any submission implementations which are not present in both the top ranking implementations for space requirements and execution times have been removed.

The top 10 performers using this category is shown below in order of best to worst. The calculated cpbs for each implementation is shown in Table 5.24.

1. Lepton
2. Three Bears
3. NewHope
4. KCL
5. KINDI
6. SABER
7. CRYSTALS-KYBER
8. LAC
9. HILA5
10. Round2

Table 5.24: Calculated cpb for all level 5 KEM submissions' implementations which qualify as top 10 in both space requirements and execution times, using pk+c and the sum of cycles needed, sorted after the calculated cpb (cpb).

Submission Implementation	**pk+c[B]**	**exec.times[cycles]**	**cpb[cycles/B]**
Lepton.CPA Moderate IV	6041	282856	46.82
Lepton-CCA Moderate IV	6073	393443	64.79
Three Bears PapaBear Ephem	3281	319000	97.23
NewHope-CPA-1024	4000	504020	126.01
Three Bears PapaBear	3281	477000	145.38

KCL AKCN-RLWE	3779	816786	216.14
NewHope-CCA-1024	4032	1059092	262.67
KINDI-512-3-2-1	5696	1984603	348.42
KCL OKCN-RLWE	3691	1341149	363.36
SABER fire	2784	1233722	443.15
KINDI-256-5-2-2	3952	1866368	472.26
CRYSTALS-KYBER 1024	2944	1408346	478.38
LAC-CCA-256	3104	1669488	537.85
HILA5	3836	2386800	622.21
Round2-uround2-nd-l5	1577	1450000	919.47
Ouroboros-R-256	3256	6940000	2131.45
NTRU Prime sntrup4591761	2265	6157140	2718.38
LOCKER III	2620	7950000	3034.35
LOCKER VI	3012	9170000	3044.49
LAKE III	1652	5030000	3044.79
LOCKER IX	4524	18490000	4087.09
DING Key Exchange 1024	4240	23973048	5654.02

5.3.3 Signature

5.3.3.1 NIST Security Level 1 and 2

The top 10 signature submissions as sorted after the size of the sum of private keys, public keys, and ciphertexts are given below. Their best implementations according to the same requirements can be seen in Table 5.26.

1. WalnutDSA
2. FALCON
3. CRYSTALS-DILITHIUM
4. qTESLA
5. SPHINCS+
6. LUOV
7. MQDSS
8. Picnic
9. Gravity-SPHINCS
10. HiMQ

Table 5.26: Space requirements for the top 10 NIST level 1 and 2 signature submissions' implementations sorted after the size of the sum of private keys, public keys, and ciphertexts (Sum).

Submission Implementation	$\mathbf{k_{private}}$**[B]**	$\mathbf{k_{public}}$**[B]**	**sig[B]**	**Sum**
WalnutDSA BKL-128	136	83	1100	1319
WalnutDSA STOC-128	136	83	1200	1419
WalnutDSA STOC-wo-DEH-128	291	128	2000	2419
FALCON 512	4097	897	690	5684
CRYSTALS-DILITHIUM medium	2800	1184	2044	6028
qTESLA-128	1856	2976	2720	7552
CRYSTALS-DILITHIUM high	3504	1472	2701	7677
SPHINCS+ haraka-128s	64	32	8080	8176
SPHINCS+ SHA256-128s	64	32	8080	8176
SPHINCS+ shake256-128s	64	32	8080	8176
FALCON 768	6145	1441	1077	8663
LUOV-49-49-242	32	7300	1700	9032
LUOV-8-63-256	32	15500	319	15851
SPHINCS+ haraka-128f	64	32	16976	17072
SPHINCS+ SHA256-128f	64	32	16976	17072
SPHINCS+ shake256-128f	64	32	16976	17072
MQDSS-48	32	62	32882	32976
Picnic-L1-FS	16	32	34000	34048
Picnic-L1-UR	16	32	53929	53977
Gravity-SPHINCS S	65568	32	12540	78140
HiMQ-3P	32	100878	67	100977
HiMQ-3F	14878	100878	67	115823
HiMQ-3	12074	128744	75	140893

The top 10 signature submissions as sorted after the sum of execution times (number of needed cycles) are given below. Their best implementations according to the same requirements can be seen in Table 5.28.

1. CRYSTALS-DILITHIUM
2. Gui
3. qTESLA
4. pqsigRM
5. WalnutDSA
6. LUOV
7. HiMQ
8. MQDSS
9. FALCON
10. Picnic

Table 5.28: Execution times for the top 10 NIST level 1 and 2 signature submissions' implementations given in number of needed cycles, sorted after the sum of all execution times (Sum).

Submission Implementation	Key.Gen	Sign	Verify	Sum
CRYSTALS-DILITHIUM medium	269844	1285476	296920	1852240
CRYSTALS-DILITHIUM high	382756	1817902	395936	2596594
Gui-184	2408000	10910000	152000	13470000
qTESLA-128	3402000	5870005	12433000	21705005
pqsigRM-4-12	9641836	15194705	81178	24917719
LUOV-8-63-256	21000000	5870000	4930000	31800000
HiMQ-3	50593934	21594	17960	50633488
WalnutDSA STOC-wo-DEH-128	2574824	48246052	147948	50968824
WalnutDSA STOC-128	2271199	51244842	101529	53617570
LUOV-49-49-242	14800000	34100000	23600000	72500000
HiMQ-3F	79256175	25613	14645	79296433
MQDSS-48	1206730	52466398	38686506	92359634
FALCON 768	91009209	8359971	666108	100035288
WalnutDSA BKL-128	2086564	137691863	96962	139875389
Picnic-L1-FS	163850	131390415	86062091	217616356
Picnic-L1-UR	146193	158826399	106128443	265101035

A sorting based on the intersection of the two previous rankings for space requirements and execution times (sum of space requirements and sum of cycles). The intersection is the calculated cycles per byte (cpb). Any submission implementations which are not present in both the top ranking implementations for space requirements and execution times have been removed.

The top 5 performers using this category is shown below in order of best to worst. The calculated cpbs for each implementation is shown in Table 5.30.

1. pqsigRM
2. Gui

3. CRYSTALS-KYBER
4. HiMQ
5. LUOV

Table 5.30: Calculated cpb for all level 1 and 2 signature submissions' implementations which qualify as top 10 in both space requirements and execution times, using pk+c and the sum of cycles needed, sorted after the calculated cpb (cpb).

Submission Implementation	**Size reqs.[B]**	**exec.times[cycles]**	**cpb[cycles/B]**
pqsigRM-4-12	2055986	24917719	12.12
Gui-184	435445	13470000	30.93
CRYSTALS-DILITHIUM medium	6028	1852240	307.27
CRYSTALS-DILITHIUM high	7677	2596594	338.23
HiMQ-3	140893	50633488	359.38
HiMQ-3F	115823	79296433	684.63
LUOV-8-63-256	15851	31800000	2006.18
MQDSS-48	32976	92359634	2800.81
qTESLA-128	7552	21705005	2874.07
Picnic-L1-UR	53977	265101035	4911.37
Picnic-L1-FS	34048	217616356	6391.46
LUOV-49-49-242	9032	72500000	8027.02
FALCON 768	8663	100035288	11547.42
WalnutDSA STOC-wo-DEH-128	2419	50968824	21070.20
WalnutDSA STOC-128	1419	53617570	37785.46
WalnutDSA BKL-128	1319	139875389	106046.54

5.3.3.2 NIST Security Level 5

The top 10 signature submissions as sorted after the size of the sum of private keys, public keys, and ciphertexts are given below. Their best implementations according to the same requirements can be seen in Table 5.32.

1. WalnutDSA
2. pqNTRUsign
3. FALCON
4. qTESLA
5. SPHINCS+
6. LUOV
7. Picnic
8. Rainbow
9. GeMSS
10. pqsigRM

Table 5.32: Space requirements for the top 10 NIST level 5 signature submissions' implementations sorted after the size of the sum of private keys, public keys, and ciphertexts (Sum).

Submission Implementation	$k_{private}$[B]	k_{public}[B]	sign[B]	Sum
WalnutDSA BKL-256	291	128	1800	2219
WalnutDSA STOC-256	291	128	2100	2519
WalnutDSA STOC-wo-DEH-256	136	83	3400	3619
pqNTRUsign Gaussian-1024	2503	2065	2065	6633
pqNTRUsign Uniform-1024	2604	2065	2065	6734
FALCON 1024	8193	1793	1330	11316
qTESLA-256	4128	6432	5920	16480
SPHINCS+ haraka-256s	128	64	29792	29984
SPHINCS+ SHA256-256s	128	64	29792	29984
SPHINCS+ shake256-256s	128	64	29792	29984
LUOV-80-86-399	32	39300	4700	44032
SPHINCS+ haraka-256f	128	64	49216	49408
SPHINCS+ SHA256-256f	128	64	49216	49408
SPHINCS+ shake256-256f	128	64	49216	49408
LUOV-8-117-404	32	98600	521	99153
Picnic-L5-FS	32	64	132824	132920
Picnic-L5-UR	32	64	209474	209570
Rainbow VIa	892079	1351361	118	2243558
Rainbow VIb	1016868	1456225	147	2473240
Rainbow Vc	1274317	1723681	204	2998202
GeMSS 256	82056	3603792	104	3685952
Gui-448	155900	5789200	83	5945183
pqsigRM-6-13	2144166	2105344	2106372	6355882

The top 10 signature submissions as sorted after the sum of execution times (number of needed cycles) are given below. Their best implementations according to the same requirements can be seen in Table 5.34.

1. qTESLA
2. Gui
3. pqsigRM
4. WalnutDSA
5. FALCON
6. LUOV
7. pqNTRUsign
8. SPHINCS+
9. GeMSS
10. Rainbow

Table 5.34: Execution times for the top 10 NIST level 5 signature submissions' implementations given in number of needed cycles, sorted after the sum of all execution times (Sum).

Submission Implementation	Key.Gen	Sign	Verify	Sum
qTESLA-256	520000	1065000	1310000	2895000
Gui-448	239502	872949	1787000	2899451
pqsigRM-6-13	22668519	1557210	540378	24766107
WalnutDSA STOC-wo-DEH-256	4836298	130993063	311116	136140477
WalnutDSA STOC-256	4519863	134509781	194916	139224560
FALCON 1024	157623028	13058641	1117624	171799293
LUOV-8-117-404	146000000	36500000	29700000	212200000
LUOV-80-86-399	96800000	216000000	124000000	436800000
pqNTRUsign Uniform-1024	268329761	202185303	2726230	473241294
WalnutDSA BKL-256	4456087	472468875	197243	477122205
pqNTRUsign Gaussian-1024	259672814	349028118	2955494	611656426
SPHINCS+ SHA256-256f	68819608	1558148364	38316192	1665284164
SPHINCS+ shake256-256f	75031996	1664510764	41469276	1781012036
Picnic-L5-UR	740633	1187481996	797249015	1985471644
SPHINCS+ haraka-256f	113876252	3172247452	76203004	3362326708
GeMSS 256	1245472262	5522622728	2202925	6770297915
SPHINCS+ SHA256-256s	1095050628	12893347756	19141296	14007539680
SPHINCS+ shake256-256s	1210939356	13842403104	20889204	15074231664
SPHINCS+ haraka-256s	1817324180	28860355888	42380420	30720060488
Rainbow VIa	45064000000	3916000	2897000	45070813000
Picnic-L5-FS	722494	1073183185	70865264744	71939170423
Rainbow Vc	116046000000	8688000	6174000	116060862000
Rainbow VIb	164689000000	16755000	11224000	164716979000

A sorting based on the intersection of the two previous rankings for space requirements and execution times (sum of space requirements and sum of cycles). The intersection is the calculated cycles per byte (cpb). Any submission implementations which are not present in both the top ranking implementations for space requirements and execution times have been removed.

The top 5 performers using this category is shown below in order of best to worst. The calculated cpbs for each implementation is shown in Table 5.36.

1. Gui
2. pqsigRM
3. GeMSS
4. LUOV
5. Picnic

Table 5.36: Calculated cpb for all level 5 signature submissions' implementations which qualify as top 10 in both space requirements and execution times, using pk+c and the sum of cycles needed, sorted after the calculated cpb (cpb).

Submission Implementation	**Size reqs.[B]**	**exec.times[cycles]**	**cpb[cycles/B]**
Gui-448	5945183	2899451	0.49
pqsigRM-6-13	6355882	24766107	3.90
qTESLA-256	16480	2895000	175.67
GeMSS 256	3685952	6770297915	1836.78
LUOV-8-117-404	99153	212200000	2140.13
Picnic-L5-UR	209570	1985471644	9474.03
LUOV-80-86-399	44032	436800000	9920.06
FALCON 1024	11316	171799293	15181.98
Rainbow VIa	2243558	45070813000	20088.99
SPHINCS+ SHA256-256f	49408	1665284164	33704.75
SPHINCS+ shake256-256f	49408	1781012036	36047.04
WalnutDSA STOC-wo-DEH-256	3619	136140477	37618.26
Rainbow Vc	2998202	116060862000	38710.15
WalnutDSA STOC-256	2519	139224560	55269.77
Rainbow VIb	2473240	164716979000	66599.67
SPHINCS+ haraka-256f	49408	3362326708	68052.27
pqNTRUsign Uniform-1024	6734	473241294	70276.40
pqNTRUsign Gaussian-1024	6633	611656426	92214.15
WalnutDSA BKL-256	2219	477122205	215016.77
SPHINCS+ SHA256-256s	29984	14007539680	467167.15
SPHINCS+ shake256-256s	29984	15074231664	502742.52
Picnic-L5-FS	132920	71939170423	541221.57
SPHINCS+ haraka-256s	29984	30720060488	1024548.44

Chapter 6

Conclusion

In this Chapter, the research questions will be briefly answered. A brief synopsis of the master's thesis will also be provided, as well as an overview of the road ahead.

6.1 Research Question Answers

6.1.1 Question 1

What is the motivation for development of quantum resistant cryptography?

The motivation behind this development is the development of quantum computers combined with the potential of Shor's algorithm. The hard problems which can be solved efficiently by this combination are dominant in today's cryptographic environment, and solving them means the end for many of today's most used algorithms, revealing not only today's information, but also encrypted information from years back.

6.1.2 Question 2

What are the current approaches to creating quantum resistant cryptographic algorithms?

All of the different categories of quantum-resistant algorithms which are most prevalent today are mentioned in section 2.6, and are explained both generally, historically, and mathematically.

6.1.3 Question 3

What cryptographic algorithms have been developed and submitted to NIST as quantum resistant, and how do they compare?

All proposed submissions for the NIST PQC Standardisation are given in Chapter 3, and are systematically compared in Chapter 4. All discussion around them and the comparison of these are given in Chapter 5.

6.2 Concluding Remarks

This master's thesis was written with several uses in mind. Firstly, it was intended to be an introduction into the subject of post-quantum cryptography, and its development as a whole, as well as NIST's PQC standardisation process. Chapters 2 is where this is found. Secondly, it was to be an overview and comparative analysis of the submissions for NIST's Post-Quantum Cryptography Standardisation process, their types, submitters, goals, properties, and performance. This is found mainly in Chapter 3 and 4. Perhaps most importantly, it was also intended to be a motivator towards looking into post-quantum cryptography development, by explaining not only why, but also how this idea went from futuristic to extremely relevant in today's technological world.

As previously mentioned, the development of quantum computers is still in its infancy, and it is highly unlikely that full-scale, stable quantum computers will be developed immediately. This does in no way mean that the development of quantum-resistant cryptography can rest on its laurels. This is not only due to the reasons previously stated in this Chapter, but also due to Mosca's theorem, as presented in Chapter 1.

It is also quite evident that the world needs to address the development of quantum computers, and with it, the possible cracking of many of today's most prevalent cryptographic algorithms. While there are no other known, public efforts to develop and standardise quantum-resistant cryptography apart from the NIST Standardisation Process as of today, it is likely that there are many groups which work towards the same goal, even if none of them are doing it using the public and the scientific community within the world of cryptography. NSA announced their plans for transitioning over to quantum resistant algorithms earlier this year, which is a strong indication of the direction of the urgency felt by organisations where a high level of security is paramount [187]. Thus, it is imperative that this development continues, to ensure that data which is to be secure, remains this way.

Appendices

Appendix A

Scripts

A.1 Attack on Odd Manhatten

```
// Run the attack as follows:
// $ gcc -Ofast -DNDEBUG -lcrypto -lgmp attack.c rng.c kem.c -o attack
// $ ./attack
//

#include <stdio.h>
#include <string.h>
#include "api.h"
#include "assert.h"
#include "gmp.h"
#include "rng.h"

/// global variables
unsigned char pk[CRYPTO_PUBLICKEYBYTES], sk[CRYPTO_SECRETKEYBYTES];
unsigned char ss0[CRYPTO_BYTES];
unsigned char ss1[CRYPTO_BYTES];

/// CCA oracle
int oracle_dec(unsigned char* ct) {
  unsigned char ss[CRYPTO_BYTES];

  int ret = crypto_kem_dec(ss, ct, sk);

  // we should have a CCA failure, but we ignore the return code :)
  assert(ret == -1);
  // we should have ss == ss0 or ss == ss1
  assert(memcmp(ss, ss0, CRYPTO_BYTES) == 0 ||
         memcmp(ss, ss1, CRYPTO_BYTES) == 0);

  // return b where ss == ssb
  return (memcmp(ss, ss1, CRYPTO_BYTES) == 0);
}

/// Decrypt with guess (from kem.c)
int decrypt_with_guess(mpz_t ciphertext, mpz_t quotient, const mpz_t guess,
                       const mpz_t det) {
  int r0 = 0;
  mpz_mul(ciphertext, ciphertext, guess);
```

```
  mpz_mod(ciphertext, ciphertext, det);

  // Extract m
  mpz_add_ui(quotient, ciphertext, C / 2);
  if (mpz_sizeinbase(quotient, 2) >= N)
    r0 += (char)(mpz_odd_p(ciphertext) == 0);
  else
    r0 += (char)(mpz_even_p(ciphertext) == 0);
  return r0;
}

int main() {
  /// Initialize randomness (attack should work for any value)
  unsigned char entropy_input[48];
  for (int i = 0; i < 48; i++) entropy_input[i] = i;
  randombytes_init(entropy_input, NULL, 256);

  /// Get shared keys corresponding to
  /// two target seeds: seed = 00..00 and seed = ff...ff00...00
  unsigned char seed[32];

  AES_XOF_struct ctx[1];
  unsigned char diversifier[8] = {0};
  unsigned long maxlen = 4294967295;

  memset(seed, 0, 32);
  seedexpander_init(ctx, seed, diversifier, maxlen);
  memset(ss0, 0, CRYPTO_BYTES);
  seedexpander(ctx, ss0, CRYPTO_BYTES);

  memset(seed, 255, 16);
  memset(seed + 16, 0, 16);
  seedexpander_init(ctx, seed, diversifier, maxlen);
  memset(ss1, 0, CRYPTO_BYTES);
  seedexpander(ctx, ss1, CRYPTO_BYTES);

  /// Generate key pair
  crypto_kem_keypair(pk, sk);

  /// Compute determinant
  mpz_t det;
  mpz_init(det);
  mpz_ui_pow_ui(det, 2, N);
  mpz_sub_ui(det, det, C);

  /// Attack!
  mpz_t guess, ciphertext, quotient;
  mpz_inits(guess, ciphertext, quotient, NULL);
  unsigned char ct[CRYPTO_CIPHERTEXTBYTES];
  unsigned char expected = 0;
  for (int i = 0; i < P; i++) {
    printf("%d/%d\n", i + 1, P);
    for (int j = 0; j < 8; j++) {
      if (8 * i + j >= N) break; // we should have everything
      if (8 * i + j == 0) continue; // the attack starts at 1

      // set all ciphertexts to 2^(8i+j)
      mpz_set_ui(ciphertext, 0);
```

```c
      mpz_setbit(ciphertext, i * 8 + j);

      // transform mpz_t into array of bytes
      memset(ct, 0, CRYPTO_CIPHERTEXTBYTES);
      for (int k = 0; k < CRYPTO_CIPHERTEXTBYTES / P; k++)
        mpz_export(&(ct[k * P]), NULL, -1, 1, -1, 0, ciphertext);

      // call oracle
      int b = oracle_dec(ct);

      if (b != expected) {
        // update our guess
        mpz_setbit(guess, N - 1 - (i * 8 + j));
      }

      /// update the "expected" value
      mpz_clrbit(ciphertext, i * 8 + j);
      mpz_setbit(ciphertext, i * 8 + j + 1);
      expected = decrypt_with_guess(ciphertext, quotient, guess, det);
    }
  }

  /// Transform mpz_t into array of bytes
  unsigned char guessed_sk[CRYPTO_SECRETKEYBYTES];
  mpz_export(&guessed_sk, NULL, -1, 1, -1, 0, guess);

  /// Success
  if (memcmp(guessed_sk, sk, P) == 0) {
    printf(
        "Success! The attack recovered the P first bytes of sk (which are the "
        "only ones used in crypto_kem_dec.\n");
  } else {
    printf("Failure.\n");
    gmp_printf("guess = %Zd\n", guess);
    mpz_t secret_key;
    mpz_init(secret_key);
    mpz_import(secret_key, P, -1, 1, -1, 0, sk);
    gmp_printf("sk=%Zd\n", secret_key);
    gmp_printf("det=%Zd\n", det);
    mpz_clear(secret_key);
  }

  // OCD
  mpz_clears(ciphertext, quotient, guess, det, NULL);

  return 0;
}
```

A.2 Attack on SRTPI

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include "api.h"
#include "rng.h"
long long nanoseconds(void)
{
  struct timespec t;
  clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&t);
  return (long long) t.tv_nsec + 1000000000 * (long long) t.tv_sec;
}
int bit(unsigned char *c,int pos)
{
  return 1 & (c[pos / 8] >> (pos & 7));
}
void xor(unsigned char *c1,const unsigned char *c2)
{
  int i;
  for (i = 0;i < 512;++i) c1[i] ^= c2[i];
}
void swap(unsigned char *c1,
  if (crypto_encrypt_keypair(pk,sk) != 0) abort();
  t1 = nanoseconds();
  printf("%lld ns for alice creating key pair (crypto_encrypt_keypair)\n",t1 - t0);
  t0 = nanoseconds();
  if (crypto_encrypt(czero,&clen,mzero,MLEN,pk) != 0) abort();
  for (i = 0;i < RANDOMCOVER;++i) {
    if (crypto_encrypt(cdiff[i],&clen,mzero,MLEN,pk) != 0) abort();
    xor(cdiff[i],czero);
  }
  i = 0;
  for (j = 0;j < 4096;++j)
    /* have reduced positions 0...j-1 using cdiff[0...i-1] */
    for (k = i;k < RANDOMCOVER;++k)
      if (bit(cdiff[k],j)) {
        swap(cdiff[i],cdiff[k]);
        for (l = 0;l < RANDOMCOVER;++l)
          if (l != i)
            if (bit(cdiff[l],j))
              xor(cdiff[l],cdiff[i]);
        cdiffpivot[i++] = j;
        break;
      }
  cdiffpivotlen = i;
  for (i = 0;i < MLEN * 8;++i) {
    for (j = 0;j < 512;++j) mbits[i][j] = 0;
    mbits[i][i / 8] = 1 << (i & 7);
    if (crypto_encrypt(cbits[i],&clen,mbits[i],MLEN,pk) != 0) abort();
    xor(cbits[i],czero);
    /* now project away from the subspace of randomness */
    /* alternative: force randombits to return 0 at this point */
    for (k = 0;k < cdiffpivotlen;++k)
      if (bit(cbits[i],cdiffpivot[k]))
```

```
      xor(cbits[i],cdiff[k]);
  }
  i = 0;
  for (j = 0;j < 4096;++j)
    for (k = i;k < MLEN * 8;++k)
      if (bit(cbits[k],j)) {
        swap(mbits[i],mbits[k]);
        swap(cbits[i],cbits[k]);
        for (l = 0;l < MLEN * 8;++l)
          if (l != i)
            if (bit(cbits[l],j)) {
              xor(mbits[l],mbits[i]);
              xor(cbits[l],cbits[i]);
            }
        cbitspivot[i++] = j;
        break;
      }
  cbitspivotlen = i;
  t1 = nanoseconds();
  printf("%lld ns for eve analyzing public key (one time, independent of
   ↪ ciphertext)\n",t1 - t0);
  t0 = nanoseconds();
  randombytes(m,MLEN);
  if (crypto_encrypt(c,&clen,m,MLEN,pk) != 0) abort();
  t1 = nanoseconds();
  printf("%lld ns for bob creating ciphertext (crypto_encrypt)\n",t1 - t0);
  t0 = nanoseconds();
  if (crypto_encrypt_open(t,&tlen,c,clen,sk) != 0) abort();
  if (tlen != MLEN) abort();
  t1 = nanoseconds();
  printf("%lld ns for alice decrypting (crypto_encrypt_open)\n",t1 - t0);
  t0 = nanoseconds();
  xor(c,czero);
  for (k = 0;k < cdiffpivotlen;++k)
    if (bit(c,cdiffpivot[k]))
      xor(c,cdiff[k]);
  for (j = 0;j < 512;++j) e[j] = 0;
  for (k = 0;k < cbitspivotlen;++k)
    if (bit(c,cbitspivot[k])) {
      xor(c,cbits[k]);
      xor(e,mbits[k]);
    }
  /* can speed up attack by composing these two matrix steps */
  /* but eve is already more than ten times faster than alice */
  t1 = nanoseconds();
  printf("%lld ns for eve analyzing ciphertext\n",t1 - t0);
  t0 = nanoseconds();
  printf("eve's plaintext   ");
  for (j = 0;j < MLEN;++j) printf("%02x",e[j]); printf("\n");
  printf("bob's plaintext   ");
  for (j = 0;j < MLEN;++j) printf("%02x",m[j]); printf("\n");
  printf("alice's plaintext ");
  for (j = 0;j < MLEN;++j) printf("%02x",t[j]); printf("\n");
  return 0;
```

A.3 Attack on CFPKM by Ron Steinfeld

```c
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "api.h"
#include "KEMheader.h"
#include "rng.h"
#include "randombytes.h"

void allocatemem(Pol *f, int n,int m){
        int i;
        for(i =0;i<m;i++)
        {
                f[i].QD = malloc((n*n) * sizeof(long));            /* allocating memory for the coefficients of each polynomial to be stored in*/
                f[i].L = malloc(n * sizeof(long));
        }
}

void freealloc(Pol *f, int m)
{
        int i;
        for(i=0;i<m;i++)                /* freeing the allocated memory*/
        {
                free(f[i].QD);
                free(f[i].L);
        }
}

void polgen(Pol *f, int m, int n )
{                                                                      /* generates a system of m polynomials over n variables */
        int i,l;
        long *out=malloc(sizeof( long));
        for(i=0; i< m; i++)
        {
                long cofval[(N*(N+1)/2) + N+1];
                for (l=0;l< ((N*(N+1)/2) + N+1);l++)
                {
                        randombytes((unsigned char*)(&out), 4);
                        cofval[l]=((long)out)%(COFSIZE);
                }
                int j,k,count=0;
                for(j=0; j<n; j++)
                {
                        for(k=0; k<n; k++)
                        {
                                if(k > j)
                                        f[i].QD[(k*n+j)] = 0;
                                else
                                        {
                                        f[i].QD[(k*n+j)] = cofval[count]%(COFSIZE);
                                        count++;
                                        }
```

```
                        }
                }
                for(j=0; j<n; j++)
                {
                        f[i].L[j] = cofval[count]%(COFSIZE) ;
                        count++;
                }
                f[i].C = cofval[count]%(COFSIZE) ;
        }
}

unsigned long long evaluate_poly(Pol unPoly,  unsigned char *pValue, int n)
{
        int i, j;
        unsigned long long result1 = 0, result2 = 0;                          /* evaluates f over a value, like f(sa)*/
        unsigned long long tabResult1[n];
/*for quad */
        for(j=0; j<n; j++)
        {
                tabResult1[j] =0;
                for(i=0; i<n; i++)
                {
                        tabResult1[j] = tabResult1[j]+ ((unsigned long)pValue[i] * unPoly.QD[i*n + j]) ;
                }
                result1 = (result1 + tabResult1[j] * (unsigned long)pValue[j]) ;
        }
/*for linear*/
        for(i=0; i<n; i++)
        {
                result2 = (result2 + unPoly.L[i] * (unsigned long)pValue[i]) ;
        }
        result1 = (result1 + result2 + unPoly.C);
        return result1;
}

void Eval_sys(Pol *pSyst, unsigned char* pValue, int m, int n,unsigned long long *result)
{
        int i;  /*evaluates a system of polynomials over a provided value, calls the evaluate_poly function for each polynomial*/
        for(i=0; i<M; i++)
                result[i] = evaluate_poly(pSyst[i], pValue, N);
}

unsigned char kem_crossround1( unsigned long long in){
        unsigned char out;
        unsigned long long rem = in >> (B_BAR-1);                   /*CrossRound function to give the CrossRound bit of a value*/
        out =(unsigned char) (rem%2);
        return out;
}

unsigned char rounding(unsigned long long in)
{
        unsigned char out;
```

```c
        unsigned long long rem =( in + (2^(B_BAR-1)));              /*Rounding function to give the rounded value*/
        unsigned long long rem2 = (rem % Q);
        out = (unsigned char)((rem2 >> B_BAR));
        return out;
}

void kem_crossround2(unsigned char *out,  unsigned  long long *in) {
        int i;                                                      /*CrossRound function over a vector*/
        for (i = 0; i < M; i++) {
                unsigned long long rem = in[i] >> (B_BAR-1);
                out[i] = (unsigned char)(rem%2);
        }
}

void kem_rounding(unsigned char *out,  unsigned long long *in) {
        int i;
        for (i=0; i < M; i++){                    /*Rounding function over a vector*/
                unsigned long long rem = (in[i] + (2^(B_BAR-1)));
                unsigned long long rem2 = (rem % Q);
                out[i] = (unsigned char)((rem2 >> B_BAR));
        }
}

void kem_rec(unsigned char *key,  unsigned long long *w, unsigned char *c){
        int i;
        unsigned long long w1,w2;
        unsigned char hint;
        for (i =0; i < M;i++){ function from the article*/
                int flag=0;
                hint= kem_crossround1(w[i]);
                if (hint==c[i])
                {
                        key[i] = rounding(w[i]);
                        flag=1;
                }
                if (flag==0)
                {
                        w1 = (w[i] + (2^(B_BAR-2))-1) ;
                        hint= kem_crossround1(w1);
                        if (hint==c[i]){
                                key[i] = rounding(w1);
                        }
                        else{
                                w2 =(w[i] - (2^(B_BAR-2))+1) ;
                                hint= kem_crossround1(w2);
                                if (hint==c[i]){
                                        key[i] = rounding(w2);
                                }
                                else key[i]=0;
                        }
                }
        }
}
```

```
void pack_sk(unsigned char *sk, unsigned char *sa, unsigned char *seed){
        int
         ↪ i;
         ↪ makes SK=(seed||sa)*/
        for(i=0;i<  SEEDSIZE;i++)
                {sk[i]=seed[i];}
        for(i=0;i < N;i++)
                sk[SEEDSIZE+i]=sa[i];
}

void unpack_sk(unsigned char *sa, unsigned char *seed, const unsigned char *sk){
        int i;
        for(i=0;i<
         ↪ SEEDSIZE;i++)
         ↪ SK to give out seed and sa*/
                {seed[i]=sk[i];}
        for(i=0;i < N;i++)
                sa[i]=sk[SEEDSIZE+i];
}

void pack_pk(unsigned char *pk,unsigned  long long *b1, unsigned char *seed){
        int i,j;
        for(i=0 ;i <SEEDSIZE;i++)
                {pk[i]=seed[i];}
        unsigned char temp;
        unsigned char
         ↪ mask=255;
         ↪ makes PK=(seed||b1)*/
        for(i =0;i<M;i++)
                {for(j=7;j>-1;j--)
                        {temp=(b1[i] & mask);
                        b1[i]=b1[i]>>8;
                        pk[SEEDSIZE+i*8+j]=temp;
                }
        }
}

void unpack_pk(unsigned long long *b1, unsigned char *seed, const unsigned char *pk){
        int i,j;
        for(i=0;i<SEEDSIZE;i++)
                seed[i]=pk[i];
        unsigned char temp;
        for(i=0;i<M;i++)
                b1[i]=0;
        for(i=0;i<M;i++)
                {
                 ↪ PK to give out seed and the public vector b1*/
                        for(j=0;j<7;j++)
                                {
                                temp = pk[i*8+j+SEEDSIZE];
                                b1[i]=b1[i]+temp;
                                b1[i]=b1[i]<<8;
                                }
                        b1[i]=b1[i]+pk[i*8+7+SEEDSIZE];
                }
}

void pack_ct(unsigned char *ct, unsigned long long *b2,unsigned char *c){
```

```
        int  i,j;
        for (i=0;i < M;i++)
                ct[i]=c[i];
                 ↪  ct=(c||b2)*/
        unsigned char temp;
        unsigned char mask=255;
        for(i =0;i<M;i++)
                {for(j=7;j>-1;j--)
                        {temp=(unsigned char)(b2[i] & mask);
                        b2[i]=b2[i]>>8;
                        ct[M+i*8+j]=temp;
                }
        }

}
void unpack_ct(unsigned long long *b2,unsigned  char *c, const unsigned char *ct){
        int  i,j;
        for (i=0;i < M;i++)
                c[i]=ct[i];
        unsigned char temp;
        for(i=0;i<M;i++)
         ↪  ct to give out the hint vector c and b2*/
                b2[i]=0;
        for(i=0;i<M;i++)
                {
                        for(j=0;j<7;j++)
                                {
                                temp = ct[i*8+j+M];
                                b2[i]=b2[i]+temp;
                                b2[i]=b2[i]<<8;
                                }
                        b2[i]=b2[i]+ct[i*8+7+M];
                }
}

int crypto_kem_keypair(unsigned char *pk, unsigned char *sk){
        unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));if (seed==NULL)
         ↪  {printf("EXIT");return 0;}
        randombytes(seed,SEEDSIZE);
        Pol *f1 = malloc(M * sizeof(Pol));
        allocatemem(f1,N,M);
        randombytes_init(seed,NULL,256);
        polgen(f1,M,N);
        int i;
        unsigned char *sa=malloc(N*sizeof(unsigned char));if (sa==NULL)
         ↪  {printf("EXIT");return 0;}
        randombytes(sa,N*SECRETVAL_LENGTH);
        unsigned char *e1=malloc(M*sizeof(unsigned char));if (e1==NULL)
         ↪  {printf("EXIT");return 0;}
        randombytes(e1,M*ERROR_LENGTH);
        for(i=0;i < N;i++)
                sa[i]=(unsigned char)((sa[i])%RANGE);
        for(i=0;i < M;i++)
                {e1[i]=(unsigned char)((e1[i])%RANGE);        }
        unsigned long long *b1=malloc(M*sizeof(unsigned long long));if (b1==NULL)
         ↪  {printf("EXIT");return 0;}
        Eval_sys(f1,sa,M,N,b1);
        for (i =0;i <M ;i++)
```

```
        {
                b1[i] = (b1[i] + e1[i]) ;
        }
        pack_sk(sk,sa,seed);
        pack_pk(pk,b1,seed);
        return 0;
}

int crypto_kem_enc(unsigned char *ct, unsigned char *ss, const unsigned char *pk){
        int i;
        unsigned long long *b1=malloc(M*sizeof(unsigned long long));
        unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));
        unpack_pk(b1, seed, pk);
        Pol *f2 = malloc(M*sizeof(Pol));
        allocatemem(f2,N,M);
        randombytes_init(seed,NULL,256);
        polgen(f2,M,N);
        unsigned char *seed1=malloc(SEEDSIZE*sizeof(unsigned char));
        randombytes(seed1,SEEDSIZE);
        randombytes_init(seed1,NULL,256);
        unsigned char *sb=malloc(N*sizeof(unsigned char));
        unsigned char *e2=malloc(M*sizeof(unsigned char));if (e2==NULL) {printf("EXIT");return 0;}
        unsigned char *e3=malloc(M*sizeof(unsigned char));if (e3==NULL) {printf("EXIT");return 0;}
        randombytes(sb, N*SECRETVAL_LENGTH);
        randombytes(e2,M*ERROR_LENGTH);
        randombytes(e3,M*ERROR_LENGTH);
        for(i=0;i < N;i++)
                {sb[i]=(unsigned char)((sb[i])%RANGE);}
        for(i=0;i < M;i++)
                {e2[i]=(unsigned char)((e2[i])%RANGE);
                e3[i]=(unsigned char)((e3[i])%RANGE);        }

        unsigned long long *b2=malloc(M*sizeof(unsigned long long));if (b2==NULL) {printf("EXIT");return 0;}
        unsigned long long *b3=malloc(M*sizeof(unsigned long long));if (b3==NULL) {printf("EXIT");return 0;}
        Eval_sys(f2,sb,M,N,b2);
        for  (i =0;i<M;i++){
                b3[i] = (b2[i]*b1[i] + e3[i]);
                b2[i] = (b2[i] + e2[i]);

        }
        kem_rounding(ss, b3);
        unsigned char *c=malloc(M*sizeof(unsigned char));
        kem_crossround2(c, b3);
        pack_ct(ct, b2, c);
        return 0;
}

int crypto_kem_dec(unsigned char *ss, const unsigned char *ct, const unsigned char *sk){
        int i;
        unsigned char *sa=malloc(N*sizeof(unsigned char));
        unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));
        unpack_sk(sa,seed,sk);
        unsigned long long *b2=malloc(M*sizeof(unsigned long long));
```

```
        unsigned char *c=malloc(M*sizeof(unsigned char));
        unpack_ct(b2,c,ct);
        Pol *f = (Pol*)malloc(M*sizeof(Pol));
        allocatemem(f,N,M);
        randombytes_init(seed,NULL,256);
        polgen(f,M,N);
        unsigned long long *w = malloc(M*sizeof(unsigned long long));
        Eval_sys(f,sa,M,N,w);
        for (i=0;i < M;i++)
                {
                        w[i]=(w[i]*b2[i]) ;}
        kem_rec(ss, w, c);
        return 0;
}

int crypto_kem_atk_dec(unsigned char *ss, const unsigned char *ct, const unsigned
 ↪   char *pk){
        int i;
        unsigned long long *b1=malloc(M*sizeof(unsigned long long));
        unsigned char *seed=malloc(SEEDSIZE*sizeof(unsigned char));
        unpack_pk(b1, seed, pk);
        unsigned long long *b2=malloc(M*sizeof(unsigned long long));
        unsigned char *c=malloc(M*sizeof(unsigned char));
        unpack_ct(b2,c,ct);
        unsigned long long *w = malloc(M*sizeof(unsigned long long));
        for (i=0;i < M;i++)
                {
                        w[i]=(b1[i]*b2[i]) ;}
        kem_rounding(ss, w);
        return 0;
}
```

A.4 Attack on CFPKM by Martin R. Albrecht and Fernando Virdia

```python
# -*- coding: utf-8 -*-
"""
Shared secret recovery attack against CFPKM 128 KATs.
The script assumes the KATs to be in "./CFPKM/KAT/KEM/CFPKM128/PQCkemKAT_128.rsp".

The (un)pack_{pk,ct} functions are translated and adapted from the reference implementation.

AUTHOR:

    Martin R. Albrecht - 2017
    Fernando Virdia - 2017

"""

from sage.all import vector, IntegerModRing, ceil, log, floor, parent, ZZ, set_random_seed, randint


def openKAT(path):
    # utility function
    def ReadHex(buf):
        if len(buf) == 0:
            return ['\x00']
        else:
            res = []
            for x in range(len(buf)/2):
                res += [int("0x" + buf[2*x:2*x+2], 0)]
            return res

    l = []
    with open(path) as f:
        el = {}
        for line in f:
            if line in ["# CFPKM\n", "\n"]:
                continue
            if "count" in line:
                l += [el]
                el = { "count": line.split("=")[1].strip() }
            else:
                pre, fix = line.split("=")
                el[pre.strip()] = ReadHex(fix.strip())

        l += [el]
    return  l[1:]


def balance(e, q=None):
    try:
        p = parent(e).change_ring(ZZ)
        return p([balance(e_, q=q) for e_ in e])
    except (TypeError, AttributeError):
        if q is None:
            try:
```

```
                q = parent(e).order()
            except AttributeError:
                q = parent(e).base_ring().order()
        e = ZZ(e) % q
        return e-q if e>q//2 else e


def size_estimate(e):
    # check x != 0 to avoid ceil(-Infinity) that fails
    return vector(ZZ, len(e), [ceil(log(abs(x), 2)) if x != 0 else 0 for x in e])


def odot(a, b, q):
    return vector(IntegerModRing(q), len(a), [a[i] * b[i] for i in range(len(a))])


LAMBDA = 256
SEEDSIZE = 48
LOG2_Q = 50
N = 80
B = 6
M = 81
Q = 1125899906842624
COFSIZE = 4096
SECRETVAL_LENGTH = 1
SHAREDKEYSIZE = M * B / 8
ERROR_LENGTH = 1
PK_LENGTH = M * 8
RANGE = 7
B_BAR = LOG2_Q - B
CRYPTO_SECRETKEYBYTES = N + SEEDSIZE
CRYPTO_PUBLICKEYBYTES = PK_LENGTH + SEEDSIZE
CRYPTO_BYTES = M
CRYPTO_CIPHERTEXTBYTES = PK_LENGTH + M


def pack_pk (b1, seed):
    """
    :params:  b1, list(int)
    :params:  seed, list(int)

    :returns: pk, list(int)
    """
    b1 = b1[::]
    pk = [0] * CRYPTO_PUBLICKEYBYTES
    for i in range(SEEDSIZE):
        pk[i] = seed[i]
    mask = 255
    for i in range(M):
        for j in range(8)[::-1]:
            temp = b1[i] & mask
            b1[i] = b1[i] >> 8
            pk[SEEDSIZE+i*8+j] = temp
    return pk


def unpack_pk(pk):
    """
```

```
    :params:  pk, list(int)

    :returns: seed, list(int)
    :returns: b1, list(int)
    """
    seed = pk[:SEEDSIZE]
    b1 = [0] * M
    for i in range(M):
        # unpacks PK to give out seed and the public vector b1*/
        for j in range(7):
            temp = pk[i*8+j+SEEDSIZE]
            b1[i]=b1[i] + temp
            b1[i]=b1[i] << 8
        b1[i] = b1[i] + pk[i*8+7+SEEDSIZE]
    return seed, b1


def pack_ct(b2, c):
    """
    :params: b2, list(int)
    :params: c, list(int)

    :returns: ct, list(int)
    """
    b2 = b2[::]
    ct = [0] * CRYPTO_CIPHERTEXTBYTES
    for i in range(M):
        ct[i] = c[i]
    mask = 255

    for i in range(M):
        for j in range(8)[::-1]:
            temp = b2[i] & mask # this is casted to (unsigned char) in the ref implementation
            b2[i] = b2[i] >> 8
            ct[M+i*8+j] = temp
    return ct


def unpack_ct(ct):
    """
    :params:  ct, list(int)

    :returns: b2, list(int)
    :returns: c, list(int)
    """
    c = [0] * M
    b2 = [0] * M
    for i in range(M):
        c[i] = ct[i]

    for i in range(M):
        for j in range(7):
            temp = ct[i*8+j+M]
            b2[i] = b2[i] + temp
            b2[i] = b2[i] << 8
        b2[i] = b2[i] + ct[i*8+7+M]
    return (b2, c)
```

```python


def test_pack_unpack():
    kat = openKAT("CFPKM/KAT/KEM/CFPKM128/PQCkemKAT_128.rsp")

    ix = randint(0, len(kat)-1)
    pk = kat[ix]["pk"]
    ct = kat[ix]["ct"]

    # test pack/unpack pk
    print "Saved pk"
    print pk
    print
    seed1, b11 = unpack_pk(pk)
    pk2 = pack_pk(b11, seed1)
    print "Packed o Unpacked (pk) = pk"
    print pk2 == pk
    print

    seed2, b12 = unpack_pk(pk2)
    print "seeds match", seed1 == seed2
    print "b1 match", b11 == b12
    print

    # test pack/unpack ct
    print "Saved ct"
    print ct
    print
    b21, c1 = unpack_ct(ct)
    ct2 = pack_ct(b21, c1)
    print "Packed o Unpacked (ct) = ct",
    print ct2 == ct
    print

    b22, c2 = unpack_ct(ct2)
    print "b2 match", b21 == b22
    print "c match", c1 == c2


def attack():
    kat = openKAT("CFPKM/KAT/KEM/CFPKM128/PQCkemKAT_128.rsp")

    est = []
    for ix in range(len(kat)):
        pk = kat[ix]["pk"]
        ct = kat[ix]["ct"]
        ss = kat[ix]["ss"]

        seed, b1 = unpack_pk(pk)
        b2, c = unpack_ct(ct)

        b1 = vector(IntegerModRing(Q), b1)
        b2 = vector(IntegerModRing(Q), b2)
        ss = vector(IntegerModRing(Q), ss)

        # Print the bitlength of the difference between b1 odot b2 and the shared
        ↪ secret.
        est += [size_estimate(balance(odot(b1, b2, Q) - 2**B_BAR * ss, Q))]
```

```
        print est[ix]


```

Bibliography

[1] I. R. Center, "What is ibm q?" https://www.research.ibm.com/ibm-q/learn/what-is-ibm-q/, accessed: 2017-09-25.

[2] C. Vu, "Ibm makes quantum computing available on ibm cloud to accelerate innovation," https://www-03.ibm.com/press/us/en/pressrelease/49661.wss, accessed: 2017-09-25.

[3] J. Aron, "Revealed: Google's plan for quantum computer supremacy," https://www.newscientist.com/article/mg23130894-000-revealed-googles-plan-for-quantum-computer-supremacy/, accessed: 2017-09-25.

[4] M. Reynolds, "Google on track for quantum computer breakthrough by end of 2017," https://www.newscientist.com/article/2138373-google-on-track-for-quantum-computer-breakthrough-by-end-of-2017/, accessed: 2017-09-25.

[5] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," https://arxiv.org/pdf/1608.00263v3.pdf, accessed: 2017-09-25.

[6] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," https://arxiv.org/pdf/quant-ph/9508027.pdf, accessed: 2017-09-25.

[7] U. o. W. Dave Bacon, "Cse 599d - quantum computing shor's algorithm," https://courses.cs.washington.edu/courses/cse599d/06wi/lecturenotes11.pdf, accessed: 2017-09-20.

[8] M. Rouse, "Rsa algorithm (rivest-shamir-adleman)," http://searchsecurity.techtarget.com/definition/RSA, accessed: 2017-09-20.

[9] M. Mosca, "Cybersecurity in a quantum world: will we be ready?" https://csrc.nist.gov/csrc/media/events/workshop-on-cybersecurity-in-a-post-quantum-world/documents/presentations/session8-mosca-michele.pdf, accessed: 2018-05-21.

[10] NIST, "Post-quantum crypto project," http://csrc.nist.gov/groups/ST/post-quantum-crypto/, accessed: 2017-09-13.

[11] M. Rouse, "bit(binary digit)," http://whatis.techtarget.com/definition/bit-binary-digit, accessed: 2017-10-24.

[12] U. of Surrey, "Physical layer," http://www.ee.surrey.ac.uk/Projects/CAL/networks/Physical_Layer.htm, accessed: 2017-10-24.

[13] R. P. Feynman, "Simulating physics with computers."

[14] M. A.Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information.* Cambridge University Press.

[15] U. o. C. Anuj Dawar, "Quantum computing," http://www.cl.cam.ac.uk/teaching/0809/QuantComp/notes.pdf, accessed: 2017-10-24.

[16] J. S. Kvin Bonsor, "How quantum computers work," https://computer.howstuffworks.com/quantum-computer1.htm, accessed: 2017-10-30.

[17] W. Diffie and M. E. Hellman, "New directions in cryptography," https://ee.stanford.edu/~hellman/publications/24.pdf, accessed: 2017-10-05.

[18] L. A. R.L. Rivest, A. Shamir, "A method for obtaining digital signatures and public-key cryptosystems," http://people.csail.mit.edu/rivest/Rsapaper.pdf, accessed: 2017-09-25.

[19] V. Kapoor, V. S. Abraham, and R. Singh, "Elliptic curve cryptography," *Ubiquity*, vol. 2008, no. May, pp. 7:1–7:8, May 2008. [Online]. Available: http://doi.acm.org/10.1145/1378355.1378356

[20] E. Barker, D. Johnson, and M. Smid, "Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)," pp. 000–000.

[21] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[22] OpenSSL, "Elliptic curve diffie hellman," https://wiki.openssl.org/index.php/Elliptic_Curve_Diffie_Hellman, accessed: 2018-02-07.

[23] p. Peter Shor, "What is shor's factoring algorithm?" http://physicsworld.com/cws/article/multimedia/2015/sep/30/what-is-shors-factoring-algorithm, accessed: 2017-10-30.

[24] S. Aaronson, "Shor, i'll do it," https://www.scottaaronson.com/blog/?p=208, accessed: 2017-10-30.

[25] NIST, "Call for proposals," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals, accessed: 2018-02-19.

[26] ——, "Evaluation criteria," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria, accessed: 2017-10-31.

[27] ——, "Security(evaluation criteria)," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria/Security-(Evaluation-Criteria), accessed: 2017-10-31.

[28] ——, "Cost(evaluation criteria)," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria/Cost-(Evaluation-Criteria), accessed: 2017-10-31.

[29] ——, "Algorithm and implementation characteristics(evaluation criteria)," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Evaluation-Criteria/Algorithm-and-Implementation-Characteristics, accessed: 2017-10-31.

[30] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 99–108. [Online]. Available: http://doi.acm.org/10.1145/237814.237838

[31] J. Hoffstein, J. Pipher, and J. H. Silverman, "Ntru: A ring-based public key cryptosystem," in *Algorithmic Number Theory*, J. P. Buhler, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 267–288.

[32] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, ser. STOC '05. New York, NY, USA: ACM, 2005, pp. 84–93. [Online]. Available: http://doi.acm.org/10.1145/1060590.1060603

[33] D. Miccaiancio and O. Regev, "Lattice-based cryptography," https://cims.nyu.edu/~regev/papers/pqc.pdf, accessed: 2018-01-30.

[34] D. P. Chi, J. W. Choi, J. S. Kim, and T. Kim, "Lattice based cryptography for beginners," https://eprint.iacr.org/2015/938.pdf, accessed: 2018-01-30.

[35] S. Khot, "Hardness of approximating the shortest vector problem in lattices," *J. ACM*, vol. 52, no. 5, pp. 789–808, Sep. 2005. [Online]. Available: http://doi.acm.org/10.1145/1089023.1089027

[36] M. Ajtai, "The shortest vector problem in l2 is np-hard for randomized reductions (extended abstract)," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98. New York, NY, USA: ACM, 1998, pp. 10–19. [Online]. Available: http://doi.acm.org/10.1145/276698.276705

[37] D. Micciancio, "The hardness of the closest vector problem with preprocessing," *IEEE Transactions on Information Theory*, vol. 47, no. 3, pp. 1212–1215, Mar 2001.

[38] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert, "Approximating shortest lattice vectors is not harder than approximating closest lattice vectors," *Information Processing Letters*, vol. 71, no. 2, pp. 55–61, 1999.

[39] R. J. Mceliece, "A public-key cryptosystem based on algebraic coding theory," *Coding Thv*, vol. 4244, pp. 114–116, 1978.

[40] R. Overbeck and N. Sendrier, *Code-based cryptography.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 95–145. [Online]. Available: https://doi.org/10.1007/978-3-540-88702-7_4

[41] N. Sendrier, "Code-based cryptography: State of the art and perspectives," *IEEE Security Privacy*, vol. 15, no. 4, pp. 44–50, 2017.

[42] ——, "Code-based cryptography," in *Encyclopedia of Cryptography and Security.* Springer, 2011, pp. 215–216.

[43] E. Berlekamp, R. McEliece, and H. van Tilborg, "On the inherent intractability of certain coding problems (corresp.)," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, May 1978.

[44] T. Matsumoto and H. Imai, "Public quadratic polynomial-tuples for efficient signature-verification and message-encryption," in *Advances in Cryptology — EUROCRYPT '88*, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and C. G. Günther, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 419–453.

[45] J. Ding, J. E. Gower, and D. Schmidt, *Multivariate Public Key Cryptosystems (Advances in Information Security).* Springer-Verlag New York, Inc., 2006, pp. 11–12. [Online]. Available: https://books.google.no/books?hl=en&lr=&id=New-AAAAQBAJ&oi=fnd&pg=PP10&dq=multivariate+cryptosystems+springer+2006&ots=2eztuPevD9&sig=eVzeEvJGTYQ5VzK9DtB6uUf2JDI&redir_esc=y#v=onepage&q&f=false

[46] L. Goubin, J. Patarin, and B.-Y. Yang, *Multivariate Cryptography.* Boston, MA: Springer US, 2011, pp. 824–828. [Online]. Available: https://doi.org/10.1007/978-1-4419-5906-5_421

[47] R. C. Merkle, "A certified digital signature," in *Conference on the Theory and Application of Cryptology.* Springer, 1989, pp. 218–238.

[48] A. Hülsing, "Practical forward secure signatures using minimal security assumptions," Ph.D. dissertation, Technische Universität, Darmstadt, August 2013. [Online]. Available: http://tuprints.ulb.tu-darmstadt.de/3651/

[49] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, "Sphincs: practical stateless hash-based signatures," https://sphincs.cr.yp.to/index.html, accessed: 2018-02-01.

[50] ——, "Sphincs: Practical stateless hash-based signatures," in *Advances in Cryptology – EUROCRYPT 2015*, E. Oswald and M. Fischlin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 368–397.

[51] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology — CRYPTO '87*, C. Pomerance, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 369–378.

[52] L. D. Feo, "Mathematics of isogeny based cryptography," *CoRR*, vol. abs/1711.04062, 2017. [Online]. Available: http://arxiv.org/abs/1711.04062

[53] NIST, "Round 1 submissions," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions, accessed: 2018-02-19.

[54] D. Liu, N. Li, J. Kim, and S. Nepal, "Compact-lwe: Enabling practically lightweight public key encryption for leveled iot device authentication," Cryptology ePrint Archive, Report 2017/685, 2017. http://eprint. iacr. org/2017/685, Tech. Rep.

[55] J. Bootle and M. Tibouchi, "Cryptanalysis of compact-lwe," Cryptology ePrint Archive, Report 2017/742, 2017, https://eprint.iacr.org/2017/742.

[56] K. X. Jonathan Bootle, Mehdi Tibouchi, "Cryptanalysis of new compact-lwe," https://gist.github.com/xagawa/ee91d51a56bda5292235e52640f57707, accessed: 2018-02-19.

[57] H. Li, R. Liu, Y. Pan, and T. Xie, "Cryptanalysis of compact-lwe submitted to nist pqc project," Cryptology ePrint Archive, Report 2018/020, 2018, https://eprint.iacr.org/2018/020.

[58] H. Li, R. Liu, Y. Pan, T. Xie, J. Bootle, M. Tibouchi, K. Xagawa, and D. Liu, "Official comments - compact lwe," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/Compact-LWE-official-comment.pdf, accessed: 2018-02-20.

[59] C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 433–444.

[60] K. Akiyama, Y. Goto, S. Okumura, T. Takagi, K. Nuida, G. Hanaoka, H. Shimizu, and Y. Ikematsu, "A public-key encryption scheme based on non-linear indeterminate equations (giophantus)," Cryptology ePrint Archive, Report 2017/1241, 2017, https://eprint.iacr.org/2017/1241.

[61] W. Castryck, F. Vercauteren, J. Alperin-Sheriff, and K. Akiyama, "Official comments - giophantus," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/Giophantus-official-comment.pdf, accessed: 2018-02-20.

[62] L. Panny, "guessedonce.py," https://yx7.cc/files/guessedonce.py.txt, accessed: 2018-02-26.

[63] ——, "Official comments - guess again," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/guess-again-official-comment.pdf, accessed: 2018-02-26.

[64] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "Ledapkc: Low-density parity-check code-based public-key cryptosystem," https://www.ledacrypt.org/documents/LEDApkc_spec_latest.pdf, accessed: 2018-02-26.

[65] J. Stern, "A method for finding codewords of small weight," in *Coding Theory and Applications*, G. Cohen and J. Wolfmann, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 106–113.

[66] T. Fabšic, V. Hromada, and P. Zajac, "A reaction attack on ledapkc."

[67] T. Fabsic, V. Hromada, P. Zajac, M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "Official comments - ledapkc," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LEDApkc-official-comment.pdf, accessed: 2018-02-26.

[68] L. Galvez, J.-L. Kim, M. Jae Kim, Y.-S. Kim, N. Lee, and R. Perlner, "Mcnie: Compact mceliece-niederreiter cryptosystem," accessed: 2018-03-02.

[69] Y. Wang, P. Gaborit, P. Barreto, J.-L. Kim, and R. Perlner, "Official comments - mcnie," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/McNie-official-comment.pdf, accessed: 2018-03-02.

[70] N. Aragon, P. Gaborit, A. Hauteville, and J.-P. Tillich, "Improvement of Generic Attacks on the Rank Syndrome Decoding Problem," Oct. 2017, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01618464

[71] K. G. Paterson, *Cryptography and Coding: 9th IMA International Conference, Cirencester, UK, December 16-18, 2003, Proceedings.* Springer, 2003, vol. 2898.

[72] T. Plantard, "Odd manhattan's algorithm specifications and supporting documentation," accessed: 2018-03-06.

[73] T. Lepoint and L. T. Phong, "Official comments - odd manhatten," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/Odd-Manhattan-official-comment.pdf, accessed: 2018-03-06.

[74] D. J. Bernstein, J. Fried, N. Heninger, P. Lou, and L. Valenta, "Post-quantum rsa 20171123," accessed: 2018-03-06.

[75] A. Couvreur, M. Bardet, E. Barelli, O. Blazy, R. Canto-Torres, P. Gaborit, A. Otmani, N. Sendrier, and J.-P. Tillich, "Big quake - binary goppa quasi-cyclic key encapsulation," accessed: 2018-04-10.

[76] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. Aguilar Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, and G. Zémor, "Bike:bit flipping key encapsulation," https://hal.archives-ouvertes.fr/hal-01671903/document, accessed: 2018-04-10.

[77] J.-C. Deneuville, P. Gaborit, and G. Zémor, "Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory," in *Post-Quantum Cryptography*, T. Lange and T. Takagi, Eds. Cham: Springer International Publishing, 2017, pp. 18–34.

[78] D. Gligoroski, N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. Aguilar Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, G. Zémor, and R. Perlner, "Official comments - bike," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/BIKE-official-comment.pdf, accessed: 2018-04-10.

[79] O. Chakraborty, J.-C. Faugére, and L. Perret, "Cfpkm: A key encapsulation mechanism based on solving system of non-linear multivariate polynomials 20171129," Ph.D. dissertation, UPMC-Paris 6 Sorbonne Universités; INRIA Paris; CNRS, 2017, accessed: 2018-04-10.

[80] R. Steinfeld, M. R. Albrecht, E. Postlethwaite, and F. Virdia, "Official comments - cfpkm," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/CFPKM-official-comment.pdf, accessed: 2018-04-10.

[81] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang, "Classic mceliece: conservative code-based cryptography," https://classic.mceliece.org/nist/mceliece-20171129.pdf, accessed: 2018-04-11.

[82] ——, "Classic mceliece," https://classic.mceliece.org/, accessed: 2018-04-11.

[83] R. Avanzi, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber - algorithm specifcations and supporting documentation," accessed: 2018-03-08.

[84] R. Avanzi, L. Ducas, J. Bos, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schank, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals - cryptographic suite for algebraic lattices," https://pq-crystals.org/kyber/index.shtml, accessed: 2018-03-08.

[85] G. Banegas, P. S. L. M. Barreto, B. Odilon Boidje, P.-L. Cayrel, G. Ndollane Dione, K. Gaj, C. Thiecoumba Gueye, R. Haeussler, J. Belo Klamti, O. N'diaye, D. Tri Nguyen, E. Persichetti, and J. E. Ricardini, "Dags: Key encapsulation from dyadic gs codes," https://www.dags-project.org/pdf/DAGS_spec.pdf, accessed: 2018-04-11.

[86] ——, https://www.dags-project.org/about, accessed: 2018-04-11.

[87] D. Smith-Tone, "Official comments - dags," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/DAGS-official-comment.pdf, accessed: 2018-04-11.

[88] J. Ding, T. Takagi, X. Gao, and Y. Wang, "Ding key exchange," accessed: 2018-04-11.

[89] I. Luengo, M. Avendaño, and M. Marco, "Dme: A public key, signature and kem system based on double exponentiation with matrix exponents," accessed: 2018-03-08.

[90] M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila, "Frodokem - learning with errors key encapsulation - algorithm specifications and supporting documentation," accessed: 2018-04-11.

[91] ——, "Frodokem - practical quantum-secure key encapsulation from generic lattices," https://frodokem.org/, accessed: 2018-04-11.

[92] M.-J. O. Saarinen, "Hila5 - key encapsulation mechanism (kem) and public key encryption algorithm," accessed: 2018-04-11.

[93] D. J. Bernstein, L. Groot Bruinderink, T. Lange, and L. Panny, "Hila5 pindakaas: On the cca security of lattice-based encryption with error correction," Cryptology ePrint Archive, Report 2017/1214, 2017, https://eprint.iacr.org/2017/1214.

[94] C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, E. Persichetti, and G. Zémor, "Hamming quasi-cyclic (hqc)," http://pqc-hqc.org/doc/hqc-spec.pdf, accessed: 2018-04-11.

[95] Z. Liu and Y. Pan, "Official comments - hqc," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/HQC-official-comment.pdf, accessed: 2018-04-11.

[96] N. Aragon, O. Blazy, J.-C. Deneuville, P. Gaborit, A. Hauteville, O. Ruatta, J.-P. Tillich, and G. Zemfor, "Lake – low rank parity check codes key exchange –," accessed: 2018-04-11.

[97] D. Gligoroski, C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, A. Hauteville, D. Jao, O. Ruatta, R. Perlner, E. Persichetti, J.-P. Tillich, and G. Zemor, "Official comments - lake," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LAKE-official-comment.pdf, accessed: 2018-04-11.

[98] H. Niederreiter, "Knapsack-type cryptosystems and algebraic coding theory," *Prob. Control and Inf. Theory*, vol. 15, no. 2, pp. 159–166, 1986.

[99] K. Xagawa, M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "Official comments - ledakem," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LEDAkem-official-comment.pdf, accessed: 2018-02-27.

[100] E. Persichetti, "Secure and anonymous hybrid encryption from coding theory," in *International Workshop on Post-Quantum Cryptography.* Springer, 2013, pp. 174–187.

[101] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "Ntru prime: reducing attack surface at low cost," Cryptology ePrint Archive, Report 2016/461, p. Appendix K, 2016, https://eprint.iacr.org/2016/461.

[102] P.-L. Cayrel, C. T. Gueye, O. Ndiaye, E. Persichetti *et al.*, "Efficient implementation of hybrid encryption from coding theory," in *International Conference on Codes, Cryptology, and Information Security.* Springer, 2017, pp. 254–264.

[103] N. Aragon, O. Blazy, J.-C. Deneuville, P. Gaborit, A. Hauteville, O. Ruatta, J.-P. Tillich, and G. Zemor, "Locker – low rank paritycheck codes encryption –," accessed: 2018-04-11.

[104] D. Gligoroski, "Official comments - locker," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LOCKER-official-comment.pdf, accessed: 2018-04-11.

[105] A. Divesh, A. Joux, A. Prakash, and M. Santha, "A new public-key cryptosystem via mersenne numbers," accessed: 2018-04-11.

[106] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, and D. Stebila, "Newhope - algorithm specifcations and supporting documentation," accessed: 2018-04-11.

[107] ——, "Newhope post-quantum key encapsualtion," https://newhopecrypto.org/, accessed: 2018-04-11.

[108] D. J. Bernstein, T. Lange, and C. van Vredendaal, "Ntru prime 20171130," https://ntruprime.cr.yp.to/nist/ntruprime-20171130.pdf, accessed: 2018-04-11.

[109] ——, "Ntru prime," https://ntruprime.cr.yp.to/, accessed: 2018-04-11.

[110] J. M. Schanck, A. Hülsing, J. Rijneveld, and P. Schwabe, "Ntru-hrss-kem algorithm specications and supporting documentation," accessed: 2018-04-11.

[111] A. Hülsing, J. Rijneveld, J. Schanck, and P. Schwabe, "High-speed key encapsulation from ntru," in *International Conference on Cryptographic Hardware and Embedded Systems.* Springer, 2017, pp. 232–252.

[112] M. Albrecht, C. Cid, K. G. Paterson, C. Jung Tjhai, and M. Tomlinson, "Nts-kem," https://drive.google.com/uc?export=download&id=17t2HWJfPqSjt12t_SZJqEVCQxq3vU1ph, accessed: 2018-04-11.

[113] C. Aguilar Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, A. Hauteville, and G. Zémor, "Ouroboros-r," http://pqc-ouroborosr.org/doc/ouroboros-r-spec.pdf, accessed: 2018-04-11.

[114] A. Yamada, E. Eaton, K. Kalach, P. Lafrance, and A. Parent, "Qc-mdpc kem: A key encapsulation mechanism based on the qc-mdpc mceliece encryption scheme," accessed: 2018-04-11.

[115] Q. Guo, T. Johansson, and P. Stankovski, "A key recovery attack on mdpc with cca security using decoding errors," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2016, pp. 789–815.

[116] A. Szepieniec, "Ramstake," accessed: 2018-04-11.

[117] J. Alperin-Sheriff and A. Szepieniec, "Official comments - ramstake," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/Ramstake-official-comment.pdf, accessed: 2018-04-11.

[118] Y. Wang, "Quantum resistant random linear code based public key encryption scheme rlce," IEEE, pp. 2519–2523, 2016.

[119] J. Alperin-Sheriff, Y. Wang, and A. Elsayed, "Official comments - rlce," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/RLCE-KEM-official-comment.pdf, accessed: 2018-04-11.

[120] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, and G. Zémor, "Rank quasi-cyclic (rqc)."

[121] J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren, "Saber: Mod-lwr based kem," accessed: 2018-04-11.

[122] D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes *et al.*, "Supersingular isogeny key encapsulation november 30, 2017," 2017.

[123] R. Perlner, D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, L. De Feo, and S. Jaques, "Official comments - sike," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/SIKE-official-comment.pdf, accessed: 2018-04-11.

[124] M. Hamburg, "Post-quantum cryptography proposal: Threebears," 2017.

[125] Y. Zhao, "Official comments - kcl (pka okcn/akcn/cnke)," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/KCL-okcn-official-comment.pdf, accessed: 2018-04-12.

[126] Y. Zhao, Z. Jin, B. Gong, and G. Sui, "Okcn/akcn/cnke: A modular and systematic approach to key establishment and public-key encryption based on lwe and its variants," 2017.

[127] R. El Bansarkhani, "Kindi 20171130 submission," 2017.

[128] ——, "Kindi - post-quantum cryptography," http://kindi-kem.de/, accessed: 2018-04-12.

[129] X. Lu, Y. Liu, D. Jia, H. Xue, J. He, and Z. Zhang, "Lac lattice-based cryptosystems," 2017.

[130] J. Alperin-Sheriff, X. Lu, Y. Liu, D. Jia, H. Xue, J. He, Z. Zhang, C. J. Peikert, M. Hamburg, and M. Tomilson, "Official comments - lac," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LAC-official-comment.pdf, accessed: 2018-04-12.

[131] Y. Yu and J. Zhang, "Lepton: Key encapsulation mechanisms from a variant of learning parity with noise," 2017.

[132] N. P. Smart, M. R. Albrecht, Y. Lindell, E. Orsini, V. Osheter, K. Paterson, and G. Peer, "Lima: A pqc encryption scheme," 2017.

[133] ——, "Lima - a pqc encryption scheme," https://lima-pq.github.io/, accessed: 2018-04-12.

[134] J. Hee Cheon, S. Park, J. Lee, D. Kim, Y. Song, S. Hong, D. Kim, J. Kim, S.-M. Hong, A. Yun, J. Kim, H. Park, E. Choi, K. Kim, J.-S. Kim, and J. Lee, "Lizard public key encryption submission to nist proposal," 2017.

[135] L. T. Phong, T. Hayashi, Y. Aono, and S. Moriai, "Lotus: Algorithm specifications and supporting documentation," 2017.

[136] ——, "Post-quantum cryptography lotus," https://www2.nict.go.jp/security/lotus/index.html, accessed: 2018-04-12.

[137] L. T. Phong and T. Lepoint, "Official comments - lotus," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/LOTUS-official-comment.pdf, accessed: 2018-04-12.

[138] C. Chen, J. Hoffstein, W. Whyte, and Z. Zhang, "Nist pq submission: Ntruencrypt a lattice based encryption algorithm," 2017.

[139] D. Stehlé and R. Steinfeld, "Making ntru as secure as worst-case problems over ideal lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 27–47.

[140] O. Garcia-Morchon, Z. Zhang, S. Bhattacharya, R. Rietman, L. Tolhuizen, and J.-L. Torre-Arce, "Round2: Kem and pke based on glwr," 2017.

[141] R. Steinfeld, A. Sakzad, and R. K. Zhao, "Titanium: Proposal for a nist post-quantum public-key encryption and kem standard," 2017.

[142] ——, "Titanium: Post-quantum public-key encryption and kem algorithms," http://users.monash.edu.au/~rste/Titanium.html, accessed: 2018-04-12.

[143] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium - algorithm specifcations and supporting documentation," accessed: 2018-03-08.

[144] V. Lyubashevsky, "Fiat-shamir with aborts: Applications to lattice and factoring-based signatures," in *Advances in Cryptology – ASIACRYPT 2009*, M. Matsui, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 598–616.

[145] R. Avanzi, L. Ducas, J. Bos, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schank, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals - cryptographic suite for algebraic lattices," https://pq-crystals.org/dilithium/index.shtml, accessed: 2018-03-08.

[146] J.-C. Faugère and L. Perret, "An efficient algorithm for decomposing multivariate polynomials and its applications to cryptography," *Journal of Symbolic Computation*, vol. 44, no. 12, pp. 1676–1689, 2009.

[147] C. Bouillaguet, P.-A. Fouque, and A. Véber, "Graph-theoretic algorithms for the "isomorphism of polynomials" problem," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2013, pp. 211–227.

[148] W. Beullens, I. Luengo, and J. Alperin-Sheriff, "Official comments - dme," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/DME-official-comment.pdf, accessed: 2018-03-08.

[149] T. Plantard, A. Sipasseuth, C. Dumondelle, and W. Susilo, "Drs : Diagonal dominant reduction for lattice-based signature," accessed: 2018-03-15.

[150] O. Goldreich, S. Goldwasser, and S. Halevi, "Public-key cryptosystems from lattice reduction problems," in *Annual International Cryptology Conference*. Springer, 1997, pp. 112–131.

[151] J.-C. Faugére, L. Perret, and J. Ryckeghem, "Dualmodems: A dual mode for multivariate-based signature," accessed: 2018-03-15.

[152] T. Matsumoto and H. Imai, "Public quadratic polynomial-tuples for efficient signature-verification and message-encryption," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1988, pp. 419–453.

[153] A. Szepieniec, W. Beullens, and B. Preneel, "Mq signatures for pki," in *International Workshop on Post-Quantum Cryptography*. Springer, 2017, pp. 224–240.

[154] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru," accessed: 2018-03-16.

[155] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," Cryptology ePrint Archive, Report 2007/432, 2007, https://eprint.iacr.org/2007/432.

[156] T. Prest, P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon - fast-fourier lattice-based compact signatures over ntru," https://falcon-sign.info/, accessed: 2018-03-16.

[157] A. Casanova, J.-C. Faugére, G. Macario-Rat, J. Patarin, , L. Perret, and J. Ryckeghem, "Gemss: A great multivariate short signature," accessed: 2018-03-16.

[158] J. Patarin, N. Courtois, and L. Goubin, "Quartz, 128-bit long digital signatures," in *Cryptographers' Track at the RSA Conference*. Springer, 2001, pp. 282–297.

[159] J.-P. Aumasson and G. Endignoux, "Gravity-sphincs," accessed: 2018-03-16.

[160] G. Endignoux, "Design and implementation of a post-quantum hash-based cryptographic signature scheme," School of Computer and Communication Sciences, Master's Thesis, 2017.

[161] J. Ding, M.-S. Chen, A. Petzoldt, D. Schmidt, and B.-Y. Yang, "Gui," accessed: 2018-03-16.

[162] K.-A. Shim, C.-M. Park, and A. Kim, "Himq-3: A high speed signature scheme based on multivariate quadratic equations," accessed: 2018-03-16.

[163] W. Beullens, B. Preneel, A. Szepieniec, and F. Vercauteren, "Luov signature scheme proposal for nist pqc project," accessed: 2018-03-16.

[164] J. Patarin, "The oil and vinegar signature scheme," in *Presented at the Dagstuhl Workshop on Cryptography September 1997*, 1997.

[165] M.-S. Chen, A. Hülsing, J. Rijneveld, S. Samardjiska, and P. Schwabe, "Mqdss specifications," accessed: 2018-03-16.

[166] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Picnic - a family of post-quantum secure digital signature algorithms," https://microsoft.github.io/Picnic/, accessed: 2018-03-16.

[167] ——, "Post-quantum zero-knowledge and signatures from symmetric-key primitives," Cryptology ePrint Archive, Report 2017/279, 2017, https://eprint.iacr.org/2017/279.

[168] ——, "The picnic signature algorithm specification," accessed: 2018-03-16.

[169] ——, "The picnic signature scheme design document," accessed: 2018-03-16.

[170] C. Chen, J. Hoffstein, W. Whyte, and Z. Zhang, "Nist pq submission: pqntrusign a modular lattice signature scheme," accessed: 2018-03-16.

[171] O. Security, "Nist post quantum crypto submission," https://www.onboardsecurity.com/nist-post-quantum-crypto-submission, accessed: 2018-03-16.

[172] W. Lee, Y.-S. Kim, Y.-W. Lee, and J.-S. No, "Post quantum signature scheme based on modified reed-muller code," accessed: 2018-03-16.

[173] J. Alperin-Sheriff, W. Lee, Y.-S. Kim, Y.-W. Lee, J.-S. No, and R. Perlner, "Official comments - pqsigrm," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/pqsigRM-official-comment.pdf, accessed: 2018-03-16.

[174] W. Lee, Y.-S. Kim, Y.-W. Lee, and J.-S. No, "pqsigrm," https://sites.google.com/view/pqsigrm/home, accessed: 2018-03-16.

[175] N. Bindel, S. Akleylek, E. Alkim, P. S. L. M. Barreto, J. Buchmann, E. Eaton, G. Gutoski, J. Kramer, P. Longa, H. Polat, J. E. Ricardini, and G. Zanon, "Submission to nist's post-quantum project: lattice-based digital signature scheme qtesla," accessed: 2018-04-06.

[176] K. Fukushima, P. Sarathi Roy, R. Xu, S. Kiyomoto, K. Morozov, and T. Takagi, "Supporting documentation of racoss (random code-based signature scheme)," accessed: 2018-04-06.

[177] A. Huelsing, D. J. Bernstein, L. Panny, T. Lange, K. Fukushima, P. Sarathi Roy, R. Xu, S. Kiyomoto, K. Morozov, and T. Takagi, "Official comments - racoss," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/RaCoSS-official-comment.pdf, accessed: 2018-04-06.

[178] M.-S. Chen, A. Petzoldt, D. Schmidt, and B.-Y. Yang, "Name of proposal:rainbow," accessed: 2018-04-09.

[179] J. Ding and D. Schmidt, "Rainbow, a new multivariable polynomial signature scheme," in *Applied Cryptography and Network Security*, J. Ioannidis, A. Keromytis, and M. Yung, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 164–175.

[180] D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, and P. Schwabe, "Sphincs+ submission to the nist post-quantumproject," accessed: 2018-04-09.

[181] D. Atkins, I. Anshel, D. Goldfeld, and P. E. Gunnells, "The walnut digital signature algorithmtm specifcation," accessed: 2018-04-09.

[182] ——, "Walnutdsatm: A quantum-resistant digital signature algorithm," accessed: 2018-04-09.

[183] D. Hart, D. Kim, G. Micheli, G. P. Perez, C. Petit, and Y. Quek, "A practical cryptanalysis of walnutdsatm."

[184] W. Buellens, D. Atkins, I. Anshel, D. Goldfeld, P. E. Gunnells, and S. Blackburn, "Official comments - walnutdsa," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/round-1/official-comments/WalnutDSA-official-comment.pdf, accessed: 2018-04-06.

[185] NIST, "Submission requirements and evaluation criteria for the post-quantum cryptography standardization process," https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf, accessed: 2018-05-12.

[186] ——, "Post-quantum cryptography," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, accessed: 2018-05-12.

[187] NSA, "Commercial national security algorithm suite," https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfmhttps, accessed: 2018-05-21.