NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CRYPTOLOGY MANAGEMENT IN A QUANTUM COMPUTING ERA

by

Nathanial Owen Rosenberg

June 2012

Thesis Advisor:	Theodore Huffmire
Second Reader:	James Luscombe
Third Reader:	Albert Barreto

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2012	Master's Thesis

4. TITLE AND SUBTITLE Cryptology Management in a Quantum Computing Era	5. FUNDING NUMBERS
6. AUTHOR(S) Nathanial Owen Rosenberg	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number _____N/A_____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	A

13. ABSTRACT (maximum 200 words)

Today's most efficient and widely used cryptographic standards such as RSA rely on the difficulty of factoring large numbers to resist cryptanalysis. Asymmetric cryptography is used in a plethora of sensitive operations from online bank transactions to international e-commerce, and the Department of Defense also uses asymmetric cryptography to transmit sensitive data. Quantum computers have the potential to render obsolete widely deployed asymmetric ciphers essential to the secure transfer of information. Despite this, alternatives are not in place.

The goal of this study is to understand the alternatives to classical asymmetric cryptography that can be used as substitutes should quantum computers be realized. This study explores quantum-resistant alternatives to traditional ciphers and involves experimenting with available implementations of ciphers described the post-quantum literature as well as developing our own implementations based on descriptions of algorithms in the literature. This study provides an original implementation of hash-based digital signature and detailed instructions on its use as well as customization of the NTRU lattice-based cryptography suite, including the use of NTRU and AES together in a hybrid cryptographic protocol. This thesis will make recommendations on future work necessary to prepare for the emergence of large-scale, fault-tolerant quantum computers.

14. SUBJECT TERMS Type Keywords Here **Quantum Computing, Quantum Key Distribution, Cryptology, RSA, NTRUEncrypt, NTRU, ECCDSA, Elliptic Curve Cryptology, Public Key Cryptography, Symmetric Cryptography, Kerberos**	15. NUMBER OF PAGES 135
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

CRYPTOLOGY MANAGEMENT IN A QUANTUM COMPUTING ERA

Nathanial O. Rosenberg
Lieutenant, United States Navy
B.S., University of Maryland University College, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS TECHNOLOGY

from the

NAVAL POSTGRADUATE SCHOOL
June 2012

Author: Nathanial O. Rosenberg

Approved by: Theodore Huffmire
Thesis Advisor

James Luscombe
Second Reader

Albert Barreto
Third Reader

Dan Boger
Chair, Department of Information Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Today’s most efficient and widely used cryptographic standards such as RSA rely on the difficulty of factoring large numbers to resist cryptanalysis. Asymmetric cryptography is used in a plethora of sensitive operations from online bank transactions to international e-commerce, and the Department of Defense also uses asymmetric cryptography to transmit sensitive data. Quantum computers have the potential to render obsolete widely deployed asymmetric ciphers essential to the secure transfer of information. Despite this, alternatives are not in place.

The goal of this study is to understand the alternatives to classical asymmetric cryptography that can be used as substitutes should quantum computers be realized. This study explores quantum-resistant alternatives to traditional ciphers and involves experimenting with available implementations of ciphers described the post-quantum literature as well as developing our own implementations based on descriptions of algorithms in the literature. This study provides an original implementation of hash-based digital signature and detailed instructions on its use as well as customization of the NTRU lattice-based cryptography suite, including the use of NTRU and AES together in a hybrid cryptographic protocol. This thesis will make recommendations on future work necessary to prepare for the emergence of large-scale, fault-tolerant quantum computers.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AES	Advanced Encryption System
ASIC	Application-Specific Integrated Circuit
CAC	Common Access Card
CIA Triad	Confidentiality, Integrity, and Availability Triad
CNSS Instruction	Committee on National Security Systems Instruction
CPU	Central Processing Unit
D-H	Diffie-Hellman
DSA	Digital Signature Algorithm
ECC	Elliptic Curve Cryptography
ECDSA	Elliptic Curve Digital Signature Algorithm
EPR Paradox	Einstein, Boris, Podolsky Paradox
FCC	Finite Field Cryptology
FIPS Pub	Federal Information Processing Standards Publication
IEEE	Institute of Electrical and Electronics Engineers
IFC	Integer Factorization Cryptology
MD5	Message Digest 5
MIT	Massachusetts Institute of Technology
NIST Pub	National Institute of Standards and Technology Publication
NTRU	NTRUEncrypt Public-Key Crypto System
Qubits	Quantum Bits

RFC	Request for Comments
RSA Algorithm	Rivest, Shamir, Adleman Algorithm
SHA	Secure Hash Algorithm
SSH	Secure Shell
SSL	Secure Socket Layer
TGS	Ticket Granting System
VM	Virtual Machine

EXECUTIVE SUMMARY

Today’s most efficient and widely used cryptographic standards such as RSA rely on the difficulty of factoring to resist cryptanalysis. Asymmetric cryptography is used in a plethora of sensitive operations from online bank transactions to international e-commerce, and the Department of Defense also uses asymmetric cryptology to transmit sensitive data.

Quantum computers have the potential to render obsolete widely deployed asymmetric ciphers essential to the secure transfer of information. Despite this, alternatives are not in place.

This thesis recommends that the rest of the industry follow the lead of the Accredited Standards Committee X9 Incorporated, Financial Industry Standards, and identify a suitable alternative cipher such as the NTRUEncrypt Cryptosystem as the primary algorithm for asymmetric cryptography to replace RSA if needed. Preparations should be made now to facilitate a smooth transition. If the concern is too great that NTRU is a new algorithm, then this thesis at least recommends that it be added to the published standards as an alternative cipher implementation so that it is available to the industry in the event quantum computers abruptly come into existence.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, a huge and heartfelt thank you to my primary thesis advisor Dr. Theodore ‘Ted’ Huffmire. I will always be extremely grateful for the patience and mentoring through topics I had nearly no training in prior to my arrival at the Naval Post Graduate School. I have utmost respect towards Professor Huffmire for the knowledge and discipline he displayed within this field and advanced computer know-how… I truly feel fortunate that I was able to have Professor Huffmire as my primary thesis advisor, thank you Sir.

0Thank you also to the leaders within the field of cryptology who either provided me direct assistance like Dr. Dorothy Denning, and to the names I recognized frequently in my research of available literature like Dr. Daniel J. Bernstein, Dr. Isaac Chuang, Dr. Gilles Brassard, and Dr. Charles H. Bennett to name ONLY a few.

Although I send this thank you last, the most important thank you goes to who supported me through this thesis process and graduate school in general… this special thank you goes to my wife Delia. Gracias y Te amo mi amor.

THIS PAGE INTENTIONALLY LEFT BLANK

I. PROBLEM INTRODUCTION

A. INTRODUCTION

Today's most widely used asymmetric ciphers such as RSA (Rivest, Shamir, and Aldeman) rely on the difficulty of factoring large numbers as the mathematical basis of their resistance to cryptanalysis. Asymmetric cryptography is used in a plethora of sensitive operations from online bank transactions to international e-commerce, and the Department of Defense also uses asymmetric cryptography to transmit sensitive data.

B. PROBLEM STATEMENT

Large-scale, fault-tolerant quantum computers have the potential to render obsolete traditional asymmetric ciphers essential to the secure transfer of information. Despite this, alternatives are not in place.

C. PURPOSE STATEMENT

The intent of this two-phase, sequential mixed methods study is to understand the alternatives to traditional asymmetric cryptography that can be used to protect information in the event that large-scale, fault-tolerant quantum computers capable of factoring large integers are realized. The first phase is a qualitative exploration of quantum-resistant ciphers that satisfy the requirements for secure communication presently fulfilled by traditional asymmetric ciphers like RSA. The first phase also explores their tradeoffs in comparison to traditional methods. This phase of the study also involves reading published academic

articles in the emerging field of post-quantum cryptography. The second phase of the study involves experimenting with available implementations of quantum-resistant ciphers described in the post-quantum literature. This phase involves compiling and executing the downloaded programs and measuring the performance of encryption and decryption. For ciphers that have no available implementation, this phase also involves developing original implementations of post-quantum ciphers described in the literature. This thesis will then form conclusions about the impact of deploying an alternative encryption infrastructure based on post-quantum ciphers. This thesis will make recommendations on future work necessary to improve the performance of these alternative ciphers and lessen the impact of the sudden emergence of large-scale, fault-tolerant quantum computers.

D. RESEARCH QUESTIONS AND HYPOTHESES

1. Research Questions

What if quantum computing reduces the time to defeat traditional ciphers from millions of years by today's supercomputers to only seconds? What if we are already living in that era and unfriendly forces have such technology?

How efficient are post-quantum ciphers proposed as alternatives to traditional ciphers like RSA and Elliptic Curve Cryptography (ECC)? Do these ciphers have enough bandwidth to meet today's cryptographic workloads? What is the performance impact of deploying an alternative cryptographic infrastructure based on post-quantum ciphers?

Could available implementations be used as a basis for constructing a cryptographic software library that is a viable alternative to classical ciphers?

2. Hypothesis

While alternative ciphers exist, available implementations do not satisfy all performance requirements of modern cryptographic workloads. A cryptographic infrastructure that allows for ciphers to be reconfigured dynamically will reduce the costs of switching cryptographic infrastructure quickly in response to the development of quantum computers.

E. RESEARCH METHOD

Since the first phase this mixed methods study involves qualitative analysis, this thesis begins with an analysis of published literature on quantum-resistant ciphers. This qualitative study also analyzes the tradeoffs of a dynamically reconfigurable cryptographic infrastructure that can rapidly deploy an updated cipher in the event that quantum computers compromise the strength of widely used, traditional asymmetric ciphers. The second phase of this study involves the engineering of original implementations of post-quantum ciphers described in the literature, detailed instructions on their use, and quantitative analysis of their performance. This phase also involves analysis of available implementations, demonstrating how to customize them to a particular purpose, and analyze their performance. The qualitative phase identifies the ciphers and implementations to be explored in the quantitative phase.

THIS PAGE INTENTIONALLY LEFT BLANK

II. QUANTUM COMPUTING

A. QUANTUM COMPUTING

1. Introduction

Quantum computers harness the laws of quantum physics, i.e., the unusual properties of matter at tiny scales to achieve performance advantages over classical computers. Although rudimentary quantum computers have been built, they small error-prone, limited to solving small problems, such as factoring the integer 15 into its prime factors of 3 and 5. While much progress has been made in the physical implementation of quantum bits and gates in a variety of technologies as well as the development of quantum algorithms and quantum error correction schemes, much work remains to fulfill the vision of large-scale, fault-tolerant quantum computers. These challenges include increasing the reliability of physical implementations and lowering their cost.

Within the field of quantum information processing, an important distinction exists between quantum computing, a technology currently in its infancy, and quantum key distribution, a relatively mature technology with commercial implementations available. Both topics will be discussed below in greater detail, but this thesis will focus on the impact quantum computers are predicted to have on asymmetric cryptography. Quantum computing is an active, interdisciplinary field motivated by the promise of vastly outperforming classical computers on certain problems (Perry, 2006). One such problem is the factoring of integers, which has significant implications for

information assurance because large numbers of online banking transactions use asymmetric ciphers that rely in the difficulty of factoring to resist cryptanalysis.

2. Digital Bit Verses Quantum Bit

Quantum computers are similar to classical computers. Today's classical computers operate on binary digits, or bits, that can represent either 0 or 1. In a classical computer, operations are performed sequentially on these bits as dictated by the algorithm. Quantum computers use "qubits," or quantum bits. Qubits also may take on a definite value of 0 or 1, but they also can be placed in a "superposition" state in which there is a certain probability of measuring a 0 and a certain probability of measuring a 1. Once the measurement is taken, the qubit takes on the definite value measured. For example, in an equal superposition of 0 and 1, there is a 50 percent chance of measuring a 0 and a 50 percent chance of measuring a 1. This particular superposition can simultaneously represent both 0 and 1. A quantum computer's processing power grows exponentially because with every added qubit the number of values represented by the quantum register doubles. For example, two quantum bits in superposition can represent four values (0, 1, 2, and 3). Unlike classical computers, a single quantum gate applied to n qubits, all of which are in an equal superposition of 0 and 1, can manipulate all 2^n values between 0 and 2^n – 1 simultaneously. A 250-qubit register can represent more numbers simultaneously (using quantum superposition) than there are atoms in the observable universe. (Perry, 2006)

3. Quantum Entanglement

Quantum computers also exploit quantum entanglement. Measurement of one half of a pair of entangled particles causes the other half of the pair to take on a definite value that is correlated with the first particle measured. This phenomenon is counterintuitive, and Albert Einstein called quantum entanglement “spooky action at a distance.” Albert Einstein, Boris Podolsky, and Nathan Rosen tried to prove that one particle could not affect the other particle because of physical separation, but their study resulted in the now famous EPR paradox (Einstein, Boris, Podolsky paradox) that established that measurement of the first particle causes the second particle to take on a definite state that is correlated to the first. Furthermore, this effect is instantaneous, which makes it faster than the speed of light. Peter Shor determined how to harness entanglement and superposition to develop an algorithm for calculating discrete logarithms and the prime factors of an integer. Shor’s algorithm can factor integers in polynomial time; the fastest known classical algorithm requires exponential time. If a quantum computer that can factor large integers is built, it could defeat asymmetric encryption schemes such as RSA and Elliptic Curve Cryptography, whose strength against cryptanalysis is based on the difficulty of factoring integers.

B. HOW IT WORKS (BLACK BOX)

1. Schrodinger’s Cat Theory

A famous thought experiment for explaining the counterintuitive nature of quantum superposition is

described in “The Code Book,” by Simon Singh. Erwin Schrodinger, a Nobel Prize winner for physics in 1933, described a hypothetical scenario in which a cat is placed in an opaque box with a vial of a toxic substance that can be broken by a hammer, releasing the toxic substance, and killing the cat. The hammer is activated by a probabilistic event: a radioactive substance may or may not decay within a certain period of time. A sensor detects whether or not the decay occurred; each outcome is equally likely. If the sensor detects that decay has occurred, a hammer driven by a motor breaks the vial of poison. At the end of this period of time, the cat could be thought of as both dead and alive because we cannot see inside the box. Clearly, this thought experiment is absurd since a cat is a macroscopic animal much too large to exhibit quantum phenomena, but the contraption magnifies the quantum effect of radioactive decay, which involves tiny particles, to the macroscopic scale of an animal through the mechanism of a hammer activated by the decay of the radioactive substance.

2. Multiverse Theory

Schrodinger’s thought experiment was intended as a critique of the Copenhagen interpretation of quantum mechanics; the other interpretation of quantum mechanics is the multiverse theory. This theory states that at every decision, the universe splits into multiple copies; the number of copies is equal to the number of decisions at the junction. This theory also states that these universes are connected somehow. Therefore, photons passing through these multiverses interfere with each other, allowing one photon to be in all possible states at once.

C. QUANTUM COMPUTING VS. QUANTUM KEY DISTRIBUTION

1. Theory Verses Proven Protocol

Sometimes people confuse quantum computing and quantum key distribution. While quantum computing is in its infancy, quantum key distribution is a relatively mature technology that has already been commercialized.

2. Quantum Key Distribution

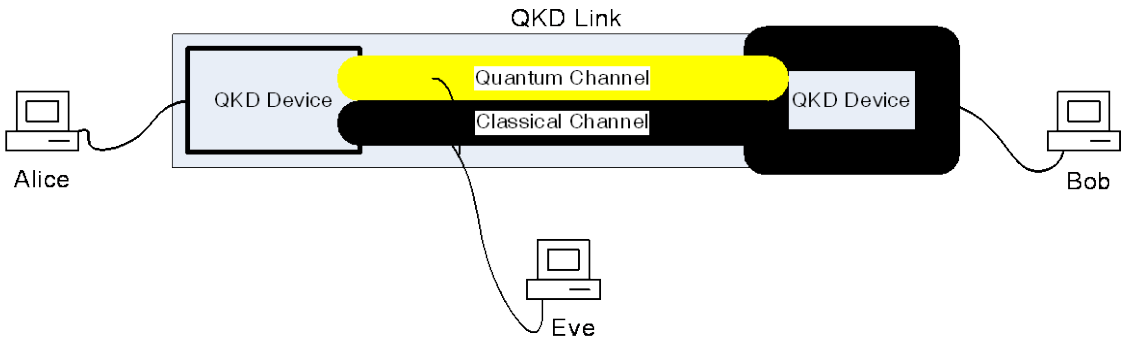


Figure 1. Structure of a Quantum Key Distribution link (From: SECOQC January 2007)

Charles Bennett and Gilles Brassard invented quantum key distribution in 1984. Quantum key distribution is used when two users want create a secure channel for electronic transmission of private information. Quantum key distribution is a method of exchanging a symmetric key for use with a classical cryptosystem. What is unique about quantum key distribution is that a malicious eavesdropper (also known as Eve) cannot eavesdrop on a quantum key exchange between two parties (also known as Alice and Bob) without detection.

Alice sends Bob a random element from a set of four polarized photons. For each photon, Bob then decides at random which canonical base to use during measurement: he chooses either the horizontal/vertical or the left/right-circular canonical base. There is also a 45/135-degree canonical base, but this base is used only by Eve and will therefore not be discussed. Bob then communicates to Alice over a classical channel the sequence bases he used to measure, and Alice tells him whether any of his decisions were incorrect. Alice and Bob then discard all of the bits that were measured with the incorrect canonical base and all bits Bob failed to receive. For all horizontal or left-circular photons, a value of 0 is registered, and for all vertical or right-circular photons, a value of 1 is registered. This series of bits can then be used as a secret key to share information as long as they determine that Eve has not been listening to their channel. This string of bits is known as the “raw quantum transmission.” (Bennet, Bessete, Brassard, Salvail, Smolin, 1991)

In order for Eve to successfully eavesdrop on a quantum key distribution channel without being detected, assuming Eve has unlimited resources, Eve must be able to intercept and resend, or split photons sent from Alice to Bob. If Eve attempts to intercept and resend photons, it is extremely hard for Eve to intercept a photon, decide the correct polarization to read, read the photon, and send a new photon with the same polarity read by Eve with the correct amount of photon intensity to enable Bob’s detectors to read Eve’s new photon. To illustrate how hard it would be for Eve to intercept and retransmit a photon, recall that a photon can be polarized in three canonical

bases or pairs. Therefore, Eve has four options from which to choose, giving Eve at best a 75 percent probability of sending the correct photon down the channel without being detected at a transmission rate between 1 and 10 kilobits per second. (Alleaume, 2007) In order to detect whether or not Eve has been listening, Bob must confirm a series of randomly selected measurement readings with Alice over the unsecure line. For example, if Alice confirmed the correct canonical base to measure as being either horizontal or vertical, Bob would then tell Alice he measured horizontal. Alice would then confirm he had the correct or incorrect measurement, and they both would then discard that bit from their key. If enough correct measurements were made in the absence of false measurements to Alice and Bob's satisfaction, they would determine that the key is secure and use it for transmitting their secret data. In contrast, if too many errors were detected, they would assume the Eve was eavesdropping and start the process over again. Eve could also perform a photon splitting attack, but the technical details of this attack are beyond the scope of this thesis.

The quantum key distribution methodology mentioned above is a very basic system. Alice and Bob could secure the classical communication line using previously exchanged secure keys via quantum key distribution. Therefore, every new successfully established key created by quantum key distribution could be added to a database of secure keys. (Bennet, Bessete, Brassard, Salvail, Smolin, 1991)

D. CAPABILITY/LIMITATIONS OF QUANTUM COMPUTING

1. One Will Not Completely Replace the Other

Although quantum computers have the potential to dramatically outperform classical computers, they will only do so for a few applications such as Shor's Algorithm, Grover's Algorithm, and the simulation of quantum physics. The successful construction of working quantum computers also has the potential to experimentally validate quantum theory. The ability to efficiently factor large integers makes asymmetric ciphers such as RSA vulnerable to quantum computers. (Nielson, M. A. & Chuang, I. L. 2002) Therefore, Quantum computers will not replace classical computers except for certain problems. To give the reader an idea of how much processing power quantum computers will have for their specific uses, it is important to understand that a register of 250 qubits (all in superposition) can represent more numbers than there are atoms in the universe. (Deutsch, D. & Ekert, A. 1998) Classical computers available today have processors with transistor counts approaching one billion and (DRAM) memories on the order of gigabytes.

2. Secure Online Transaction Protocols Cracked

Large-scale, fault-tolerant quantum computers will make any application that uses some of the most common cryptographic applications for online secure transactions vulnerable. For example, online banking and shopping or any online secure transactions that use Secure Socket Layer, the key-lock many people look to see is active

before proceeding with their online transaction, will be compromised if measures are not put in place before quantum computers are realized.

E. WHERE WE ARE TODAY WITH QUANTUM COMPUTING

1. Quantum Discrete Log and Factoring, 1994

In November 1994, the Foundations of Computer Science published Peter W. Shor's "Quantum Computation: Discrete Log and Factoring" research paper at their 35th annual symposium. This paper demonstrates how quantum computers can take discrete logarithms and factoring problems that become exponentially harder on classical computers as the numbers become larger, and reduce the computational time down to polynomial time on quantum computers (e.g., going from 1,000,000 years of computation down to 1 month).

2. Quantum Mechanics Help in Searching, 1997

Building on Shor's algorithm, Lov K. Grover in 1997 published his research paper, "Quantum Mechanics help in searching for a needle in a haystack." This paper outlined how quantum mechanics can speed up different search applications over unsorted data in contrast to classical computers. This algorithm has several useful applications including boolean satisfiability, classical random walk (i.e., diffusion), and signal processing.

3. Realization of Shor's Algorithm, 2001

In December 2001, Isaac L. Chuang et al. published their findings from their implementation of a seven-qubit quantum computer that could factor 15 into its prime

factors. The paper is titled “Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance.”

4. Scalable Quantum Logic Array, 2005

In September 2005, Tzvetan S. Metodi et al. published “A Quantum Logic Array Microarchitecture: Scalable Quantum Data Movement and Computation.” This proposes a quantum logic array architecture for building fault-tolerant and scalable quantum computers. Metodi et al. apply concepts from classical computer architecture to the design of large-scale quantum computers, which will require millions of quantum bits and gates.

5. Quantum Threshold Theorem

An important factor limiting the scalability of quantum computers is explained by the Quantum Threshold Theorem. In June 1999, Dorit Aharonov and Michael Ben-Or published “Fault-Tolerant Quantum Computation with Constant Error Rate.” This paper states that Shor’s assumption that the probability for an error in a qubit or gate decays with the size of the computation is physically unreasonable. Aharonov and Ben-or show that once a specific error rate threshold is met, quantum computers will have overcome all the physical limitations preventing the realization of quantum computers. They also state, “the point at which the physical data meets the theoretical threshold is where the quantum computation becomes practical.” However, no team has yet built a physical implementation of a quantum bit or a set of universal quantum gates that satisfy the minimum reliability requirements of the threshold theorem.

Essentially, the threshold theorem states that physical implementations must reach a minimum threshold of reliability before quantum error correction schemes can be effective and practical. One cannot simply use unreliable quantum bits that do not meet the threshold and then apply to them aggressive quantum error correction to compensate for an unreliable technology.

F. IF QUANTUM COMPUTING FOLLOWS MOORE'S LAW

1. Classical Moore's Law

Today's computers have followed closely, but not exactly, a statement made by Gordon E. Moore back in the 1970's that the number of transistors that can be placed on a fixed piece of silicon doubles about every 18 months. Therefore, given a fixed CPU size, the computing power would theoretically double every 18 months because the processor would have twice the number of transistors. The diagram (Figure 2) shows the transistor density of various processors over time.

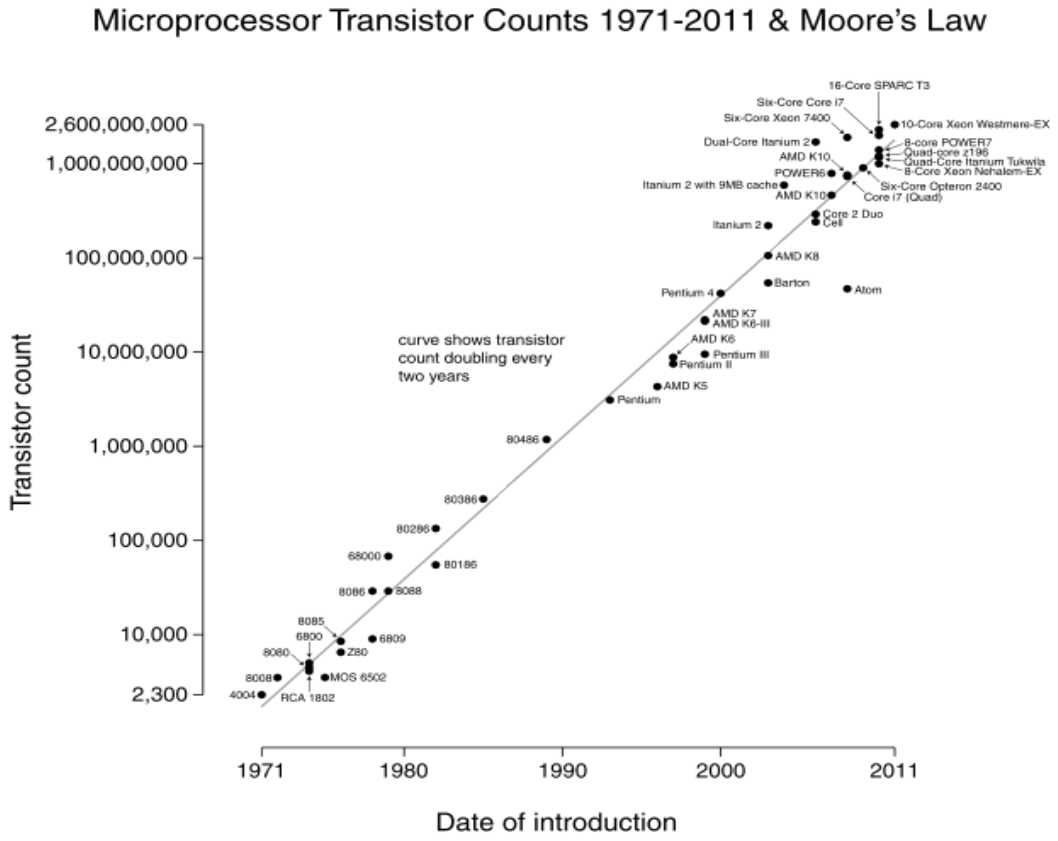


Figure 2. Microprocessor Transistor Counts—Moore's Law (From:http://en.wikipedia.org/wiki/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg)

2. Quantum Moore's Law

Assume for a moment that quantum computers will follow Moore's Law in a similar manner. Specifically, assume that the sizes of the qubit register will double every 18 months. Figure 3 shows how much processing power quantum computers would have in relation to classical computers if the qubit register were to double every 18 months.

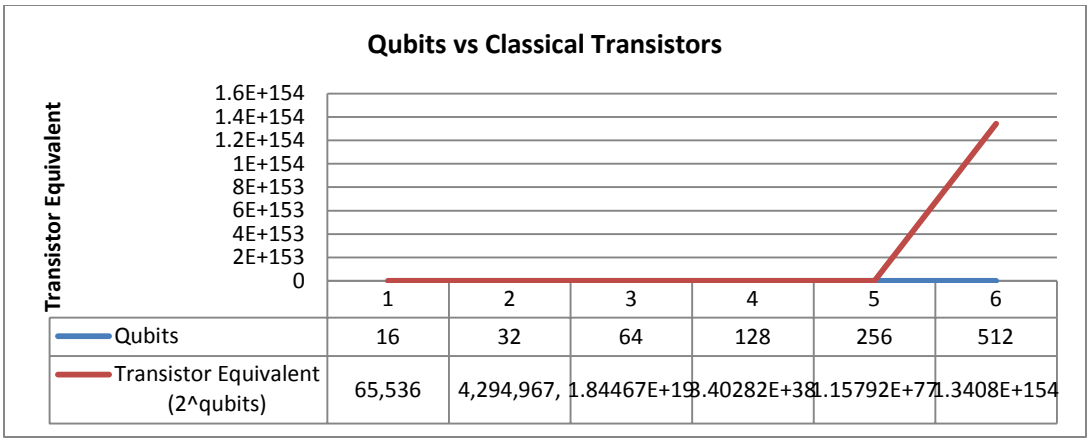


Figure 3. Qubits vs. Classical Transistor Equivalents

Keep in mind that a typical classical CPU contains on the order of one billion transistors. Figure 3 assumes that the first quantum computer that achieves the requirements as laid out in the Quantum Threshold Theorem has a register of 16 qubits, and the figure also assumes that the size of the qubit register will double every 18 months as per Moore's Law. Therefore, if the first quantum computer register has 16 qubits, after the first iteration of Moore's Law, quantum computers would have the processing power equivalent to a 4.3 terahertz computer, and the processing power would continue to grow exponentially. Recall from the discussion above that a 250-qubit register could hold more numbers in superposition then there are atoms in the universe. It is estimated that there are 10^80 atoms in the known universe, and Figure 3 shows that after 4.5 years, if quantum computing follows Moore's Law, we will have computers that can hold more numbers in superposition then there are atoms in the universe.

The problem with the above argument is that it ignores the cost of quantum error correction, which is significant.

Designing an effective large-scale, fault-tolerant quantum computer architecture requires balancing the gains of quantum parallelism against the costs of quantum error correction. In a “winning” design, the gains of parallelism exceed the costs of error correction by a big enough margin to make the whole enterprise worthwhile. Indeed, if quantum error correction were not necessary, then a 250-qubit register could hold in superposition more states than there are atoms in the universe. Similarly, to factor an n-bit number would require on the order of n quantum bits. However, it is speculated that to factor a number on the order of a thousand bits will require on the order of a million quantum bits and a million quantum gates. This increase is the cost imposed by quantum error correction, which requires additional (redundant) quantum bits and gates.

The above argument also ignores the fact that it is unknown whether progress in physical implementations of quantum computers will follow Moore’s Law. …we shall see.

III. POST QUANTUM CRYPTOLOGY

A. CLASSICAL CRYPTOGRAPHY

1. Cryptography

Originating from the Greek words “kryptos” and “graphia” meaning “hidden” and “writing” respectively, the science of keeping messages secret through encryption and decryption is cryptography. Today we use different forms of cryptography to prohibit data from being read by unauthorized users, and to prohibit data from being changed unintentionally or maliciously. The majority of cryptographic algorithms can be categorized into either symmetric or asymmetric cryptography. There are also one-way hash functions, which act like a fingerprint of the message. Cryptographic hash functions can be used to protect data integrity, but they are not used to encode and decode data. Hash functions are “one-way” in that given the output of the hash function, which is referred to as the hash digest; it is very hard for an adversary to determine the original message that was the input to the hash function. Together with symmetric and asymmetric cryptography, one-way hash functions can be used to provide confidentiality and integrity; availability is the third aspect of data protection. This chapter will describe the uses of cryptographic primitives and the hard mathematical problem on which their strength is based. This chapter will also describe which cryptographic algorithms are suspected to be vulnerable to large-scale, fault-tolerant quantum computers and which algorithms are believed to be quantum computing resistant and why.

2. Confidentiality, Integrity, and Availability

Before we get into the different forms of cryptography, we first describe the fundamental facets of data protection referred to as the "CIA Triad." The CIA Triad covers data Confidentiality, Integrity, and Availability. The Committee on National Security Systems defines confidentiality as assurance that information is not disclosed to unauthorized individuals, processes, or devices; integrity is a condition existing when data is unchanged from its source and has not been accidentally or maliciously modified, altered or destroyed; and availability is the timely and reliable access to data and information services for authorized users (CNSS Instruction No. 4009, 2010). For the rest of this thesis, we will refer to the CIA Triad when discussing the protection of data. In this thesis, we will consider cryptology to be useful for providing data confidentiality and integrity but not availability.

3. Symmetric Cryptography

Symmetric cryptography, also known as secret-key cryptography, uses the same key for encryption and decryption. First, each user must agree to the secret key and find a secure location to discuss and share the key upon which they agree. It is important to note that each user that is going to participate in the secure communication must also be provided the key in a secure manner (e.g., if there are 100 different sites that are going to participate in a secure video teleconference using only symmetric cryptography, each of the 100 sites must be provided the secret key securely in advance, and this could

be a logistical problem depending on the locations of each site and depending on time constraints). This example illustrates a problem known as the key distribution problem. Once the key is distributed, then each site can use the secret key to encrypt outbound or decrypt inbound data. In general, symmetric cryptography is faster than asymmetric cryptography. Depending on the implementation, symmetric cryptography is on the order of 1,000 to 10,000 times faster than asymmetric cryptography.

4. Asymmetric Cryptography

Asymmetric cryptography, also known as public-key cryptography, helps to address symmetric cryptography's key distribution problem. Asymmetric cryptography can be used to send a file in a secure manner to a recipient that the sender has never met. The sender encrypts the message with the receiver's public key, and the receiver decrypts the message with his or her private key. Only the receiver knows the private key; therefore, only the receiver can read the message. This protects the confidentiality of the message during transmission from sender to receiver.

Asymmetric cryptography can also be used to support digital signatures. The sender computes the hash of a file and encrypts the hash with the sender's private key. Anyone can decrypt the hash value by using the sender's public key. The receiver computes the hash of the received file and compares this to the received hash digest once it has been decrypted with the sender's public key. This helps to ensure the integrity of the message during

transmission from sender to receiver. Only the sender could have sent the message because the sender's private key was used to encrypt the hash value.

To protect both confidentiality and integrity, the message can be encrypted with a one-time symmetric session key, which the sender encrypts with the receiver's public key. This is done in conjunction with a digital signature where the hash of the message is encrypted with the sender's private key. The reason for using public-key cryptography to exchange a symmetric session-key is that symmetric ciphers are faster than asymmetric ciphers, in general. If the same two devices wanted to create another secure communication channel after the termination of their previous session, they would exchange a new session key.

If you are a security manager and the security policy for your company requires the use of asymmetric cryptography, your security system must be capable of using digital certificates. These digital certificates are issued by a certificate authority, which digitally signs a message containing an identity and a public key in order to cryptographically bind them. Since the message is signed with the certificate authority's private key, anyone can verify the signature with the certificate authority's public key. Digital certificates make it possible to trust that Alice's public key really belongs to Alice and not to an imposter. If an imposter were to hack Alice's website and replace Alice's public key with the imposter's public key, then someone might inadvertently send Alice a sensitive message encrypted with the imposter's public key rather than Alice's public key. However, the imposter will

not be able to carry out this attack if Alice has a digital certificate, because the sender of the sensitive message can verify Alice’s digital certificate prior to sending the message. Verification of the imposter’s public key will be unsuccessful.

5. Cryptographic Hash Functions

Cryptographic Hash Functions are used to create a message digest, or fingerprint, of the original message. If Alice were to send an unclassified message, but wanted to ensure that message integrity was maintained or that it was not tampered with in transit to Bob, she could create a digital signature of the message using a one-way hash function. Alice then takes the fingerprint and encrypts it with her private-key and sends Bob an email that contains the original message, the encrypted fingerprint, and her public-key. Bob then receives the message, runs the original message through the same cryptographic hash algorithm Alice used, and then decrypts the fingerprint with Alice’s public-key that was encrypted with Alice's private-key. Finally, Bob compares the fingerprint Alice sent against the fingerprint he computed. If the hash values are equal, Bob knows that the message Alice sent was not tampered with while in transit.

Cryptographic hash functions differ from public-key and secret-key cryptography because asymmetric and symmetric cryptography are used to encrypt and decrypt data (i.e., encrypting and then decrypting a message results in the original message), whereas cryptographic hash functions are one-way functions. It is extremely difficult to determine the input to a hash function given only the hash

digest. The size of the hash digest depends on the algorithm. For example, some of the most common hash algorithms are the Secure Hash Algorithms (SHA) SHA-256, SHA-384, and SHA-512, each algorithm outputting a digest of 256, 384, and 512 bits respectively regardless of the input size. In order for a cryptographic hash function to be considered secure, the hash function must be able to input any amount of data; output a (normally) smaller fixed-length digest defined by the algorithm; be fast to compute; be hard to invert; and be designed so that it is very difficult for an adversary to find collisions. A hash collision occurs whenever two inputs produce the same output. Finding a collision is impossible, but it must be very difficult for the adversary to do. Among the most popular hash algorithms is MD5 (Message Digest 5), which has the smallest hash output of 120 bits, and as of 2003, no collisions had been discovered (Thorsteinson, P., Ganesh, G.G., 2003). However, MD5 is currently considered "broken," as it is now possible to find a collision in less than O(2^N) steps. Another required characteristic of a cryptographic hash function is called the avalanche effect, i.e., a small change to the input results in a very large change to the input.

To illustrate the avalanche effect, consider the DNA of identical twins. Even if their genomes only differ by a single nucleotide, the cryptographic hashes of the digital representation of their respective genomes will be completely different. To illustrate the one-way property of a hash-function, consider literature as an example. Given a 256-bit hash digest of Shakespeare's play, Romeo and Juliet, to reverse the one-way property would require

the adversary to generate the entire play from only 256-bit digest. Assuming a 96-character alphabet and given that the play is 138,386 characters long, the adversary would have to perform O(96^138386) hash computations before finding the string of characters that correspond to the play Romeo and Juliet.

B. CIPHERS BELIEVED TO BE VULNERABLE TO QUANTUM COMPUTING

1. Quantum Computing Capability Review

This section will discuss algorithms believed to be vulnerable to quantum computers. Each of these algorithms relies on the difficulty of factoring large numbers or computing discrete logarithms as the basis for their security because these are difficult problems for classical computers to solve in polynomial time. Recall that Shor's algorithm used in conjunction with quantum computers will make algorithms that rely on the difficulty of factoring or computing discrete logarithms vulnerable. The ciphers presented in Section 'C' do not rely on the difficulty of factoring or of computing discrete logarithms. Their strength lies in the difficulty of solving different hard mathematical problems. Since quantum algorithms do not exist for these problems, these ciphers are believed to be quantum computing resistant.

2. Rivest, Shamir, and Adleman (RSA)

Building on the Diffie and Hellman “Public-key Cryptosystem,” Ron Rivest, Adi Shamir, and Leonard Adleman created the RSA cipher in 1978 to ensure that the properties of the “paper mail system” were preserved in the email era; specifically, that the mail remained

confidential and could be signed. (Rivest, R.L., Shamir, A., Adleman, L., 1978) Both RSA and the Diffie-Hellman algorithms provide key exchange, but RSA added public key encryption, making RSA more versatile. (Schmeh, K.) Today, the RSA cipher is the most common form of public-key cryptology in use. After licensing a patent from the Massachusetts Institute of Technology, the RSA cipher was offered as a commercial product in 1982. (Russell, D., Gangemi, G.T., 1991)

The strength of the RSA cipher relies on the difficulty of factoring large numbers. The minimum recommended key length for RSA is 1024-bits until they year 2015; then 2048-bits will be recommended until the year 2030. (www.rsa.com)

3. Digital Signature Algorithm (DSA)

Based on the difficulty of solving the discrete logarithm problem, the Digital Signature Algorithm (DSA) is used to electronically sign digital messages. The DSA is a standard specified by the National Institute of Standards and Technology and was issued in May 1994. The three functions of the DSA are to generate a key used to “sign” the message, sign the document, and verify the signature on the other end. (FIPS PUB 186)

4. Elliptic Curve Cryptography

Like the Digital Signature Algorithm, Elliptic Curve Cryptography relies on the difficulty of solving the discrete logarithm problem as the basis of its security. (NIST Pub 800-57) The mathematics required for Elliptic curve Cryptography is well beyond the scope of this thesis.

An excellent tutorial including Java applets is located on the website of Certicom (http://www.certicom.com/index.php/ecc-tutorial). Elliptic Curve Cryptography is a newer version of Public-Key Cryptology and can provide the same level of security as RSA, but with smaller key sizes. This enables platforms with constrained resources such as handheld wireless devices to use strong cryptographic algorithms. With all variables being equal, Elliptic Curve Cryptography can run more transactions per second than RSA. (Mogollon, M., 2008) Figure 4 is from National Institute of Standards and Technology Special (NIST) Publication NIST PUB 800-57, March 2008, Recommendation for Key Management.

Table 2: Comparable strengths

Bits of security	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
80	2TDEA[19]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

Figure 4. NIST Comparable Key Strengths
(From:NIST Publication 800-57)

In the NIST comparable key strengths table, D-H stands for Diffie-Hellman, and FFC and IFC stand for Finite Field Cryptology and Integer Factorization Cryptology,

respectively. This table clearly articulates that Elliptic Curve Cryptology (ECC) achieves the same level of security as symmetric ciphers with keys having roughly twice the number of bits, whereas the D-H and RSA algorithms have significantly larger key sizes. A symmetric key length of 112 bits is the standard minimum as of 2010 and will be until 2030. The National Security Agency has adopted ECDSA (Elliptic Curve Digital Signature Algorithm) because it is considered to be second-generation public key cryptography and offers relatively smaller key sizes in contrast to the first generation (e.g. D-H and RSA). The National Security Agency stated, "As vendors look to upgrade their systems they should seriously consider the elliptic curve alternative for the computational and bandwidth advantages they offer at comparable security." (http://www.nsa.gov/business/programs/elliptic_curve.shtml)

C. CIPHERS BELIEVED TO BE RESISTANT TO QUANTUM COMPUTING

1. Hash-Based Digital Signature Schemes

To recap, hash-based algorithms are one-way algorithms that take in any size of input in the form of bits and produce a digital signature that is a fixed size that depends on the algorithm. In order for a cryptographic hash function to be considered secure, the hash function must be able to input any amount of data, output a fixed-length digest, be fast to compute, be hard to invert, and produce few collisions. In other words, if Y = F(x), where Y is the digest, F is the hash algorithm, and x is the

message, if an adversary obtained Y and knew F, it would be "effectively impossible to compute 'x.'" (Merkle, R. C., 1979)

Since hash-based algorithms are only considered secure if they are collision resistant, hash-based signature schemes are considered to be the "…most important post-quantum signature candidate" (Bechmann, J., Dahman, E., Szydlo, M., 2009) because the security of these functions relies on their collision resistance. The digital signature schemes are also useful because they can be implemented in hardware and software making them prospective alternatives to the popular RSA and elliptic-curve digital signature schemes that are predicted to be vulnerable in a quantum computing era. (Bechmann, J., Dahman, E., Szydlo, M., 2009)

The mathematics of the following hash-based signature schemes is beyond the scope of this thesis, but Bechmann, Dahman, and Szydlo state that the Lamport-Diffie One-Time Signature Scheme, Winternitz One-Time Signature Scheme, and Merkle Signature Scheme (Merkle's tree) are all hash-based algorithms, with Merkle's tree being the most efficient. The Merkle scheme is actually a multi-time signature that employs a version of the Lamport-Diffie signature scheme, but the Merkle scheme can convert any one-time signature scheme to create a multiple-use or multi-time signature scheme. (Garcia, L. C.)

2. McElice Code-Based Encryption System

Code-based cryptography relies on error-correcting codes such as the McEliece Hidden-Goppa-Code cryptosystem, in which the security of the algorithm relies on the

difficulty of decoding a general linear code in polynomial time. (Berlekamp, E. R., McEliece, R. J., Van Tilborg, H. C., 1978) Although not as efficient as RSA, the McEliece Hidden-Goppa-Code cryptosystem is expected to hold up to quantum computers. The current drawback to the McEliece cryptosystem is that the key sizes are in the millions of bits, whereas RSA key sizes are in the thousands of bits. (Bernstein, D. J., 2009)

The McEliece cryptosystem uses three algorithms to create the public and private-key pair, to encrypt the message, and to decrypt the message. To create the public key ($G_{(prime)}$, $T_{(errors)}$) Alice selects a binary linear code $C_{(linear\ code)}$ capable of correcting $T_{(errors)}$ and creates a generator matrix $G_{(generator\ matrix)}$ of ($N_{(length)}$, $K_{(dimension)}$). The generator matrix is hidden using a random non-singular binary square substitution matrix $S_{(substitution\ matrix)}$ of ($K_{(dimension)}$ x $K_{(dimension))}$ size and a random permutation matrix $P_{(permutation\ matrix)}$ of ($N_{(length)}$ x ($N_{(length)}$) size. (Heyse, S., 2009) A permutation matrix is also a binary square matrix that has exactly one entry of 1 for any column and row with 0 in all other spaces. Figure 5 provides an example of a permutation matrix. $G_{(prime)}$ is created by computing the product of $G_{(generator\ matrix)}$, $S_{(substitution\ matrix)}$, and $P_{(permutation\ matrix)}$.

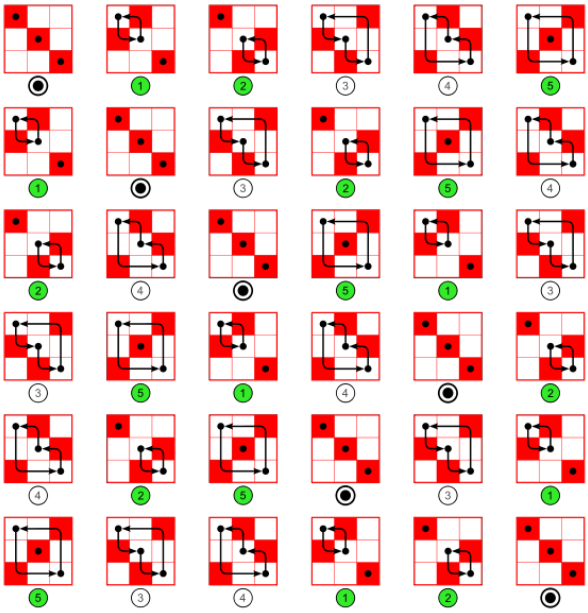


Figure 5. Permutation Matrix Example (From:http://en.wikipedia.org/wiki/File:Symmetric_group_3;_Cayley_table;_matrices.svg)

The secret key is the combined knowledge of three different matrices that created the public key: $G_{(\text{generator matrix})}$, $S_{(\text{substitution matrix})}$, and $P_{(\text{permutation matrix})}$. (Heyse, S., 2009)

Encryption and decryption is similar to other public-key cryptosystems: if Bob wishes to send Alice an encrypted message, Bob uses Alice's public key ($G_{(\text{prime})}$, $T_{(\text{errors})}$), but he also introduces error into the message not to exceed the amount of $T_{(\text{errors})}$. To decrypt the message, Alice uses her secret key to produce the plaintext from the ciphertext provided that Bob did not introduce an error larger then $C_{(\text{linear code})}$. (Heyse, S., 2009)

3. NTRU Lattice-Based Cryptography

Recall that quantum computers will excel at cracking algorithms that rely on the difficulty of factoring large

numbers or solving the discrete logarithm problem as the basis for their security. Therefore, cryptologists have searched for a different mathematical problem to use as the basis of an algorithm's security, and Lattice problems are one such problem. (Perlner, R. A., Cooper, D. A., 2009) "Lattice based systems provide a good alternative since they are based on a long-standing open problem for classical computation." (Hallgren, S, Vollmer, U., 2009)

Collectively, the basis of a lattice is a set of vectors that can be expressed as a sum of integer multiples of a set of *n* vectors. "(It is important to) note that there are an infinite number of different bases that will all generate the same lattice." (Perlner, R. A., Cooper, D. A., 2009) Two problems believed to be hard for classical and quantum computational models are solving either the closest vector problem or shortest vector problem of high-dimensional lattices. The mathematics of Lattice-Based Cryptography is beyond the scope of this thesis, but there is commercial deployment of Lattice-Based Cryptography, and the NTRUEncrypt Public-Key Cryptosystem "appears to be the most practical." (Perlner, R. A., Cooper, D. A., 2009)

"The (NTRU) encryption procedure uses a mixing system based on polynomial algebra and reduction modulo two numbers *p* and *q*, while the decryption procedure uses an unmixing system whose validity depends on elementary probability theory. The security of the NTRU public-key cryptosystem comes from the interaction of the polynomial mixing system with the independence of reduction modulo *p* and *q*. Its security also relies on the (experimentally observed) fact that for most lattices, it is very difficult

to find extremely short (as opposed to moderately short) vectors." (Hoffstein, J., Pipher, J., Silverman, J. H., 1998)

Like Elliptic Curve Cryptology, the NTRU Public-key cryptosystem might become an alternative algorithm for computing devices that require high-performance security but that have fewer resources then a typical PC, e.g., handheld devices such as tablets, cellphones, and PDAs. With key length requirements of 112 bits or greater, the NTRU cryptosystem is able sign and verify signatures, encrypt messages, and decrypt messages faster than Elliptic Curve Cryptology and therefore faster than RSA.

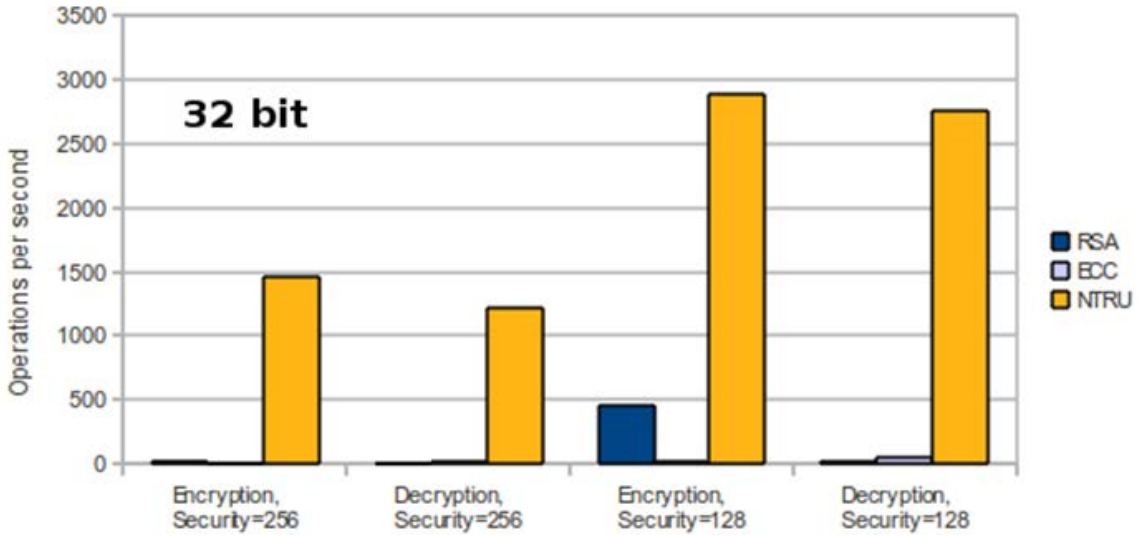


Figure 6. Encryption/Decryption Operations per second for RSA, Elliptic Curve Cryptology, and NTRU for a 32-bit processor(From:http://tbuktu.github.com/ntru/)

Parameters				public key and signature size				sign/s			vfy/s		
k	*N*	*d*	*q*	NTRU	ECDSA key	ECDSA sig	RSA	NTRU	ECDSA	Ratio	NTRU	ECDSA	Ratio
80	157	29	256	1256	192	384	1024	4560	5140	0.89	15955	1349	11.83
112	197	28	256	1576	224	448	~ 2048	3466	3327	1.04	10133	883	11.48
128	223	32	256	1784	256	512	3072	2691	2093	1.28	7908	547	14.46
160	263	45	512	2367	320	640	4096	1722	—	—	5686	—	—
192	313	50	512	2817	384	768	7680	1276	752	1.69	4014	170	23.61
256	349	75	512	3141	512	1024	15360	833	436	1.91	3229	100	32.29

Figure 7. Signatures and signature verifications per second for Elliptic Curve Digital Signature and NTRUSign (From:Practical lattice-based cryptography NTRUEncrypt and NTRUSign, Hoffstein, J. et al)

As evident in Figures 6 and 7, the NTRU cryptosystem offers higher performance than ECDSA, but unfortunately NTRUEncrypt did not have a formal proof of security like RSA, Elliptic Curve Cryptology, and other practical schemes until 2008. (Naslund, M. Shparlinski, I. E., Whyte, W., 2003). NTRU received much popular support for ten years until the proposed IEEE standard P1363.1 became an approved standard in December 2008. Now IEEE Std 1363.1™-2008, IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices, is an international standard and is starting to be adopted by commercial vendors.

4. Multivariate Quadratic Public-Key Cryptography

Although the mathematics is beyond the scope of this thesis, Daniel Bernstein, a leader in this field, lists Multivariate-Quadratic-Equations Public-Key Cryptography as another alternative in his book, “Introduction to Post Quantum Cryptography.”

The security of the Multivariate-Quadratic-Equations Public-Key Cryptography is based on the difficulty of solving nonlinear equations over a finite field, which is considered to be an NP-hard problem. This algorithm has been under intensive study for the last couple of decades, but experts do not recommend using Multivariate-Quadratic-Equations for protecting security-critical applications yet because the basis of its security is not well understood, and vulnerabilities are being found on a regular basis. (Bernstein, D. J., 2009)

5. Advanced Encryption System (AES) - Symmetric (Secret-Key) Cryptography

Throughout this thesis, references have been made to key-size security equivalence to symmetric cryptography when discussing RSA, Elliptic-Curve Cryptography, and NTRU Lattice-Based Cryptography. The Advanced Encryption Standard (AES) is the standard cipher for symmetric cryptography, also known as “secret-key” cryptography. Recall that secret-key cryptography is the fastest and provides the most encryption strength per bit of key and that a major purpose of slower public-key cryptography is to help facilitate the exchange of a symmetric session key.

In the late 1990s, the National Institute of Standards and Technology (NIST) held a competition to replace the successful symmetric algorithm DES (Data Encryption Standard) that was developed by IBM in the 1970s. NIST reached out to the cryptographic community for those who were interested in submitting an algorithm to be the new standard. After fifteen nominated AES candidates, the Rijndael algorithm, developed by two Belgians, Joan Daemen

and Vincent Rijmen, won the contest, and their cipher was announced as the winner in October 2000. As per the requirements of the NIST contest for AES, the Rijndael algorithm supports block lengths of 128, 192, and 256-bits. (Schmeh, K, 2003)

Although a brute-force attack for a key length of 128 bits is "…out of the question for Rijnadael," as Klaus Schmeh stated, there is a small concern for the reliability of the algorithm because it is fairly new. However, the strength and usability of AES is evident by the US Government's approval of its use for classified information processing. Specifically, the National Security Agency approved all block lengths of the AES algorithm to protect classified information up to Secret, but only the 192- and 256-bit block lengths are approved for Top Secret material as per the Security on National Security Systems Policy, CNSS Policy No. 15, Fact Sheet No. 1, National Policy on the Use of Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information, June 2003.

IV. APPLICATIONS OF CRYPTOGRAPHY CURRENTLY IN USE

A CIPHERS VULNERABLE TO QUANTUM COMPUTERS

1. Introduction

Suppose that CNN were to report that a large-scale, fault-tolerant quantum computer has been developed; how would users of cryptography cope? This section focuses on the potential impact of quantum computers on various parties from individual users to the world economy.

2. Online Banking Statistics

Since 2009, the American Bankers Association reported that online banking is the preferred method of performing banking transactions. Figure 8 shows the online banking trend from 2007 to 2011 for all age groups. In 2007, online banking made up roughly 23 percent of all banking transactions, but in 2011, online banking accounted for roughly 61 percent of all banking transactions. This report does not include the statistics from other large banking spheres such as the European Union or banking organizations within the Asia-Pacific region.

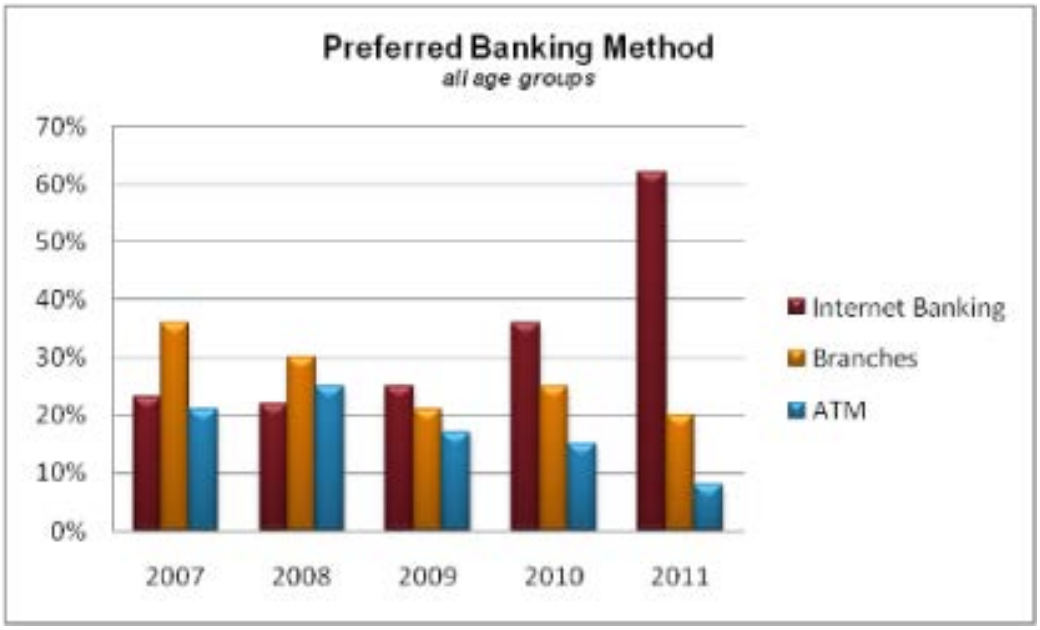


Figure 8. Preferred Banking Method 2011 Report (From:http://www.aba.com/Press+Room/090811ConsumerPreferencesSurvey.htm)

3. Online Shopping Statistics

The sudden emergence of quantum computers could have a significant impact on the world economy if users of cryptography are caught unprepared. The Nielsen Company reported in 2007 that 875 million consumers had shopped online, an increase of over 40 percent since the previous survey was conducted in 2005. The survey also found that 85 percent of all Internet users had conducted an online transaction. Comparing the percentage of those who had Internet access and those who used the Internet to conduct an online transaction, the countries that had the highest online shopping percentages in 2007 were South Korea at 99 percent, the United Kingdom at 97 percent, Germany at 97 percent, Japan at 97 percent, and in eighth place, the

United States, where 94 percent of those who had Internet access used the Internet to conduct an online transaction.

B. TECHNOLOGIES CURRENTLY IN USE FOR SECURING INTERNET TRANSACTIONS

1. Introduction

The reason why most of the transactions outlined in Section 'A' of this chapter are vulnerable is that they use the technologies discussed in this section. The sudden emergence of a large-scale, fault-tolerant quantum computer would render the following cryptographic technologies ineffective.

2. Secure Socket Layer (SSL)

Invented by Netscape in the 1990s, the Secure Socket Layer (SSL) uses the Transport Control Protocol (TCP) to provide encryption, authentication, and integrity for HTTP, LDAP, and POP3 applications. SSL is the most commonly used technology for securing online transactions.

Secure Socket Layer was designed to have the server authenticate itself to the user. During a SSL session, the user requests to setup a secure channel with the server. The server then sends to the user the server's public key so that the user can validate whether or not the server is using a trusted certificate authority. A certificate authority is the issuer of certificates and will be discussed further in the Public-Key Infrastructure section below. If the user confirms that the certificate authority is trustworthy, the client creates a session key that is based on a symmetric encryption algorithm and encrypts it with the server's public key so that only the server can

decrypt the session key. Once the server decrypts the session key, secure communication using symmetric cryptography can begin.

The problem is that the Secure Socket Layer protocol uses RSA and Diffie-Hellman for the majority of its public-key transactions. SSL also can use Fortezza Cards that have been used by government, military, and banking institutions to protect sensitive data, but since Fortezza Cards are not common, they go beyond the scope of this thesis. Therefore, unless you are required to use a Fortezza card for your Internet transaction, you are using either RSA or Diffie-Hellman as your public-key algorithm, and this renders your SSL session vulnerable to quantum computers.

3. Secure Shell (SSH)

As a secure alternative to Telnet for remote networking administration, Secure Shell (SSH) can also be used to secure protocols like HTTP and FTP that are used to transfer data from websites or files like the Secure Socket Layer system. Secure Shell was originally created in 1996 by Tatu Ylonen at the Helsinki University of Technology in Finland. Ylonen started his own company, SSH Communications, and later improved the protocol in 1998 when he released SSH2. After working with the Internet Assigned Numbers Authority, Ylonen's work was implemented as an Internet Standard under RFC 4250 in 2006 as the Secure Shell Protocol.

Secure Shell uses RSA as its Public-Key Algorithm to initially set up the session key. Therefore, since RSA uses factoring large numbers as the basis of its security, and

quantum computers reduce the time to factor large numbers from exponential to polynomial time, Secure Shell in its current form will be vulnerable in a quantum computing era.

4. Digital Certificates

In the sections above, we have discussed how public-key cryptography is used when Alice wants to verify that Bob is who he claims to be while exchanging a session key to conduct secure communication between Alice and Bob. Digital certificates, also known as certificates, enable this transaction to occur. Digital certificates are used to certify that Alice is in fact Alice. With public-key cryptography, recall that two keys are generated, one being the public-key that is available to the public, and the other being Alice's private-key that is held securely in Alice's possession. The private-key can be held on a disk, programmed into a smartcard, or loaded onto Alice's personal computer, with the latter being less secure. Digital certificates bind Alice's identity to her public key. Along with the public key, the digital certificate holds other information like Alice's name, a serial number, the Certificate Authority who issued Alice her certificate, the algorithm used to generate the digital certificate, and the certificate's expiration date. With this information, Bob could first see that the certificate is assigned to Alice, verify that the certificate has not expired, identify whether or not the Certificate Authority itself is a trustworthy issuer of certificates, and then verify Alice's public key.

Digital certificates are extremely popular because once the digital certificates have been issued to a user,

iterative validation can occur, or validation that is pushed to individual PCs vice a central server. If validation were recursive in nature and could only be performed by the Certificate Authority, this would make denial-of-service attacks easier because a successful denial-of-service attack on the single point of failure would disrupt all users who were issued certificates from that Certificate Authority. Therefore, because digital certificate validation is iterative and decentralized by design, millions of secure transactions may occur without the threat of availability attacks on a central server. Unfortunately, the system is only as secure as the strength of algorithm that is used. Under RFC 3279, Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile, the public-key algorithm used is RSA. We have shown that the use of RSA as a public-key algorithm will make digital certificates vulnerable once large-scale, fault-tolerant quantum computers come into existence, if no suitable alternative algorithm is put in place before the emergence of quantum computers.

5. Digital Signatures

Using the private-key contained in Alice's digital certificate, Alice is able to sign the messages she sends to Bob so that Bob recognizes the signature to be only from Alice. Although digital signatures are supposed to mimic the actual signing of hard-copy letters, they are not physical signatures that are scanned and attached to letters because this format could easily be copied by Eve and used to send erroneous emails with Alice's signature,

i.e., through a replay attack. In contrast to a physical signature, a digital signature is a combination of the message, the hash of the message, and a hash of the message encrypted with Alice's private-key. Alice sends her signed message along with her public certificate to Bob, who uses Alice's public key to decrypt the hash, after first using Alice's certificate to validate her public key (using the certificate authority's public key to verify the certificate). To protect the confidentiality of the message, it may be encrypted with a symmetric cipher, and the symmetric key is encrypted with Bob's public key (and decrypted with Bob's private key). Bob computes the hash of the message and compares it with the decrypted hash value.

Like digital certificates, digital signatures are only as good as the algorithm that makes them, and per RFC 3279, the two algorithms in use for digital certificates are DSA and ECDSA. Like RSA, quantum computers are believed to make these algorithms vulnerable because of the mathematics these algorithms are based upon. Therefore, unless another algorithm is added to RFC 3279 that is resistant to quantum computers as an alternate standard, quantum computers will make the forging of digital signatures possible.

6. Public-Key Infrastructure

One of the largest organizational public-key infrastructures in existence is the United States Department of Defense infrastructure, and this system will be used to explain the elements and processes within a public-key infrastructure that secures communication. The process begins with the issuance of a Department of Defense

identification card. A government employee, who could be military, government employed civilian, or government employed contractor, obtains an ID card by providing different forms of government-issued identification to the Department of Defense Identification office in person. If sufficient identification is present, the government employee is issued a Common Access Card (CAC), also known as a CAC Card.

A Department of Defense Common Access Card contains standard identification elements found on government-issued ID cards (e.g., driver's license) such as photograph, name, birth date, issue date, and expiration date. The CAC also contains a chip that holds the member's public and private certificates that are issued at the time the member receives his or her CAC. The ID card office receives the certificates securely from the Department of Defense Certificate Authority and installs them onto the CAC. After a process of about 20 minutes, the new employee can access websites that are secured with the Department of Defense's public-key infrastructure or digitally sign messages with the member's new digital certificates that can be verified by other members within the system.

If the government employee is a student at the Naval Postgraduate School and wishes to access the school's websites that are secured with the Department of Defense's public-key infrastructure without receiving multiple security warnings from his or her Internet browser, he or she must install the Department of Defense's root certificate onto his or her machine. A public-key infrastructure is a hierarchical system where all certificates stem from the root certificate. The root

certificate is the Certificate Authority's public key along with the Certificate Authority's digital signature of its public key. The root certificate is the only certificate that is self-signed by the owner's (Certificate Authority's) private key.

This system harnesses the strength of RSA's public-key algorithm for the protection of data confidentiality and data integrity. Furthermore, since public-key infrastructures use iterative/decentralized validation, where the user's PC can validate a website's certificates if the PC has the root-certificate installed, it is difficult for an attacker to conduct a denial-of-service attack on the Department of Defense's public-key infrastructure, making this system highly reliable for sending information in a manner that protects confidentiality, integrity, and availability. Unfortunately, this system is based on the RSA algorithm, and if an attacker had access to large-scale, fault-tolerant quantum computers, this infrastructure would be made vulnerable.

C. APPLICATIONS BELIEVED TO BE QUANTUM COMPUTING RESISTANT

1. Introduction

The following section identifies cryptographic technologies believed to be quantum computing resistant. Therefore, assuming that large-scale, fault-tolerant quantum computers have not been realized yet, there is time to implement alternative ciphers.

2. Symmetric Cryptography

Symmetric cryptography that currently protects the largest bulk of secure electronic communications will remain in a quantum-computing era because it is believed to be quantum computing resistant. Given that symmetric cryptography is believed to be resistant to quantum computers, algorithms like the Advanced Encryption System (AES), which as mentioned earlier is cleared by the National Security Agency to secure Top Secret transmission, the United States' most classified and sensitive information, will still hold strong against quantum computers. Therefore, the session keys that are exchanged within the client/server architecture for Internet transactions like the end-user's PC and his or her bank server will also remain secure for these transactions. Unfortunately, key distribution is more challenging with symmetric cryptography; therefore, symmetric cryptography must be combined with some form of asymmetric cryptography in order to distribute symmetric keys over the web. Fortunately, several quantum-resistant algorithms and implementations, such as the NTRUEncrypt Public-Key Crypto system, which uses lattice-based cryptography, are available to distribute the symmetric session keys, and NTRUEncrypt's implementation of lattice-based cryptography is believed to be quantum resistant.

3. NTRUEncrypt Public-Key Crypto System

According to techworld.com, which announced the X9.98 standard in April of 2011, "NTRUEncrypt, [is] the fastest public key algorithm you've never heard of." In contrast to RSA and Elliptic Curve Cryptology (ECC), NTRUEncrypt is

faster, more efficient, and resistant to quantum computers because the cipher is lattice-based (the mathematics of lattice-based cryptography are beyond the scope of this thesis). I.e., NTRUEncrypt is able to function fully as a public-key algorithm like the widely used RSA algorithm, but does so in a more efficient manner.

Fortunately, the techworld.com statement wasn't 100 percent accurate because obviously if the Accredited Standards Committee X9 Incorporated, Financial Industry Standards, created the X9.98 standard for financial institutions to start using NTRUEncrypt to establish secure communications for financial service in November 2010, then someone has heard of NTRUEncrypt. In fact, JAVA released an application programming interface, Bouncy Castle 1.47, containing variants that include a lightweight version of the NTRUEncrypt Public-Key Cryptosystem in March 2012 (bouncycastle.org).

Security Level (bits)	LBP-PKE Key size	ECC Key Size	RSA Key Size	LBP-PKE ops/ Sec	ECC ops/ Sec	RSA private key ops/ Sec
112	5951	224	2048	2284	951	156
128	6743	256	4096	1896	650	12
192	9757	384	7680	1034	285	8
256	12881	512	15360	638	116	1

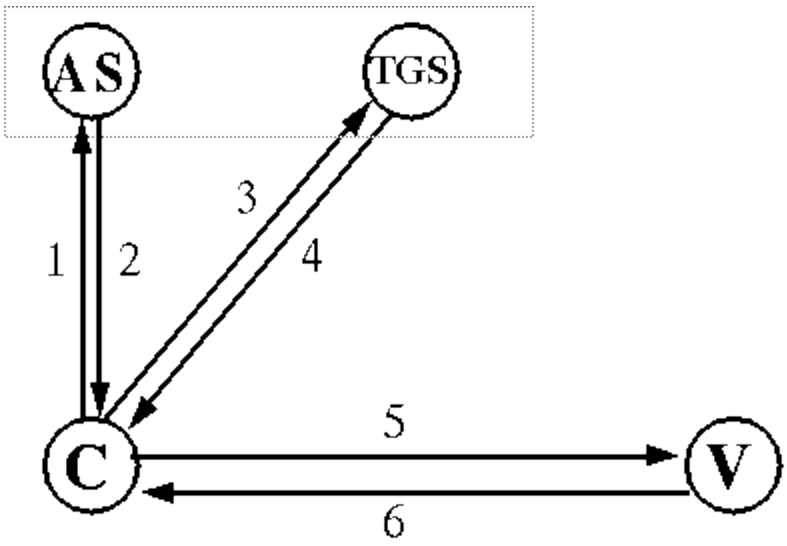
Figure 9. Relative Performance of LBP-PKE, RSA, and ECC (From:X9extra, Volume 2, Number 1, April 2011)

Figure 9 shows the relative performance for "Lattice-Based Polynomial Public-Key Encryption" (X9extra, Volume 2, Number 1, April 2011), RSA, and ECC. The reader can see via

the operations per second in the last three columns that NTRUEncrypt outperforms ECC and RSA by as much as 638:1.

4. Kerberos

In Greek mythology, there was the country of Kerberos where no one could be trusted. The country was named after a three-headed dog that guarded the gate of purgatory. Greek legend states that in the land of Kerberos, if you desired to deliver any package to anyone, you and the package would be exposed to evil monsters and goblins that could take your shape and do malicious things in your name. Therefore, no one in the land of Kerberos could be trusted. Ironically, in the Information Age, the land of Kerberos has become reality, where evil users like Eve can take your identification and use it to perform malicious activities. Thanks to the founders of the Kerberos Authentication System invented at MIT, a tool is available that uses an authentication process similar in form to the three-headed dog Kerberos, as shown in Figure 10. For more information on the Legend of Kerberos, please visit: http://www.tamacom.com/~shigio/legend/kerberos.html.



1. as_req: c, tgs, $time_{exp}$, n
2. as_rep: $\{K_{c,tgs}, tgs, time_{exp}, n, ...\}K_c$, $\{T_{c,tgs}\}K_{tgs}$
3. tgs_req: $\{ts, ...\}K_{c,tgs}$ $\{T_{c,tgs}\}K_{tgs}$, v, $time_{exp}$, n
4. tgs_rep: $\{K_{c,v}, v, time_{exp}, n, ...\}K_{c,tgs}$, $\{T_{c,v}\}K_v$
5. ap_req: $\{ts, ck, K_{subsession}, ...\}K_{c,v}$ $\{T_{c,v}\}K_v$
6. ap_rep: $\{ts\}K_{c,v}$ (optional)

Figure 10. Simplified Kerberos authentication protocol (From:http://gost.isi.edu/publications/kerberos-neuman-tso.html)

The Kerberos application, invented at MIT, is a trusted third-party protocol that handles user authentication. Using Figure 10 as a visual reference for information flow, if Alice (who is represented by 'C' for client) wants to talk to Bob (who is represented by 'V' for verifier) on a Kerberos Authentication system, Alice must first register her required information on the Kerberos Server (which is represented by 'AS' for Authentication Server) that includes a secret shared only between Alice and the Kerberos server. Once registered, when Alice wants to talk to Bob, she must first authenticate herself to the

Kerberos server by logging onto the network. During the authentication process, the workstation that Alice is using to log onto the network sends Alice's identification to the Kerberos server. The Kerberos server responds by sending a session key that will later be shared with the Ticket Granting Server (TGS) and a ticket, both encrypted with the secret that Alice shares with Kerberos server. If Alice's workstation can successfully decrypt the session key and ticket, the workstation will continue the login process because Alice has successfully authenticated herself.

Once Alice has successfully logged onto the network, she then requests to talk to Bob via the TGS by sending the ticket that the Kerberos server provided her. The ticket contains a copy of the session key provided to Alice and Alice's identity encrypted with the secret that only the Kerberos server and TGS share. Once the TGS decrypts the ticket and verifies that Alice's identification is bound to the session key, the TGS knows that Alice is a trusted party because the Kerberos server (third-party), which the TGS identifies as a trusted user, provided Alice with the ticket encrypted with a secret that is shared between the TGS and Kerberos server only. The TGS then responds to Alice by sending another session key to use with Bob and another ticket to send to Bob. The session key and ticket that the TGS sends to Alice is encrypted with the session key shared between Alice and the TGS. Furthermore, the session key that Alice and the TGS share can be used by Alice to request additional session keys and tickets to use with other entities on the network.

Once Alice receives the new session key to share with Bob and the ticket to send to Bob encrypted with the session key shared with the TGS and Alice, she decrypts the file, holds onto the session key, and sends Bob the ticket. The ticket Alice sends to Bob is similar to the ticket Alice first sent to the TGS. It contains her identification and a copy of the session key that the TGS sent Alice, both encrypted with the session key that Bob and the TGS share. Bob established his session key with the TGS the same way Alice established hers using the Kerberos server. Once Bob decrypts the ticket and verifies that Alice's identification is bound to the session key, secure communication can begin using secret-key cryptography. If Bob were a fileserver for example, the TGS would also verify Alice's access rights to the file to which she is requesting access and include these privileges along with the name of the file Alice wishes to access for Bob (the fileserver) to verify prior to giving Alice read and/or write access. (Pfleeger, C. P. et al, 2003)

The benefit of this authentication system is that authentication at every step of the process is successfully completed without sending any encrypted or unencrypted passwords over the network where attackers could capture Alice's identity and perform an impersonation attack. Furthermore, since Kerberos uses symmetric cryptography, Alice's identity will be safe from any new attackers that may appear in a quantum-computing era. Microsoft has already adopted Kerberos version 5 as its Windows Sever 2008 client/server domain logon authentication. Kerberos also supports asymmetric cryptography, but this feature is

not covered in this thesis since the algorithm that is used for public-key cryptology is RSA (RFC 4556), and is vulnerable to quantum computing.

V. EXPERIMENTATAL METHODOLOGY AND IMPLEMENTATION

A. HASH-BASED CRYPTOGRAPHY

1. Hash-Based Cryptography Background

Our first task was to implement hash-based cryptography in the C programming language. We followed the description of hash-based cryptography in the 2010 Springer book, *Post-Quantum Cryptography*, edited by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. For our hash function, we used the MD5 implementation by Ronald Rivest at MIT (http://people.csail.mit.edu/rivest/Md5.c)

Post-Quantum Cryptography describes a hash-based public-key signature system based on a standard cryptographic hash function *H* with a digest of length *2b* bits. The signer's public key consists of *4b* strings $y_1[0]$, $y_1[1]$, $y_2[0]$, $y_2[1]$, …, $y_{2b}[0]$, $y_{2b}[1]$ of length *2b* bits. The signer's secret key consists of *4b* random strings $x_1[0]$, $x_1[1]$, $x_2[0]$, $x_2[1]$, …, $x_{2b}[0]$, $x_{2b}[1]$ of length *2b* bits. The signer generates the public key by computing $y_1[0] = H(x_1[0])$, $y_1[1] = H(x_1[1])$, $y_2[0] = H(x_2[0])$, $y_2[1] = H(x_2[1])$, …, $y_{2b}[0] = H(x_{2b}[0])$, $y_{2b}[1] = H(x_{2b}[1])$.

A message *m* is signed by computing *y = H(r, m)*, where *r* is a random string. The signer then sends the signature, which consists of *r* followed by $x_1[h_1]$, …, $x_{2b}[h_{2b}]$. The unused *x* values are discarded, and no further messages may be signed. This scheme is the Lamport-Diffie one-time signature system [W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, Vol. 22, No. 6, November 1976, pages 644-654]

2. Hash-Based Cryptography Implementation

Our implementation consists of two separate programs. The first program (separate1.c) is the signature generation process, and the second program (separate2.c) is the signature verification process. Both involve modifying Rivest's Md5.c program as follows (please note that for the sake of brevity, only the modified functions are shown):

```
/********** Helper Functions ***********/

static char *MDString2 (inString, outString)
char *inString;
char *outString;
{
  int i, j, k;
  MD5_CTX mdContext;
  unsigned int len = strlen (inString);

  MD5Init (&mdContext);
  MD5Update (&mdContext, inString, len);
  MD5Final (&mdContext);

  k = 0;
  for (i = 0; i < 16; i++) {
    for (j = 0; j < 8; j++) {
      if ((mdContext.digest[i] >> (7-j)) & 1 == 1)
        outString[k] = '1';
      else
        outString[k] = '0';
```

```
      k++;
    }
  }
  outString[k] = '\0';
}

void replaceCR(char *buf, int size)
{
  int i;
  for (i = 0; i < size; i++)
    if (buf[i] == '\n')
      buf[i] = '\0';
}

void clear(char *buf, int size)
{
  int i;
  for (i = 0; i < size; i++)
    buf[i] = '\0';
}

#define B 64

int main (argc, argv)
int argc;
char *argv[];
{
  int i, j;
  int b = B;
```

```
char **y0, **y1, **x0, **x1;
unsigned int r, bit;
char R[32+1], message[256], concat[512], concat2[512];
char buf[2 * B + 1], buf2[2 * B + 1];
char buf3[2 * B + 1], output[2 * B + 1];

// Clear out buffers before using
clear(R, 33);
clear(message, 256);
clear(concat, 512);
clear(concat2, 512);
clear(buf, 2 * B + 1);
clear(buf2, 2 * B + 1);
clear(buf3, 2 * B + 1);
clear(output, 2 * B + 1);

// Generate random number r
srand(time(0));
r = (unsigned int)random();

// Convert r into a string
clear(R, 33);
for (i = 0; i < 32; i++) {
  bit = (r >> i) & 0x1;
  R[i] = '0' + bit;
}
printf("%s\n", R);

// Read user-generated message to be signed
```

```
clear(message, 256);
fgets(message, 256, stdin);
replaceCR(message, 256);
printf(“%s\n”, message);

// Concatenate the two strings: message and R
clear(concat, 512);
clear(concat2, 512);
strcpy(concat,R);
strcat(concat,message);

// Compute the MD5 hash of the concatenation
MDString2(concat,concat2);

clear(output, 2 * b + 1);
strncpy(output,concat2,2 * b);
output[2 * b] = '\0';

// Allocate space for the private key
x0 = (char **)malloc(2 * b * sizeof(char *));
if (!x0) {
  fprintf(stderr, "malloc returned null!");
  return -1;
}
x1 = (char **)malloc(2 * b * sizeof(char *));
if (!x1) {
  fprintf(stderr, "malloc returned null!");
  return -1;
}
```

```
// Generate the private key
for (i = 0; i < 2 * b; i ++) {
  x0[i] = (char *)malloc((2 * b + 1)* sizeof(char));
  if (!x0[i]) {
    fprintf(stderr, "malloc returned null!");
    return -1;
  }
  for (j = 0; j < 2 * b; j++) {
    x0[i][j] = (((unsigned int)random()) % 2) + '0';
  }
  x0[i][j] = '\0';
}

for (i = 0; i < 2 * b; i ++) {
  x1[i] = (char *)malloc((2 * b + 1) * sizeof(char));
  if (!x1[i]) {
    fprintf(stderr, "malloc returned null!");
    return -1;
  }
  for (j = 0; j < 2 * b; j++) {
    x1[i][j] = (((unsigned int)random()) % 2) + '0';
  }
  x1[i][j] = '\0';
}

// Allocate space for the public key
y0 = (char **)malloc(2 * b * sizeof(char *));
if (!y0) {
```

```
  fprintf(stderr, "malloc returned null!");
  return -1;
}
y1 = (char **)malloc(2 * b * sizeof(char *));
if (!y1) {
  fprintf(stderr, "malloc returned null!");
  return -1;
}

// Generate the public key
for (i = 0; i < 2 * b; i ++) {
  y0[i] = (char *)malloc((2 * b + 1) * sizeof(char));
  if (!y0[i]) {
    fprintf(stderr, "malloc returned null!");
    return -1;
  }
  clear(buf, 2 * b + 1);
  clear(buf2, 2 * b + 1);
  strncpy(buf,x0[i],2 * b);
  MDString2(buf,buf2);
  clear(y0[i], 2 * b + 1);
  strncpy(y0[i],buf2, 2 * b);
  y0[i][2 * b] = '\0';
  printf("%s\n", y0[i]);
}
for (i = 0; i < 2 * b; i ++) {
  y1[i] = (char *)malloc((2 * b + 1) * sizeof(char));
  if (!y1[i]) {
    fprintf(stderr, "malloc returned null!");
```

```
      return -1;
    }
    clear(buf, 2 * b + 1);
    clear(buf2, 2 * b + 1);
    strncpy(buf,x1[i],2 * b);
    MDString2(buf,buf2);
    clear(y1[i], 2 * b + 1);
    strncpy(y1[i],buf2,2 * b);
    y1[i][2 * b] = '\0';
    printf("%s\n", y1[i]);
  }

  // Signature generation
  for (i = 0; i < 2 * b; i++) {
    if (output[i] == '0') {
      printf("%s\n",x0[i]);
    } else if (output[i] == '1') {
      printf("%s\n",x1[i]);
    } else {
      fprintf(stderr, "fatal error\n");
      return;
    }
  }
  fprintf(stderr, "Signature generation suceeded\n");
  return 0;
}
```

Signature verification (separate2.c) is accomplished by modifying Md5.c as follows (note that for the sake of

brevity, only the modified functions are shown, and the helper functions MDString2(), replaceCR(), and clear() already shown above are not shown here):

```
#define B 64

int main (argc, argv)
int argc;
char *argv[];
{
  int i, j;
  int b = B;
  char **y0, **y1, **x0, **x1;;
  unsigned int r, bit;
  char R[256], message[256], concat[512], concat2[512];
  char buf[2 * B + 2], buf2[2 * B + 2];
  char buf3[2 * B + 2], buf4[256], output[2 * B + 2];

  // Clear out buffers before using
  clear(R, 256); clear(message, 256); clear(concat, 512);
  clear(concat2, 512); clear(buf, 2 * B + 2);
  clear(buf2, 2 * B + 2); clear(buf3, 2 * B + 2);
  clear(buf4, 256); clear(output, 2 * B + 2);

  // Read in random string R
  clear(R, 256);
  fgets(R, 256, stdin);
  replaceCR(R, 256);
```

```
 // Read in the message
 clear(message, 256);
 fgets(message, 256, stdin);
 replaceCR(message, 256);
 clear(concat, 512);
 clear(concat2, 512);
 strcpy(concat,R);
 strcat(concat,message);
 MDString2(concat,concat2);
 clear(output, 2 * B + 2);
 strncpy(output,concat2,2*b);
 replaceCR(output, 2 * B + 2);

 // Allocate space for the public key
 y0 = (char **)malloc(2 * b * sizeof(char *));
 if (!y0) {
   fprintf(stderr, "malloc returned null!");
   return -1;
 }
 y1 = (char **)malloc(2 * b * sizeof(char *));
 if (!y1) {
   fprintf(stderr, "malloc returned null!");
   return -1;
 }

// Read in the public key
 for (i = 0; i < 2 * b; i++) {
   y0[i] = (char *)malloc((2 * b + 2) * sizeof(char));
   if (!y0[i]) {
```

```
    fprintf(stderr, "malloc returned null!");
    return -1;
  }
  clear(y0[i], 2 * b + 2);
  fgets(y0[i], 2 * b + 2, stdin);
  replaceCR(y0[i], 2 * b + 2);
}

for (i = 0; i < 2 * b; i++) {
  y1[i] = (char *)malloc((2 * b + 2) * sizeof(char));
  if (!y1[i]) {
    fprintf(stderr, "malloc returned null!");
    return -1;
  }
  fgets(y1[i], 2 * b + 2, stdin);
  replaceCR(y1[i], 2 * b + 2);
}

// Signature verification
for (i = 0; i < 2 * b; i++) {
  if (output[i] == '0') {
    clear(buf, 2 * b + 2);
    // Read in part of the signature
    fgets(buf, 2 * b + 2, stdin);
    replaceCR(buf, 2 * b + 2);
    // Is H(buf) the same as y0[i]?
    MDString2(buf,buf2);
    clear(buf3, 2 * b + 2);
    for (j = 0; j < 2 * b; j++) {
```

```
        buf3[j] = y0[i][j];
      }
      if (strncmp(buf2, buf3, 2 * b) != 0) {
        fprintf(stderr, "Signature verification failed\n");
        return -1;
      }
    } else if (output[i] == '1') {
      clear(buf, 2 * b + 2);
      // Read in part of the signature
      fgets(buf, 2 * b + 2, stdin);
      replaceCR(buf, 2 * b + 2);
      // Is H(buf) the same as y1[i]?
      MDString2(buf,buf2);
      clear(buf3, 2 * b + 2);
      for (j = 0; j < 2 * b; j++) {
        buf3[j] = y1[i][j];
      }
      if (strncmp(buf2, buf3, 2 * b) != 0) {
        fprintf(stderr, "Signature verification failed\n");
        return -1;
      }
    } else {
      fprintf(stderr, "fatal error\n");
      return;
    }
  }
  fprintf(stderr,"Signature verification succeeded\n");
  return 0;
}
```

The user follows the following steps to compile the above code:

```
% gcc –o separate1 separate1.c
% gcc –o separate2 separate2.c
```

Next, the user generates the signature:

```
% echo “Post-quantum cryptography is fun.” > message
% ./separate1 < message > signature
```

The user should see the following output:

```
Sender: Signature generation succeeded
```

Next, the user verifies the signature:

```
% ./separate2 < signature
```

The user should see the following output:

```
Signature verification succeeded
```

The file signature should be similar to the following:

```
11100110101000101101000111010110
Post-quantum cryptography is fun.
```

```
000101010010101011110000000110100010101100101111010111110010
111001011110000101111100101000010101011110111001110000001100
1100100010
011111000000110111100101101100001110111001100101010101010011
000100001101001100110111010111000011111010100001101010111100
0000100110
000111010010010000010100000101011101110010101011100100111100
011110110010011100001101000101111111111000000100101111101101
1110001010
001000101110101110000100100000010001010011111100111111111101
001111100000010110100101101110100100010000110001011010110100
1111010100
111101111100000100011101010110001111101011001111010010100000
000100000100101110000000101110001111101001100010001011010010
0100111110
101111011101110010000111101110111110000000000111100010110110
011000001100011010100111110110111111010111000011110110100100
1000110000
…
```

B. MCELIECE CRPTOSYSTEM

1. McEliece Cryptosystem Background

We found an implementation of a variant of the McEliece code-based cryptosystem implemented by Bhaskar Biswas and Nicolas Sendrier of the French National Institute for Research in Computer Science and Control (Institut national de recherché en informatique et en automatique, INRIA) in Rocquencourt, France. The source code is distributed as part of the SUPERCOP toolkit

developed by the VAMPIRE lab for measuring the performance of cryptographic software [http://bench.cr.yp.to/supercop.html]. SUPERCOP stands for System for Unified Performance Evaluation Related to Cryptographic Operations and Primitives. VAMPIRE stands for Virtual Applications and Implementations Research Lab, the third lab of ECRYPT, the European Network of Excellence in Cryptology II.

2. McEliece Cryptosystem Implementation

SUPERCOP measures a variety of cryptographic primitives. Anyone can contribute computer time to this benchmarking effort by downloading, unpacking, and running SUPERCOP on a Unix computer:

```
% wget http://hyperelliptic.org/ebats/supercop-20120316.tar.bz2
% bunzip2 < supercop-20120316.tar.bz2 | tar -xf -
% cd supercop-20120316
% nohup sh do &
```

The do script compiles the source code of the cryptographic software and generates a file, which the user posts to the web and then sends the URL to a mailing list.

The code compiles and runs on our system, but the comments, variable names, and function names are all in French.

Data on the performance of this cryptosystem on a variety of machines is available at http://bench.cr.yp.to/results-encrypt.html.

C. NTRUENCRYPT PUBLIC-KEY CRYPTO SYSTEM

1. NTRUEncrypt Public-Key Crypto System Background

An open-source implementation of NTRU (http://tbuktu.github.com/ntru/) is available. NTRU is an example of lattice-based cryptography. It has both a public-key encryption scheme (NTRUEncrypt) and a digital signature scheme (NTRUSign).

We were able to download and compile the source code for NTRU. We were also able to write our own programs that use NTRU in order to encrypt files, decrypt files, sign files, and verify signatures.

2. NTRUEncrypt Public-Key Crypto System Implementation

First, we downloaded the NTRU source code and unpacked it on a Unix machine. Then, we navigated to the demo folder:

```
% jar xvf ntru-1.0-src.jar
% cd src/main/java/net/sf/ntru/demo
```

This folder contains an example program that demonstrates both encryption and digital signature using NTRU. We modified this SimpleExample.java program to open the plaintext as a file and store the ciphertext as a file. In addition, our modified program saves the public and private keys to a file. Without our modifications, the example program generates an EncryptionKeyPair, encrypts a hard-coded string with the public key, decrypts the ciphertext (stored in a byte array that is a local variable

of the function), and then prints the decrypted string. With our modifications, it is possible to transfer the ciphertext to a different machine for decryption, which we successfully tested. The following is the code of Encrypt.java, our first of two encryption programs:

```
package net.sf.ntru.demo;
import net.sf.ntru.encrypt.EncryptionPrivateKey;
import net.sf.ntru.encrypt.EncryptionPublicKey;
import net.sf.ntru.encrypt.EncryptionKeyPair;
import net.sf.ntru.encrypt.EncryptionParameters;
import net.sf.ntru.encrypt.NtruEncrypt;
import net.sf.ntru.sign.NtruSign;
import net.sf.ntru.sign.SignatureKeyPair;
import net.sf.ntru.sign.SignatureParameters;
import java.io.*;

public class Encrypt {

    public static void main(String[] args) {
        encrypt();
    }

    private static void encrypt() {
        // create an instance of NtruEncrypt
        NtruEncrypt ntru = new NtruEncrypt(
                                EncryptionParameters
                                .APR2011_439_FAST);
        // create an encryption key pair
```

```
EncryptionKeyPair kp = ntru.generateKeyPair();
byte[] enc = {};
byte[] buf = new byte[64];
File f;
FileOutputStream fos;
FileInputStream fis;
try {
    // Load the plaintext from disk
    f = new File("plaintext");
    fis = new FileInputStream(f);
    fis.read(buf);
    fis.close();
    // encrypt the message with the public key
    enc = ntru.encrypt(buf, kp.getPublic());

    // Store the public key to disk
    f = new File("public_key");
    fos = new FileOutputStream(f);
    kp.getPublic().writeTo(fos);
    fos.close();
    // Store the private key to disk
    f = new File("private_key");
    fos = new FileOutputStream(f);
    kp.getPrivate().writeTo(fos);
    fos.close();
    // Store the ciphertext to disk
    f = new File("ciphertext");
    fos = new FileOutputStream(f);
    fos.write(enc);
```

```
            fos.close();
        } catch (Exception e) {
            System.err.println("Exception! " + e);
        }
    }
}
```

To compile and run this program, the user creates a plaintext file called plaintext and then types the following commands:

```
% javac -classpath ../../../.. Encrypt.java
% java -classpath ../../../.. net.sf.ntru.demo.Encrypt
```

Next, we created another encryption program Encrypt2.java that does not create a key pair but instead uses an existing key pair stored in a file:

```
package net.sf.ntru.demo;
import net.sf.ntru.encrypt.EncryptionPrivateKey;
import net.sf.ntru.encrypt.EncryptionPublicKey;
import net.sf.ntru.encrypt.EncryptionKeyPair;
import net.sf.ntru.encrypt.EncryptionParameters;
import net.sf.ntru.encrypt.NtruEncrypt;
import net.sf.ntru.sign.NtruSign;
import net.sf.ntru.sign.SignatureKeyPair;
import net.sf.ntru.sign.SignatureParameters;
import java.io.*;
```

```
public class Encrypt2 {

    public static void main(String[] args) {
        encrypt();
    }

    private static void encrypt() {
        // create an instance of NtruEncrypt
        NtruEncrypt ntru = new NtruEncrypt(
                                EncryptionParameters
                                .APR2011_439_FAST);

        byte[] enc = {};
        byte[] buf = new byte[64];
        File f;
        FileOutputStream fos;
        FileInputStream fis;
        EncryptionPrivateKey pri;
        EncryptionPublicKey pub;
        try {
            // Load the public key from disk
            f = new File("public_key");
            fis = new FileInputStream(f);
            pub = new EncryptionPublicKey(fis,
                      EncryptionParameters
                      .APR2011_439_FAST);
            fis.close();
            // Load the plaintext from disk
            f = new File("plaintext");
```

```
            fis = new FileInputStream(f);
            fis.read(buf);
            fis.close();
            // Encrypt the message with the public key
            enc = ntru.encrypt(buf, pub);
            // Store the ciphertext to a file
            f = new File("ciphertext");
            fos = new FileOutputStream(f);
            fos.write(enc);
            fos.close();
        } catch (Exception e) {
            System.err.println("Exception! " + e);
        }
    }
}
```

To compile and run this program, the user must already have an existing public_key file. The user creates a plaintext file called plaintext and then types the following commands:

```
% javac -classpath ../../../.. Encrypt2.java
% java -classpath ../../../.. net.sf.ntru.demo.Encrypt2
```

Next, we created a decryption program Decrypt.java that requires existing public_key, private_key, and ciphertext files:

```
package net.sf.ntru.demo;
```

```
import net.sf.ntru.encrypt.EncryptionPrivateKey;
import net.sf.ntru.encrypt.EncryptionPublicKey;
import net.sf.ntru.encrypt.EncryptionKeyPair;
import net.sf.ntru.encrypt.EncryptionParameters;
import net.sf.ntru.encrypt.NtruEncrypt;
import net.sf.ntru.sign.NtruSign;
import net.sf.ntru.sign.SignatureKeyPair;
import net.sf.ntru.sign.SignatureParameters;
import java.io.*;

public class Decrypt {

    public static void main(String[] args) {
        decrypt();
    }

    private static void decrypt() {
        // create an instance of NtruEncrypt
        NtruEncrypt ntru = new NtruEncrypt(
                                EncryptionParameters
                                .APR2011_439_FAST);
        byte[] dec = {};
        byte[] buf = new byte[1024];
        File f;
        FileOutputStream fos;
        FileInputStream fis;
        EncryptionPrivateKey pri;
        EncryptionPublicKey pub;
        EncryptionKeyPair pair;
```

```
        try {
            // Load the public key from disk
            f = new File("public_key");
            fis = new FileInputStream(f);
            pub = new EncryptionPublicKey(fis,
                      EncryptionParameters
                      .APR2011_439_FAST);
            fis.close();
            // Load the private key from disk
            f = new File("private_key");
            fis = new FileInputStream(f);
            pri = new EncryptionPrivateKey(fis,
                      EncryptionParameters
                      .APR2011_439_FAST);
            pair = new EncryptionKeyPair(pri, pub);
            // Load the ciphertext from disk
            f = new File("ciphertext");
            fis = new FileInputStream(f);
            fis.read(buf);
            System.out.println(buf);
            dec = ntru.decrypt(buf, pair);
        } catch (Exception e) {
            System.err.println("Exception! " + e);
        }
        // Print the decrypted message
        System.out.println("Message:  " + new String(dec));
    }
}
```

To compile and run this program, the user must already have existing ciphertext, public_key and private_key files. The user types the following commands:

```
% javac -classpath ../../../.. Decrypt.java
% java -classpath ../../../.. net.sf.ntru.demo.Decrypt
```

Next, we created a program Sign.java to sign a file (modified from the example program to read the message from the file message and to write the public_signing_key, private_signing_key, and signature to disk):

```
package net.sf.ntru.demo;
import net.sf.ntru.encrypt.EncryptionPrivateKey;
import net.sf.ntru.encrypt.EncryptionPublicKey;
import net.sf.ntru.encrypt.EncryptionKeyPair;
import net.sf.ntru.encrypt.EncryptionParameters;
import net.sf.ntru.encrypt.NtruEncrypt;
import net.sf.ntru.sign.NtruSign;
import net.sf.ntru.sign.SignatureKeyPair;
import net.sf.ntru.sign.SignaturePublicKey;
import net.sf.ntru.sign.SignaturePrivateKey;
import net.sf.ntru.sign.SignatureParameters;
import java.io.*;

public class Sign {

    public static void main(String[] args) {
        sign();
```

```
}

private static void sign() {
    // create an instance of NtruSign
    NtruSign ntru = new NtruSign(
                        SignatureParameters.TEST157);
    // create an signature key pair
    SignatureKeyPair kp = ntru.generateKeyPair();
    byte[] buf = new byte[64];
    File f;
    FileOutputStream fos;
    FileInputStream fis;
    try {
        // Read the message from disk
        f = new File("message");
        fis = new FileInputStream(f);
        fis.read(buf);
        fis.close();
        // Sign the message with the private key
        byte[] sig = ntru.sign(buf, kp);
        // Write the public signing key to disk
        f = new File("public_signing_key");
        fos = new FileOutputStream(f);
        kp.getPublic().writeTo(fos);
        fos.close();
        // Write the private signing key to disk
        f = new File("private_signing_key");
        fos = new FileOutputStream(f);
        kp.getPrivate().writeTo(fos);
```

```
            fos.close();
            // Write the signature to disk
            f = new File("signature");
            fos = new FileOutputStream(f);
            fos.write(sig);
            fos.close();
        } catch (Exception e) {
            System.err.println("Exception! " + e);
        }
    }
}
```

To compile and run this program, the user creates a message file and then types the following commands:

```
% javac -classpath ../../../.. Sign.java
% java -classpath ../../../.. net.sf.ntru.demo.Sign
```

Next, we created Sign2.java, a signature program that uses an existing public_signing_key and private_signing_key:

```
package net.sf.ntru.demo;
import net.sf.ntru.encrypt.EncryptionPrivateKey;
import net.sf.ntru.encrypt.EncryptionPublicKey;
import net.sf.ntru.encrypt.EncryptionKeyPair;
import net.sf.ntru.encrypt.EncryptionParameters;
import net.sf.ntru.encrypt.NtruEncrypt;
import net.sf.ntru.sign.NtruSign;
```

```
import net.sf.ntru.sign.SignatureKeyPair;
import net.sf.ntru.sign.SignaturePublicKey;
import net.sf.ntru.sign.SignaturePrivateKey;
import net.sf.ntru.sign.SignatureParameters;
import java.io.*;

public class Sign2 {

    public static void main(String[] args) {
        sign();
    }

    private static void sign() {
        // Create an instance of NtruSign
        NtruSign ntru = new NtruSign(
                            SignatureParameters.TEST157);
        SignatureKeyPair kp;
        byte[] buf = new byte[64];
        byte[] sig = {};
        File f;
        FileOutputStream fos;
        FileInputStream fis;
        SignaturePrivateKey pri;
        SignaturePublicKey pub;
        try {
            // Read public signing key from disk
            f = new File("public_signing_key");
            fis = new FileInputStream(f);
            pub = new SignaturePublicKey(fis,
```

```
                    SignatureParameters.TEST157);
            fis.close();
            // Read private signing key from disk
            f = new File("private_signing_key");
            fis = new FileInputStream(f);
            pri = new SignaturePrivateKey(fis,
                    SignatureParameters.TEST157);
            fis.close();
            kp = new SignatureKeyPair(pri, pub);
            // Read message from disk
            f = new File("message");
            fis = new FileInputStream(f);
            fis.read(buf);
            fis.close();
            // Sign the message with the key pair
            sig = ntru.sign(buf, kp);
            // Write signature to disk
            f = new File("signature");
            fos = new FileOutputStream(f);
            fos.write(sig);
            fos.close();
        } catch (Exception e) {
            System.err.println("Exception! " + e);
        }
    }
}
```

To compile and run this program, the user must have an existing public_signing_key and private_signing_key. The

user creates a message file and then types the following commands:

```
% javac -classpath ../../../.. Sign2.java
% java -classpath ../../../.. net.sf.ntru.demo.Sign2
```

Next, we created a program Verify.java to verify the signature. This program requires the following files: public_signing_key, message, and signature:

```
package net.sf.ntru.demo;

import net.sf.ntru.encrypt.EncryptionPrivateKey;
import net.sf.ntru.encrypt.EncryptionPublicKey;
import net.sf.ntru.encrypt.EncryptionKeyPair;
import net.sf.ntru.encrypt.EncryptionParameters;
import net.sf.ntru.encrypt.NtruEncrypt;
import net.sf.ntru.sign.NtruSign;
import net.sf.ntru.sign.SignatureKeyPair;
import net.sf.ntru.sign.SignaturePublicKey;
import net.sf.ntru.sign.SignaturePrivateKey;
import net.sf.ntru.sign.SignatureParameters;
import java.io.*;

public class Verify {

    public static void main(String[] args) {
        verify();
    }
```

```
private static void verify() {
    // Create an instance of NtruSign.
    NtruSign ntru = new NtruSign(
                        SignatureParameters.TEST157);
    byte[] buf = new byte[64];
    byte[] sig = new byte[1024];
    File f;
    FileInputStream fis;
    SignaturePrivateKey pri;
    SignaturePublicKey pub;
    try {
        // Read the public signing key from disk
        f = new File("public_signing_key");
        fis = new FileInputStream(f);
        pub = new SignaturePublicKey(fis,
                  SignatureParameters.TEST157);
        fis.close();
        // Read the message from disk
        f = new File("message");
        fis = new FileInputStream(f);
        fis.read(buf);
        fis.close();
        // Read the signature from disk
        f = new File("signature");
        fis = new FileInputStream(f);
        int nbytes = fis.read(sig);
        fis.close();
        byte[] sig2 = new byte[nbytes];
```

```
            for (int i = 0; i < nbytes; i++)
                sig2[i] = sig[i];
            // Verify the signature
            boolean valid = ntru.verify(buf, sig2, pub);
            System.out.println("Valid? " + valid);
        } catch (Exception e) {
            System.err.println("Exception! " + e);
        }
    }
}
```

To compile and run this program, the user must have an existing public_signing_key, message, and signature. The user types the following commands:

```
% javac -classpath ../../../.. Verify.java
% java -classpath ../../../.. net.sf.ntru.demo.Verify
```

THIS PAGE INTENTIONALLY LEFT BLANK

VI. EXPERIMENTAL RESULTS

A. HASH-BASED CRYPTOLOGY

After implementing the hash-based cryptography scheme in C, we evaluated it both on a Mac OS X system as well as in Ubuntu. We established two Ubuntu Virtual Machine (VM) environments, one for the sender, and another for the receiver. We validated the correct operation of the signature generation process in the sender's environment, transferred the signature to the receiver's environment, and validated the correct operation of the signature verification process in the receiver's environment. We then made a small change to the message to validate that the signature verification process failed.

We also attempted to generate the signature on the Mac OS X system and verify the signature on the Ubuntu system. However, signature verification failed on the Ubuntu system because of a subtle implementation issue with the MD5 code we downloaded from Ronald Rivest's website at MIT. We discovered that this MD5 code produces a different hash value on the Mac than in the Ubuntu VM. To fix this problem will require finding the source code of a cryptographic hash function that produces the same hash value for the same message on a Mac and in the Ubuntu environment.

Currently, only messages with a maximum length of 256 bytes are supported. Future work will involve making a small change to the code to support messages of arbitrary

size. This is a simple fix. The time required to sign a message consisting of 34 characters is less than one hundredth of a second.

B. NTRUENCRYPT PUBLIC-KEY CRYPTO SYSTEM

After implementing our modifications to the NTRU source code in Java, we evaluated the programs on both a Mac OS X system as well as in an Ubuntu VM environment. We established two Ubuntu environments, one for the sender, and another for the receiver. We validated the correct operation of the encryption program in the sender's environment. Then, we transferred the public_key, private_key, and ciphertext files to the receiver's environment and validated the correct operation of the decryption program.

Note that there are two encryption programs: one that generates a new encryption key pair and stores it to disk, and another that uses an existing encryption key pair, loading the public_key file from disk. We validated the correct operation of both encryption programs.

Also, we validated the correct operation of the signature generation process in the sender's environment. Then, we transferred the public_signing_key, message, and signature to the receiver's environment and validated the correct operation of the signature verification program.

Note that there are two signature generation programs: one that generates a new signing key pair and stores it to disk, and another that uses an existing signing key pair, loading it from disk. We validated the correct operation of both signature generation programs.

The time to encrypt a 37-byte message is less than one half of a second. We are currently limited to messages with a maximum length of 64 characters. A simple fix will allow messages of arbitrary length (below we describe a hybrid encryption scheme that already supports messages of any length). The time to decrypt the message is less than 0.3 seconds.

The time to sign a 37-byte message is approximately one second. We are currently limited to messages with a maximum length of 64 bytes. A simple fix will allow messages of arbitrary length. The time to verify the signature is approximately 0.3 seconds.

C. OPENSSL

Symmetric encryption is not vulnerable to quantum computers. To demonstrate how easy it is for anyone to use symmetric encryption, we show the following examples of using OpenSSL, which is installed on many systems, including Mac OS X and Ubuntu Linux.

To encrypt a file plain.txt, all one must do is type the following at the command prompt:

```
% openssl enc -aes-128-cbc -e -in plain.txt -out cipher.txt
-K bead1234 -iv feed4321
```

This will produce a file cipher.txt containing the ciphertext encrypted under key 0xbead1234 and initialization vector of 0xfeed4321 using the AES cipher with a key size of 128 bits and the cipher-block chaining

(CBC) mode of operation. It takes less than one hundredth of a second to encipher a file containing 37 characters.

To decrypt the file, the user types the following at the command prompt:

```
% openssl enc -aes-128-cbc -d -in cipher.txt -out decrypted.txt -K bead1234 -iv feed4321
```

Deciphering the 37-character file also takes less than one hundredth of a second. Note that the key and the initialization vector must be the same for encryption and decryption. Otherwise, the file will not decrypt properly. Also, the output file decrypted.txt has a different filename than the original plaintext file plain.txt so that it is possible to compare the two rather than overwriting the original plaintext file. When using a 128-bit key, the user may specify up to 128 hexadecimal digits of the key. In the above example, they key only has 8 hexadecimal digits, or 32 bits.

D. NTRU + OPENSSL

We now demonstrate a hybrid encryption protocol combining the quantum-resistant asymmetric cipher NTRU together with the quantum-resistant symmetric cipher AES implemented by OpenSSL.

1) First, we generate a 128-bit key. Suppose that this key is the hexadecimal value 25c16a7af74b53d421754fadc0f1b531. We create a text file plaintext containing these hexadecimal characters in ASCII format. We place this file in the sender’s directory. Note

that you may also encrypt the initialization vector iv if you wish as a separate step.

2) Next, we encrypt a file plain.txt containing the (long) message using AES as implemented by OpenSSL:

```
% openssl enc -aes-128-cbc -e -in plain.txt -out cipher.txt
-K 25c16a7af74b53d421754fadc0f1b531 -iv feed4321
```

3) Next, we run the NTRU encryption program on the sender’s machine. Either encryption program Encrypt.java or Encrypt2.java is acceptable depending on your requirements.

```
% cd src/main/java/net/sf/ntru/demo
% javac -classpath ../../../.. Encrypt.java
% java -classpath ../../../.. net.sf.ntru.demo.Encrypt
```

4) Next, we transfer the files public_key, private_key, ciphertext (the key encrypted with NTRU), and cipher.txt (the message encrypted with AES) to the receiver’s machine and run the NTRU decryption program on the receiver’s machine.

```
% javac -classpath ../../../.. Decrypt.java
% java -classpath ../../../.. net.sf.ntru.demo.Decrypt
```

5) We note the hexadecimal characters printed to the screen as a result of the decryption operation and use them as the decryption key as follows:

% openssl enc -aes-128-cbc -d -in cipher.txt -out decrypted.txt -K 25c16a7af74b53d421754fadc0f1b531 -iv feed4321

6) The file decrypted.txt on the receiver’s machine should be identical to the file plain.txt on the sender’s machine.

VII. QUALITATIVE MANAGEMENT ANALYSIS

A. BACKGROUND

1. Algorithms are Broken Eventually

This thesis has covered how large-scale, fault-tolerant quantum computers have the potential to render vulnerable algorithms like RSA, Elliptic Curve Cryptography, Diffie-Hellman, and others that use factoring of large numbers or computing discrete logarithms as the basis of their security. In general, ciphers are eventually broken over time, as shown by the trend for hash algorithms depicted in Figure 11. The growth of computing power as described by Moore's Law, new ways of harnessing computing power like daisy-chaining the processing power of multiple XBOX 360s, or new breakthroughs in mathematics all contribute to rendering vulnerable (i.e., breaking) algorithms considered strong today.

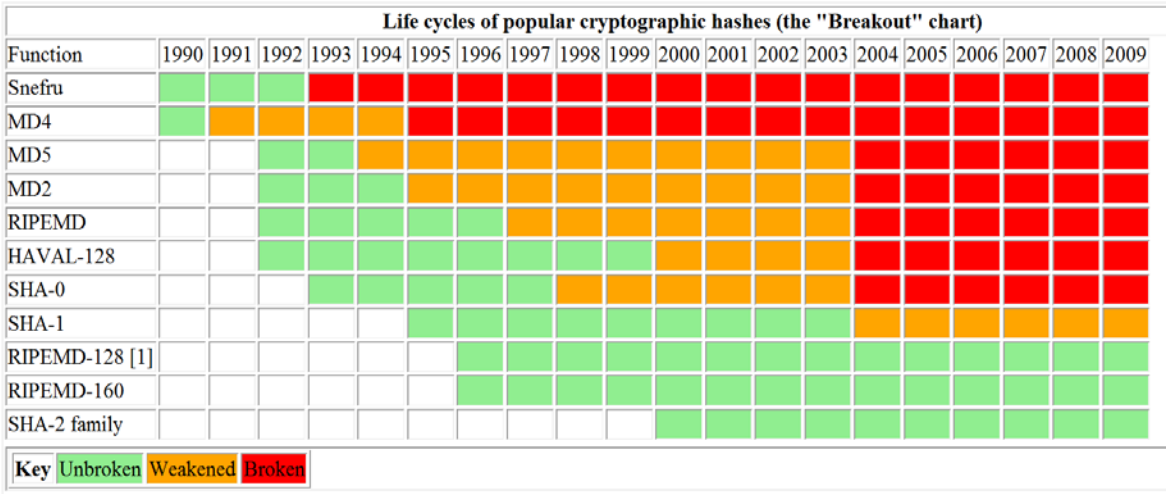


Figure 11. Life cycles of popular cryptographic hashes (From:http://valerieaurora.org/monkey.html)

It is the continuous life-cycle of algorithms being invented and broken that motivates the field of cryptology to continuously invent new algorithms that are hardened through peer-review. In security and cryptography, there are no silver-bullets today or in a quantum-computing era, and a defense-in-depth approach ranging from cipher design to cryptosystem implementation is essential whenever sensitive information is stored or distributed via an electronic medium.

2. Protection of the CIA Triad in a Quantum Era

For those who choose to store and transfer sensitive information via electronic means to have a complete system that protects our data with respect to all facets of the CIA Triad (Confidentiality, Integrity, and Availability) in a quantum-computing era, we must replace algorithms believed to be vulnerable with those believed to be quantum computing resistant to prepare for the emergence of large-scale, fault-tolerant quantum computers. Therefore, we

must have an algorithm to create digital fingerprints of our messages, and it is essential for our system to have a secret-key algorithm to encrypt and decrypt the bulk of our secure communication. We also need to be able to digitally sign our messages as we would hard-copy messages. Finally, we require a secure method of distributing symmetric session keys.

B. ASSESSMENT OF OUR QUANTUM COMPUTING READINESS

1. Introduction

The following figure depicts the authors' assessment of the world's quantum computing readiness.

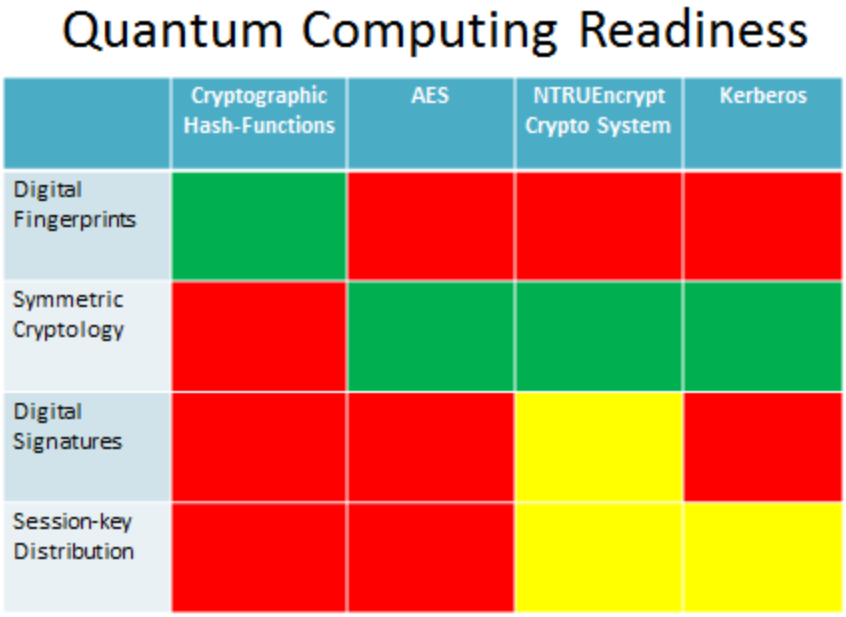


Figure 12. Quantum Computing Readiness

2. Digital Signatures

Modern cryptographic hash functions provide one method for creating digital signatures. Since a hash function must accept a variable-size input, output a fixed-length digest, be fast to compute, difficult to invert, and produce few collisions, these requirements of a secure hash algorithm also make hash functions quantum-resistant. Therefore, we are already capable of producing digital signatures for a quantum-computing era, and that is why the block is labeled green for digital fingerprints under Cryptographic Hash-Functions.

3. Symmetric (Secret-Key) Cryptology

Another algorithm already in wide use that will be quantum-resistant is the Advanced Encryption Standard (AES). This symmetric algorithm is cleared to protect the United States Government's most sensitive material when using AES key sizes of 192 bits or greater. There will be no change required to the symmetric cryptography system when quantum computers come into existence, and this is why the block is labeled green for symmetric cryptography under AES. The NTRUEncrypt and Kerberos columns are also labeled green for symmetric cryptography because both systems support the AES algorithm as part of their secret-key algorithm.

4. Digital Signatures

One viable and efficient system we have available today for creating digital signatures in a quantum-computing era is the NTRUEncrypt Cryptosystem. This block is labeled yellow because even though NTRU Encrypt is

capable of creating digital signatures and is even more efficient at digital signatures than algorithms currently in wide use, NTRU is not widely accepted yet, and RSA reigns as the most common algorithm used for digital signatures.

5. Session-Key Distribution

NTRUEncrypt and Kerberos both provide secure methods for session key distribution, but they are labeled yellow in the figure for two separate reasons. First, NTRU is able to distribute session keys in a public-key infrastructure just like the popular RSA algorithm does today. As discussed above, NTRUEncrypt is more efficient than RSA. The reason why NTRU is shown in the figure in yellow for session-key distribution is for the same reason provided for digital signatures: NTRUEncrypt is not yet widely accepted and has not been added to the list of possible public-key algorithms for many RFC standards as an alternative algorithm for public-key cryptography.

On the other hand, Kerberos could be considered widely accepted because Microsoft has been using Kerberos since Microsoft Server 2000, and with the latest version, Kerberos is an extremely secure system that assumes the network it operates in is untrusted and requires user authentication whenever the user is trying to perform any function outside of his or her workstation. The drawback to Kerberos, and the reason that it is labeled yellow in the figure, is that Kerberos authentication is recursive in nature. All users must be authenticated with a single server prior to being provided access to the ticket granting server, which then provides the user subsequent

access to other services like file and print servers. This authentication system would be fine for a closed and trusted intranet like multiple University intranets connected to one another, but this single point of authentication, and the fact that every user must physically register with a Kerberos server prior to logging onto a Kerberos network, makes a Kerberos system unsuitable for Internet scalability.

C. SCENARIO-BASED PREPARATION; IF QUANTUM COMPUTERS WERE INVENTED

1. Tomorrow

Yogi Berra once said that prediction is difficult, especially about the future. This statement is applicable to quantum computing: it is difficult to know when or if large-scale, fault-tolerant quantum computers will become available. Nevertheless, this section offers some speculation, a glimpse into the future, with educated guesses based on the analysis presented in this thesis. If quantum computers suddenly become available to attackers, no one may know this fact until after an attack has already occurred. Attackers/adversaries could be terrorist organizations or governments with the ability to fund the development of quantum computers. A successful program to develop a quantum computer would likely want to keep its success a secret in order to maximize the amount of eavesdropping. Disclosure of the successful development would weaken the effectiveness of the tool since countermeasures would likely be put into place upon disclosure.

Assuming that quantum computers have not been invented, and that the inventor(s) of the large-scale, fault-tolerant quantum computer would benefit (e.g., financially or in terms of status) from publicizing the invention of a quantum computer immediately, then there would be a large push for the rapid implementation of alternative public-key cryptosystems to replace the popular RSA algorithm. It is likely that leading Internet browser companies would work around the clock until their browsers supported the quantum-resistant alternatives, and most browsers would be able to support the alternatives in a short period of time. A large-scale boycott of the Internet is unlikely because most users are ignorant of basic computer security. The hasty implementation of changes could itself result in bugs, which would need to be addressed by software patches. While software changes can be made relatively easily (e.g., by downloading a patch), changes to hardware are more expensive. For example, ASIC implementations of cryptosystems would need to be redesigned and fabricated, which is extremely expensive and time-consuming. Programmable hardware (e.g., field-programmable gate arrays, or FPGAs) can be more easily updated with new cryptographic hardware designs, but engineering of the new designs must take place whether ASIC or FPGA is used.

Enumerating the various cryptosystems that use special-purpose hardware is beyond the scope of this thesis, but in general, organizations that use custom, dedicated hardware to implement cryptography do so in order to keep up with large volumes of data, i.e., high bandwidth/throughput. A company that uses custom,

dedicated hardware could either switch over to a software implementation temporarily as a coping mechanism and later to FPGA and finally to ASIC. A company that is already using FPGAs could reprogram the FPGAs with the new custom design when it is available. In the worst case, unless that company feels that costs of shutting the system down is more costly than keeping the system up and running in a potentially unsecure environment, all aspects of operation that depend on that piece of hardware will be down until a company can replace it. Even after replacement hardware is fabricated, the laws of supply and demand will cause prices to skyrocket, and the organizations that have the most money will pay top dollar to receive the first available replacement products, while other smaller companies would have to wait.

If quantum computers suddenly become available, this could have a serious impact on some organizations. Fortunately, a large institution like the Accredited Standards Committee X9 Incorporated, Financial Industry Standards, has taken heed to the warnings of quantum computers, has noted the efficiency of the NTRUEncrypt Public-Key Cryptosystem, and has already made the transition to NTRU from RSA. Therefore, it is encouraging to see steps being taken in a positive direction, but these are not enough. Even though our financial institutions are able to communicate securely among themselves using alternatives such as NTRU, ordinary users are unable to communicate securely with our financial institutions using alternatives such as NTRU. Therefore, our sensitive

financial data will be vulnerable in a quantum-computing era during transit between our Internet browsers and our banks' web servers unless changes are implemented.

2. A Year from Today

If quantum computers become available a year from today, we have time to manage the change without causing too much chaos for those organization that are predicted to be impacted the most. IT organizations that publish software and/or hardware implementations that perform digital signatures or support public-key infrastructures could begin the transition tomorrow. The software aspect of this problem can likely be remedied the fastest because the Internet can be used to publish the updates and patches as required. The hardware aspect of this problem is likely to require the most time since after a prototype has been created and tested, manufacturing plants must produce the new hardware. After the hardware has begun mass production, it must be shipped to the customer via means that are typically no faster than 24 hours. Then, once the product has arrived, it must be installed and tested. Assuming installation occurs with zero problems (one could easily argue that this assumption is unrealistic) installation of the hardware implementation takes longer than a software implementation, in general.

It is worthwhile to consider the question, "are IT companies at least working towards adding alternatives such as NTRUEncrypt to the list of possible implementations?" If not, we will more likely find ourselves in the first scenario, where quantum computers come into existence tomorrow, and we are totally unprepared for their arrival.

3. Beyond a Year, but Sometime in the Near Future

If quantum computers become available more than a year from now, but sometime in the near future, we could potentially benefit from all of the remedies described in the section where quantum computers were invented a year from today and more. In fact, this scenario would look very similar to the Y2K problem the world faced in the 1990s, the main difference being that the world knew that by 01/01/01, all systems must be made Y2K-compliant. Unfortunately, the quantum computing cryptology management problem does not know exactly when someone will invent fault-tolerant quantum computers. Assuming we have multiple years to prepare for the quantum-computing era (i.e., the quantum singularity), we could use methodologies learned from Y2K to transition into a quantum-computing era, such as the publishing of best practices. With the Accredited Standards Committee X9 Incorporated, Financial Industry Standards, being one of the first if not the first large industry to change over to alternatives such as NTRU, they could publish their lessons learned for others to follow. Organizations could designate a Quantum Computer Transition Coordinator who is a senior IT manager to assess the organization's vulnerability to quantum computers and the impact they will have, identify the optimal solution, and then take the time to refine the solution while providing scheduled updates to the senior executives.

Even the companies that have critical hardware implementations of algorithms that are predicted to be vulnerable to quantum computing could start soliciting IT hardware companies to fabricate an alternative. If

predictions indicated the lack of an available alternative hardware implementation within a year, the organization could consider installing a software backup running on more machines, or even leverage field-programmable gate-arrays (FPGAs) that are the hybrid of software and hardware implementations. FPGAs are faster than software for certain high-throughput applications, but they are not dedicated hardware and therefore are not as fast as pure hardware implementations (e.g., Application-Specific Integrated Circuits, or ASICs). Another benefit of FPGAs is that they are reprogrammable like software implementations, whereas dedicated hardware requires complete replacement, and this is more costly than updating software or FPGA implementations.

In conclusion, if we have a few years to manage the cryptography transition from RSA to NTRU, there would be minimal to no rush in the transition; we could learn from published best-practices or lessons-learned of how to best manage the transition; organizations could make full assessments of the vulnerabilities and what courses of action are best to take for each vulnerability; and finally, new systems could be tested while still having the strong and popular RSA algorithm on standby just in case the organization had a problem with the new system. Once quantum computers come into existence, RSA will not be available to use as training wheels.

D. ANALYSIS CONCLUSION

This thesis recommends that the rest of the industry follow the lead of the Accredited Standards Committee X9

Incorporated, Financial Industry Standards, and identify a suitable alternative cipher such as the NTRUEncrypt Cryptosystem as the primary algorithm for asymmetric cryptography to replace RSA if needed. Preparations should be made to facilitate a smooth transition if it becomes necessary. If the concern is too great that NTRU is a new algorithm, then this thesis at least recommends that it be added to the published standards as an alternative cipher implementation so that it is available to the industry in the event quantum computers abruptly come into existence.

VIII. CONCLUSION

A. HYPOTHESIS QUESTIONS REVIEWED

What if quantum computing reduces the time to defeat traditional ciphers from millions of years by today’s supercomputers to only seconds? What if we are already living in that era, and unfriendly forces have such technology?

How efficient are post-quantum ciphers proposed as alternatives to traditional ciphers like RSA and Elliptic Curve Cryptography (ECC)? Do these ciphers have enough bandwidth to meet today's cryptographic workloads? What is the performance impact of deploying an alternative cryptographic infrastructure based on post-quantum ciphers? Could available implementations be used as a basis for constructing a cryptographic software library that is a viable alternative to classical ciphers?

B. HYPOTHESIS

While alternative ciphers exist, available implementations do not satisfy all performance requirements of modern cryptographic workloads. A cryptographic infrastructure that allows for ciphers to be reconfigured dynamically will reduce the costs of switching cryptographic infrastructure quickly in response to the development of quantum computers.

C. HYPOTHESIS VALIDATION

This research validates the hypothesis that alternative ciphers exist, and implementations of some

quantum-resistant ciphers are available. This research validated the feasibility of developing an original implementation of a quantum-resistant cipher, specifically hash-based digital signature, based on a description of the algorithm in the post-quantum cryptography literature. This implementation was used to send a digitally signed message from the sender's machine to the receiver's machine, where the signed message was verified successfully. This thesis also validated the feasibility of adapting and customizing existing implementations of quantum-resistant ciphers, showing how to modify the source code of the NTRU cryptosystem to send an encrypted message from the sender's machine to the receiver's machine, where it was successfully decrypted. This thesis also showed how to modify the source code of NTRU to send a digitally signed message from the sender's machine to the receiver's machine, where it was successfully verified. This thesis also showed how to use NTRU and AES together as part of a hybrid encryption protocol. Specifically, NTRU was used as a quantum-resistant asymmetric cipher to exchange a symmetric session key. The quantum-resistant symmetric cipher AES, implemented by OpenSSL, was used to encrypt a long message. Future work is needed to ensure that implementations of quantum-resistant ciphers are free of implementation flaws and that the ciphers themselves (i.e., the algorithms) have been exposed to rigorous peer review to ensure that the algorithms are mathematically sound and provably secure.

The research shows that the phrase of the hypothesis that states "available implementations do not satisfy all performance requirements of modern cryptographic workloads"

is partially incorrect, at least with respect to one implementation of lattice-based cryptography. Specifically, existing studies of the NTRUEncrypt Public-Key Crypto System have demonstrated that NTRU outperforms RSA, today's leading public-key algorithm. According to the published literature, since NTRU is a new system, it has not yet gained industry acceptance.

Qualitative analysis shows the usefulness of a flexible, configurable design approach to cryptosystems and large-scale cryptographic infrastructure. Future work is needed to measure the costs of developing and deploying an alternative infrastructure that is more configurable than the existing infrastructure. While such an effort would be extremely costly, so too would be the impact of the sudden emergence of large-scale, fault-tolerant quantum computers requiring the shutting down of critical systems until software patches and even expensive and time-consuming hardware changes are completed. The decision to use infrastructure that allows for ciphers to be reconfigured dynamically involves performance considerations and tradeoffs. For example, a software implementation is more general, flexible, and configurable than a custom hardware like Application-Specific Integrated Circuit (ASIC) implementation, but this generality often comes at the cost of performance for high-throughput workloads. On the other hand, while an ASIC may provide higher throughput for certain high-bandwidth cryptographic workloads in comparison with software implementations, ASICs are expensive to design and manufacture, and they are difficult to quickly replace should a cipher be broken. Future work is needed to compare the tradeoffs of CPU/software, FPGA,

and ASIC implementations of cryptosystems for use in a cryptographic infrastructure that is adaptable to the possibility of a sudden shift to a quantum-computing era.

D. MANAGEMENT RECOMMENDATIONS

This thesis recommends that industry follow the lead of the Accredited Standards Committee X9 Incorporated, Financial Industry Standards, and identify a suitable alternative cipher such as the NTRUEncrypt Cryptosystem as the primary algorithm for asymmetric cryptography to replace RSA if needed. Preparations should be made to facilitate a smooth transition if it becomes necessary. If the concern is too great that NTRU is a new algorithm, then this thesis at least recommends that it be added to the published standards as an alternative cipher implementation so that it is available to the industry in the event quantum computers abruptly come into existence.

E. RECOMMENDATIONS EXPLAINED

The author recommends the continued development of quantum-resistant ciphers and the refinement of existing quantum-resistant ciphers such as the NTRUEncrypt Public-Key Cryptosystem as alternatives to traditional public-key algorithms. Peer review is essential to ensure that the ciphers are mathematically sound and that their implementations are free of exploitable implementation flaws. Once quantum computers are realized, we will no longer have RSA to fall back upon in the event we find NTRU is not mathematically sound or has exploitable implementation flaws.

F. RECOMMENDATIONS FOR FUTURE WORK

With published studies showing that the NTRUEncrypt Public-Key Crypto System is more efficient than RSA, future work is needed to consider how NTRU would compare to Kerberos, a system that is solely based on symmetric cryptography, which is also quantum-resistant.

Further testing of implementations of post-quantum ciphers such as the NTRUEncrypt Public-Key Cryptosystem is recommended to validate the implementation. Peer review of the algorithm itself should continue to ensure the security of the cipher.

Research on the use of Field Program Gate Arrays (FPGAs) as part of a strategy to provide a reconfigurable cryptographic infrastructure that is adaptable to the sudden emergence of a quantum-computing era is needed. Such research should explore the performance and cost tradeoffs with respect to software/CPU and custom ASIC implementations. An adaptable infrastructure is one that can easily transition from using ciphers that are vulnerable to quantum computers to using those ciphers that are quantum-resistant instead.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Aaronson, S. (2008). The Limits of Quantum. *Scientific American, Inc, March 2008,* 62-69

Accredited Standards Committee X9 Inc., (2011, April). Cryptographic Security for Financial Services. *X9extra, 2(1)*. Retrieved from https://www.x9.org/home/X9_Extra_April_2011.pdf

Aharonov, D. * Ben-or, M. (1997). Fault-Tolerant Quantum Computation with Constant Error Rate. Retrieved from http://arxiv.org/pdf/quant-ph/9906129.pdf

Barker, E., Barker, W., Burr, W., Polk, W., & Miles, S. (2007). Recommendation for Key Management – Part 1: General (Revised). *National Institute of Standards and Technology*. doi: NIST Special Publication 800-57

Bennett, C. H., Bernstein, E., Brassard, G., & Vazirani, U. (1996). Strengths and Weaknesses of Quantum Computing. Retrieved from http://arxiv.org/pdf/quant-ph/9701001.pdf

Berlekamp, E. R., McEliece, R. J., & Van-Tilborg, H. C. A. (1978). On the Inherent Intractability of Certain Coding Problems. *IEEE Transactions on Information Theory, IT-24(3)*, 384-386. doi: 0018-9448/78/0500-0386

Bernstein, D. J. (2010). Grover vs. McEliece. *National Science Foundation*. doi: e2bbcdd82c3e967c7e3487dc945f3e87

Bernstein, D. J., Buchmann, J., & Dahmen, E. (2009). *Post-Quantum Cryptography*. New York, NY: Srpinger.

Bernstein, D. J., Lange, T., & Peters, C. (2008). Attacking and defending the McEliece cryptosystem. *National Science Foundation.* doi: 7868533f20f51f8d769be2aa464647c9

Bhiogade, M. S. (2002). Secure Socket Layer. *Informing Science,* 85-90. Retrieved from: http://www.informingscience.org/proceedings/IS2002Proceedings/papers/Bhiog058Secur.pdf

Brassard, G., Chuang, I., Lloyd, S., * Monroe, C. (1998). Quantum Computing. *The National Academy of Sciences, 95,* 11,032-11,033. doi: 0027-8424/98/9511032-3

Buckmann, J., Dahmen, E., Ereth, S., Hulsing, A., & Ruckert, M. (2011). On the Security of the Winternitz One-Time Signature Scheme. Retrieved from http://www.springerlink.com/content/a7167435r6587811/fulltext.pdf

Chen, W. ISRC Future Topic Brief: Quantum Computing, *Bauer College of Business Administration, University of Texas.* Retrieved from http://www.bauer.uh.edu/uhisrc/FTB/Quantum/QuantumComputing.pdf

CNSS. (2003, June). National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security and National Security Information. *CNSS Policy No. 15, Fact Sheet No. 1*

CNSS. *(2010, April 26).* National Information Assurance (IA) Glossary. *CNSS Instruction No. 4009*

CyberTech Publishing. (2008). Elliptic Curve Cryptography. Retrieved from http://www.irma-international.org/viewtitle/7306/

Ding, J., Gower, J. E., & Schmidt, D. S. (2006). Springer Science+Business Media, LLC. *Multivariate Public Key Cryptosystems*. (pp. xv-xviii) New York, NY: Springer.

Elgamal, T. & Kipp, E. B. (1998) United States Patent: Secure Socket Layer Application Program Apparatus and Method. *United States Patent Number 5,825,890*

Experimental Quantum Cryptography. (1991, September). Retrived from http://www.hit.bme.hu/~gyongyosi/quantum/cikkek/BBBSS92.pdf

FIPS. (2009). Digital Signature Standard (DSS). *National Institute of Standards and Technology*. FIPS Pub 186-3

Gama, N., Howgrave-Graham, N., & Nguyen, P. Q. (2006). Symplectic Lattice Reduction and NTRU. *Lecture Notes in Computer Science, 4004/2006,* 233-253. doi: 10.1007/11761679_15

Garcia, L. C. C. (2005). On the security and the efficiency of the Merkle signature scheme. *Technical University Darmstadt*. Retrieved from http://eprint.iacr.org/2005/192

Gershenfeld, N., & Chuang, I. L. (1998) Quantum Computing with Molecules. *Scientific American, Inc.* Retrieved from http://www.mat.ucm.es/catedramdeguzman/drupal/sites/default/files/mguzman/01historias/haciaelfuturo/Burgos090900/quantumcomputingSciAmer/0698gershenfeld.html

Grover, L. K. (1996) Quantum Mechanics help in searching for a needle in a haystack. *Physical Review Letters, 79(2)*, 325-328. doi: 0031-9007/97/79(2)/325(4)

Heyse, S. (2009). Code-based Cryptography: Implementing the McEliece Scheme on Reconfigurable Hardware. *Diploma Thesis of the Ruhr-University Bochum.* Retrieved from http://www.emsec.rub.de/media/crypto/attachments/files/2010/04/da_heyse.pdf

Hoffstein, J. Pipher, J. & Silverman, J. H. (1998). NTRU: A Ring-Based Public Key Cryptosystem. *Lecture Notes in Computer Science, 1423/1998,* 267-288. doi: 10.1007/BFb0054868

Hoffstein, J., Howgrave-Graham, N., Pipher, J., & Whyte, W. (2010). Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. *Information Security and Cryptography,* 349-390. doi: 10.1007/978-3-642-02295-1_11

IEEE. (2008). *IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices*. IEEE Std 1363.1-2008

Isaac, C. (2004). Quantum Information: Joining the Foundations of Physics and Computer Science. *MIT Physics Annual, 2004, 26-45*

Lamport, L. (1979). Constructing Digital Signatures from a One Way Function. *Computer Science Laboratory SRI International.* Retrieved from http://research.microsoft.com/en-us/um/people/lamport/pubs/dig-sig.pdf

Lehtinen, S. & Lonvick, C. (2006). The Secure Shell (SSH) Protocol Assigned Numbers. *RFC 4250*

Lo, H. K., & Chau, H. F. (2008). Unconditional Security of Quantum Key Distribution Over Arbitrarily Long Distances. Retrieved from http://arxiv.org/pdf/quant-ph/9803006v5.pdf

Maxim, M. & Pollino, D. (2002). *Wireless Security*. New York, NY: McGraw-Hill Professional

McEliece, R. J. (1978). A Public-Key Cryptosystem Based on Algebraic Coding Theory. *DSN Progress Report 42-44.* Retrieved from http://www.cs.colorado.edu/~jrblack/class/csci7000/f03/papers/mceliece.pdf

Merkle, R. C. (1979). Secrecy, Authentication, and Public Key System. *Student Dissertation of Stanford University.* Retrieved from http://www.merkle.com/papers/Thesis1979.pdf

Metodi, T. S., Thaker, D. D., Cross, A. W., Chong, F. T., & Chuang, I. L., (2005). A Quantum Logic Array Microarchitecture: Scalable Quantum Data Movement and Computation. Retrieved from http://arxiv.org/pdf/quant-ph/0509051.pdf

Naslund, M., Shparlinski, I. E., & Whyte, W. (2002). On the Bit Security of NTRUEncrypt. *Lecture Notes in Computer Science, 2567/2002,* 62-70. doi: 10.1007/3-540-36288-6_5

Nielsen, M. A., & Chuang, I. L. (2002) *Quantum Computation and Quantum Information*. Retrieved from http://n.ethz.ch/~ddukaric/zusammenfassungen/Buecher/Quantum%20Computation%20and%20Quantum%20Information.pdf

NIST. (2007). Recommendation for Key Management – Part 1: General (Revised). *National Institute of Standards and Technology.* NIST Special Publication 800-57.

Onyszko, T. (2002). *Secure Socket Layer.* Retrieved from: http://scholar.googleusercontent.com/scholar?q=cache:Ri17ndmlX6IJ:scholar.google.com/+secure+socket+layer+onyszko&hl=en&as_sdt=0,5

Perlner, R. A., & Cooper, D. A. (2009). Quantum Resistant Public Key Cryptography: A Survey. *National Institute of Standards and Technology.* Retrieved from http://delivery.acm.org/10.1145/1530000/1527028/p85-perlner.pdf?ip=205.155.65.226&acc=ACTIVE%20SERVICE&CFID=83946916&CFTOKEN=17805596&__acm__=1337632583_487c439bf2f39f96658ad849e5690cba

Perry, R. T. (2006, April 29). *The Temple of Quantum Computing.* Retrieved from http://scholar.googleusercontent.com/scholar?q=cache:lwFPe6br5dwJ:scholar.google.com/+The+Temple+of+Quantum+Computing&hl=en&as_sdt=0,5

Pfleeger, C. P. & Pfleeger, S. L. (2003). *Security in Computing* Upper Saddle River, NJ: Pearson Education, Inc.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystem. Retrieved from http://people.csail.mit.edu/rivest/Rsapaper.pdf

Russell, D. & Gangemi, G. T. (2006). *Computer Security Basics, 2nd ed*. Sebastopol, CA: O'Reilly

Schmeh, K. (2003). *Cryptography and Public Key Infrastructure on the Internet*. West Sussex, England: Wiley

Schneier, B. (1996). *Applied Crytography*. West Sussex, England: Wiley

SECOQC. (2007, January 22). *SECOQC White Paper on Quantum Key Distribution and Cryptography*. Retrieved from http://arxiv.org/pdf/quant-ph/0701168v1.pdf

Seroussi, G. (1999). Elliptic curve cryptography. *ITW 1999, Metsovo, Greece, June 27-July 1.* Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00814351

Shor, P. W., (1994). *Algorithms for Quantum Computation: Discrete Log and Factoring.* Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=365700

Steane, A. M. (1995). Error Correcting Codes in Quantum Theory. *Physical Review Letters, 77(5)*, 793-797. doi: 0031-9007/96/77(5)/793(5)

Thorsteinson, G. & Ganesh, G. A. (2003) *.Net Security and Cryptography*. Upper Saddle River, NJ: Prentice Hall

Vandersypen, L. M. K., Steffen, M., Breyta, G., Yannoni, C. S., Sherwood, M. H., & Chuang, I. L. (2001). *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*. Retrieved from http://arxiv.org/pdf/quant-ph/0112176v1.pdf

Walther, P., Resch, K. J., Rudolph, T., Weinfurter, H., Vedral, V., Aspelmeyer, M., & Zeilinger, A., (2005) Experimental One-Way Quantum Computing. *Nature, 7030, 169-176.*

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dr. Dan Boger
   Department of Information Science
   Naval Postgraduate School
   Monterey, California