Quantum Attacks on Modern Cryptography

and Post-Quantum Cryptosystems

Zachary Marron

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2018

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial fulfillment of the requirements for graduation from the Honors Program of Liberty University.

_____

Robert Tucker, Ph.D.
Thesis Chair

_____

Daniel Joseph, Ph.D.
Committee Member

_____

Terri Sipantzi, M.S.
Committee Member

_____

James H. Nutter, D.A.
Honors Director

_____

Date

Abstract

Cryptography is a critical technology in the modern computing industry, but the security of many cryptosystems relies on the difficulty of mathematical problems such as integer factorization and discrete logarithms. Large quantum computers can solve these problems efficiently, enabling the effective cryptanalysis of many common cryptosystems using such algorithms as Shor's and Grover's. If data integrity and security are to be preserved in the future, the algorithms that are vulnerable to quantum cryptanalytic techniques must be phased out in favor of quantum-proof cryptosystems. While quantum computer technology is still developing and is not yet capable of breaking commercial encryption, these steps can be taken immediately to ensure that the impending development of large quantum computers does not compromise sensitive data.

Quantum Attacks on Modern Cryptography and Post-Quantum Cryptosystems

Encryption is a vital technology to our modern world. In an economy and society that thrives from and depends on the proliferation of free and secure access to information by means of the Internet, encryption is a fundamental part of the stack of computer technologies that enables private information to be exchanged securely over public channels. Additionally, encryption safeguards national security by protecting sensitive information and enables private citizens to maintain the integrity and privacy of their own data. Acknowledged or not, encryption is fundamental to modern life. However, encryption is only as useful as it is secure, and many previously secure encryption technologies may soon become vulnerable to attack by new, advanced *quantum computing* systems. Quantum computers manipulate and store data in a manner that leverages the quantum-mechanical structure of the small-scale Universe – this property allows quantum computers to perform calculations that have never been possible with any other type of computer system. While these abilities will certainly lead to massive breakthroughs in the data processing and data analytics industries, the development of quantum computers has an unfortunate side effect. Many of the underlying cryptographic primitives that make modern encryption possible will be rendered insecure in view of the massive calculation power of a quantum computer of appropriate size and design.

However, quantum computers do not spell the doom of cryptography as a whole. Many existing cryptosystems are known to be resistant to the threat posed by quantum

computing technology. While they are not commonly used in practice, quantum computers are becoming increasingly well-studied and thus these algorithms will be vital to the survival of cryptography in the post-quantum-computer age. A full understanding of the vulnerabilities of modern encryption technology, as well as the necessary solutions, is critical in order for our information-based civilization and economy to continue to grow and thrive in spite of the threats posed by this new technology.

Basics of Modern Cryptography

Although cryptography applications are endless and their effects on the computing industry are profound, cryptography's foundation is nothing more than simple mathematics. Although the theoretical background behind some specialized forms of cryptography and the various attack classes that exist for various cryptosystems may require a more thorough education in mathematics, the basics of the most commonly used forms of cryptography are more substantially accessible.

Most cryptosystems can be broadly categorized into one of two groups – *symmetric-key cryptosystems* and *asymmetric-key cryptosystems* (Schneier, 1996). These categories differ in construction and in application, and as a result they present different attack surfaces. Commonly-used asymmetric-key cryptosystems such as RSA (Rivest, Shamir, & Adleman, 1978) and Diffie-Hellman (1976) tend to have mathematical structures that are prone to attack by quantum computers (Shor, 1996). Symmetric-key cryptosystems are generally able to compensate for the potential of quantum attacks by utilizing higher key sizes – in most cases, this achieves an equivalent level of security to that which would be available with a lower key size if the threat of a quantum attack was

not considered (Augot et al., 2015). However, the quantum-specific vulnerabilities found in the most commonly used asymmetric-key cryptosystems are typically fundamental in the design of the cryptosystem (Shor, 1997), and thus cannot be offset by an increase in key size or by a change in some parameter of the cryptosystem.

Symmetric-Key Cryptosystems

Symmetric-key cryptosystems are named for the fact that the same encryption key is used to perform the encryption and decryption operations of the cryptosystem – in other words, the operations are *symmetric* with respect to the key used (Schneier, 1996). To formalize this, let M be the set of all possible "encryptable" messages, let C be the set of all possible ciphertexts, and let K be the keyspace. A symmetric-key cryptosystem defines the functions $E: M \times K \to C$ and $D: C \times K \to M$ such that for any $m \in M$ and $k \in K$, there exists $c \in C$ such that E, D satisfy:

$$E(m, k) = c \qquad D(c, k) = m$$

Symmetric-key cryptography is highly suitable for applications involving a single party, such as encryption of files on a disk (Schneier, 1996). Since the key is known to the owner of the encrypted files, the files can be encrypted and decrypted by their owner whenever necessary. However, they cannot be read by other parties while they are encrypted. Symmetric-key cryptography is highly useful for communicating large amounts of data securely between two or more parties, as symmetric-key algorithms tend to be relatively efficient and support the fast encryption and decryption of large quantities of data. Some symmetric-key cryptosystems that are common in modern usage include

AES (Advanced Encryption Standard), 3DES (Triple Data Encryption Standard), Blowfish, and Serpent.

However, this form of communication requires that all parties pre-establish the symmetric key to be used, and this presents a challenge when the parties are forced to communicate over an insecure channel. The key cannot simply be transmitted unprotected over the channel, as an eavesdropper could obtain the key in transit and passively decrypt all further communications between the two parties. Fortunately, asymmetric-key cryptography enables the secure exchange of data over an insecure channel without any kind of pre-established shared secret, albeit at a performance cost (as asymmetric-key algorithms tend to be dramatically slower than symmetric-key algorithms) (Schneier, 1996). Thus, a common strategy is to use an asymmetric-key encryption to agree on a symmetric key, which is then used for further communications.

Asymmetric-Key Cryptosystems

Asymmetric-key (or public-key) cryptosystems are named for the fact that different encryption and decryption keys are used for the encryption and decryption operations of the cryptosystem (Schneier, 1996). The encryption key is commonly referred to as the *public key* and the decryption key is commonly referred to as the *private key*. This terminology comes from asymmetric-key cryptography's key use case – the ability to publish the encryption key, allowing third parties to encrypt data that will henceforth only be readable by the owner of the private decryption key (Schneier, 1996). Formally, if K_e is the public-key keyspace and K_d is the private-key keyspace, an

asymmetric-key cryptosystem defines the functions $E: M \times K_e \rightarrow C$ and $D: C \times K_d \rightarrow M$ such that for any $m \in M$ and keypair $(k_e, k_d) \in K_e \times K_d$, there exists $c \in C$ such that:

$$E(m, k_e) = c \qquad D(c, k_d) = m$$

As mentioned previously, asymmetric-key algorithms tend to be substantially slower than their symmetric-key counterparts. This is due to the computationally expensive mathematical operations that are involved in running E, D for an asymmetric-key cryptosystem (Schneier, 1996). However, as public keys in such a scheme can be disseminated publicly without threatening the integrity of the encryption, asymmetric-key cryptography is very useful for solving the problem of an initial secure data exchange over an insecure connection (Schneier, 1996).

If a possibly insecure but reliable connection exists between two parties in a communications network, an asymmetric encryption scheme can be used to negotiate and agree on a *shared secret key* that can then be used for high-speed symmetric-key encryption between the two parties. The connection is allowed to be insecure because the only data that needs to pass over the connection is one of the parties' public key and the corresponding response consisting of the encrypted shared secret. Neither of these allow an eavesdropper to read the content of the encrypted message, but the initial sender of the public key retains the private key needed to decipher the transmission and read the response, obtaining the shared secret (Schneier, 1996).

However, reliability is critical – if an attacker can intercept and replace communications on the network between the two parties, the attacker can relay his own public key to the second party. When the second party responds, the attacker may then

decipher the shared secret with his own key, re-encrypting and passing along the message under the initial sender's public key. The attacker will then have the ability to read all further symmetric communications passed between the two parties. This form of attack is known as a man-in-the-middle (MITM) attack and, other than a minor amount of added latency, may be undetectable by either legitimate party (Schneier, 1996). The endpoint parties believe that they are communicating directly while their messages are, in fact, being intercepted and altered by the attacker. Thus, to ensure that each party receives the genuine public key owned by their counterpart, it is critical that the connection be reliable or that the parties have some additional external method for authenticating the public keys as genuine (Schneier, 1996).

As long as its weaknesses are properly understood and mitigated, asymmetric-key cryptography fills a crucial gap in the capabilities provided by symmetric-key cryptography alone. This technology enables the initialization of secure communication over insecure channels – this can be used to bootstrap ad-hoc symmetric-key encryption such as that provided by SSL/TLS (Dierks & Rescorla, 2008). However, many common asymmetric-key cryptosystems are also highly vulnerable to quantum-computer-specific attacks due to their unique mathematical construction (Shor, 1997), and thus it is critical that the vulnerabilities that exist in commonly-used asymmetric-key cryptosystems are discovered, understood, and mitigated before the theoretical attacks made possible by a quantum computer become a reality.

Quantum-Vulnerable Cryptography

Many of the most commonly used cryptographic standards and protocols are

vulnerable to quantum-computer-based attacks. As described previously, symmetric and asymmetric cryptosystems alike must define an encryption function $E(m, k_e) = c$ and a decryption function $D(c, k_d) = m$. In this notation, m refers to the (unencrypted) message to be encrypted, c refers to the corresponding ciphertext, and k_e, k_d refer to the encryption and decryption keys respectively. An asymmetric cryptosystem has $k_e \neq k_d$, while symmetric cryptosystems have $k_e = k_d$.

The foundation of a secure cryptosystem is the assumption that one cannot obtain c solely from m (without knowledge of k_e) or obtain m solely from c (without knowledge of k_d.) Focusing on asymmetric cryptosystems, we find that many of these systems apply problems that are (or are believed to be) outside of the P algorithmic complexity class – as solutions to these problems are not known to be deterministically computable in polynomial time (but can generally be verified in polynomial time,) they are ideal for use in an effective encryption scheme. If a cryptosystem is constructed such that the computation of c from m is a non-P problem, we can clearly not reverse the encryption *efficiently* (i.e. in polynomial time) without additional information. However, the encryption can easily be constructed or "verified" in polynomial time. The result is an encryption that is efficient to form but cannot be easily broken – a highly desirable trait in asymmetric cryptosystems.

Quantum computers adversely affect these security assumptions – many of the underlying mathematical problems leveraged in asymmetric cryptography are known to be in the BQP complexity class, which is the class of problems that can be solved by a quantum computer in polynomial time under a bounded error threshold (Shor, 1997).

BQP problems can generally be solved efficiently given a quantum computer of sufficient size, and this is an obvious problem as BQP problems are frequently used in modern cryptography (Shor, 1997). Cryptosystems that are vulnerable to efficient quantum cryptanalysis are, in general, reducible to a BQP problem that can then be efficiently solved using an appropriate quantum computing system.

RSA

The RSA cryptosystem was developed by MIT cryptographers Ronald Rivest, Adi Shamir, and Leonard Adleman (for whom the cryptosystem was named) in the late 1970s (Rivest, Shamir, & Adleman, 1978). RSA was one of the first examples of a practical asymmetric encryption scheme, and its relative simplicity and (to the best extent known) high level of security caused it to quickly become the "gold standard" for asymmetric encryption. As the fields of computing, cryptography, and eventually internet communications continued to grow throughout the latter decades of the 20th century, RSA enjoyed continued use. Today, RSA is very widely used across the entire computing industry and is one of the default algorithms used in SSL/TLS to secure internet communications (Dierks & Rescorla, 2008).

Since RSA is an asymmetric encryption system, each party in an RSA encryption transaction must possess a private decryption key and a public encryption key. To generate the public RSA encryption key, each party must select two large, random prime numbers p and q. These numbers are kept secret, but their product $n = pq$ is computed and is referred to as the *public modulus* (Rivest, Shamir, & Adleman, 1978). Then, an *encryption exponent* e is selected that satisfies $gcd\big(e, \phi(n)\big) = 1$ (ϕ denoting the Euler

totient.) In practice, e may be selected as a small prime for simplicity of the calculation, as $gcd(p,k) = 1$ for any prime p and any $k \in \mathbb{Z}$. The RSA public key is the pair (n, e). Once these calculations have been performed, the calculation of the RSA private key is a 1-step process – the decryption exponent d is given by:

$$d = e^{-1} \left(\text{mod } \phi(n)\right)$$

In other words, the decryption exponent is the inverse of the encryption exponent modulo $\phi(n)$. The RSA private key is the pair (n, d).

Encryption and decryption. The RSA encryption function is defined by:

$$E(m, n, e) = m^e \ (\text{mod } n)$$

We then denote $c = E(m, n, e)$. The RSA decryption function is defined by:

$$D(c, n, d) = c^d \ (\text{mod } n)$$

Note that as $ed \equiv 1\left(\text{mod } \phi(n)\right)$, a corollary to Euler's totient theorem provides that $m^{ed} \equiv m \ (\text{mod } n)$. Then, observe:

$$\begin{aligned} D(c, n, d) &= c^d \ (\text{mod } n) && \text{(Definition)} \\ &= (m^e)^d \ (\text{mod } n) && (c = m\text{^}e \ (\text{mod } n)) \\ &= m^{ed} \ (\text{mod } n) && \text{(Properties of exponents)} \\ &= m \ (\text{mod } n) && (m^{ed} \equiv m \ (\text{mod } n)) \end{aligned}$$

We therefore see that $D(c, n, d) = m \ (\text{mod } n)$, so if $m < n$, we fully recover m and find that $D(c, n, d) = m$. Thus, in general we must have $m < n$, but best practices for usage of the RSA cryptosystem (and most commercial implementations) generally recommend that n be at least a 2048-bit or a 4096-bit number (Barker, 2016). Thus, dependent on key size, several hundred bytes can be exchanged in a single RSA transaction, and this is

often sufficient for many of RSA's applications (such as symmetric encryption key exchange.) Larger messages can be accommodated by dividing the message into smaller chunks and performing multiple RSA calculations.

Vulnerability to quantum computers. The vulnerability of the RSA cryptosystem to quantum computer-based attacks lies in the RSA algorithm's reliance on the difficulty of the *integer factorization problem* as the basis for the cryptosystem's security. The integer factorization problem, which refers to the decomposition of an arbitrary positive integer into its unique prime factorization, has no known solution in polynomial time (Shor, 1997). However, the nonexistence of such a solution has never been proven and is an open problem in computer science.

Furthermore, the reversal of an RSA encryption without the knowledge of the private key (informally referred to as the RSA problem) is *at most* as difficult as the integer factorization problem, and this can be seen by the brief description of the RSA algorithm given above. Suppose that an attacker is in possession of an encrypted RSA ciphertext c and would like to discover m, the original message. The attacker is also presumed to be in possession of the associated RSA public key (n, e).

Traditionally, the RSA private key (n, d) is used to decrypt the payload – by raising the ciphertext to the d-th power and taking the residue modulo n, m will be recovered (as outlined in the proof sketch above). However, the attacker does not know d. As noted above, d is the modular inverse of e (which is known to the attacker) with respect to $\phi(n)$, but $\phi(n)$ is also unknown to the attacker and cannot (classically) be efficiently computed directly from n. As $\phi(n) = (p-1)(q-1)$, this computation

requires knowledge of n's prime factors p and q (Shor, 1997). If $\phi(n)$ is known, d can be efficiently computed via the extended Euclidean algorithm – this is in fact the exact calculation used in the initial derivation of an RSA private key. Thus, as soon as p and q are known to the attacker, the attacker can easily compute $\phi(n)$, which allows easy computation of d. This provides the attacker with the corresponding RSA private key and the ability to decrypt the encrypted payload. We must therefore conclude that an efficient solution to the integer factorization problem is an efficient solution to the RSA problem. However, integer factorization is known to be a BQP problem (Shor, 1997) and thus has an efficient solution for a quantum computer of sufficient size and capability. Shor's algorithm is the best known such solution and was the first to prove that integer factorization is a BQP problem.

We have shown that solving the RSA problem is at least as easy as solving the integer factorization problem and have indicated that the integer factorization problem is solved efficiently (in polynomial time) using Shor's algorithm on a quantum computer of appropriate size. Thus, we are forced to conclude that RSA encryption of arbitrary key-length is directly vulnerable to efficient cryptanalysis by quantum-computer-based methods. Thus, the continued use of RSA is unsafe in an environment where a quantum computer capable of running Shor's algorithm for RSA-sized (2048 or 4096-bit) numbers exists. Fortunately, no such quantum computer is known to exist at the present. However, ongoing research and development in quantum computing may yield a quantum computing system in the near future that is capable of accomplishing this calculation.

Diffie-Hellman

The Diffie-Hellman key exchange protocol was developed by Stanford University researchers Whitfield Diffie and Martin Hellman during the 1970s (Diffie & Hellman, 1976). Diffie-Hellman is a simple protocol to establish a shared secret key between two parties over an insecure channel and was one of the first examples of a cryptographic public-key protocol. The construction of Diffie-Hellman is similar in many ways to the RSA algorithm, but Diffie-Hellman's narrower scope (supporting only the establishment of a shared secret, not general-purpose communications) and simpler algorithm makes it a common choice for this application. The original Diffie-Hellman algorithm (as well as variants of the algorithm that substitute groups such as an elliptic curve group in place of the multiplicative group $\mathbb{Z}_p$ in the algorithm) is still widely in use today, including as a part of the default cryptography suites for SSL/TLS (Dierks & Rescorla, 2008). Diffie-Hellman derives its cryptographic difficulty from the discrete logarithm problem, which is believed to have no efficient classical solution over groups such as $\mathbb{Z}_p$ (though this is unproven and is an open problem in number theory and computer science.)

The protocol. Conventionally, the two parties involved in an encrypted communication are informally referred to as Alice and Bob – this terminology was devised by Rivest, Shamir, and Adleman (1978) in their original RSA paper and its use has since become common. Thus, let Alice and Bob refer to the two parties that wish to establish a shared secret using Diffie-Hellman. First, Alice and Bob must agree on a finite cyclic group of large order and a generator g of the cyclic group. In the most common applications, a large prime p is selected and the cyclic group used in the Diffie-Hellman

calculation is $\mathbb{Z}_p$ (Diffie & Hellman, 1976). The generator g is thus a primitive root modulo p. Both p and g are communicated between Alice and Bob and are not private. Alice and Bob then select *secret exponents* a, b respectively – Alice computes the value $x = g^a \pmod p$ and Bob computes the value $y = g^b \pmod p$ (Diffie & Hellman, 1976). Alice then sends x to Bob and Bob sends y to Alice. Alice computes $y^a \pmod p$ and Bob computes $x^b \pmod p$. However, observe that:

$$y^a \pmod p = \left(g^b\right)^a \pmod p = g^{ba} \pmod p = g^{ab} \pmod p = (g^a)^b \pmod p = x^b \pmod p$$

Thus, Alice and Bob have computed the same value, as we see that $y^a \equiv x^b \pmod p$. This value is the shared secret (Diffie & Hellman, 1976).

The Diffie-Hellman protocol allows Alice and Bob to arrive at a single shared value securely because of the limited nature of the information communicated between the two parties (Diffie & Hellman, 1976). An eavesdropper to the execution of the Diffie-Hellman protocol can determine p and g from the initial calculation and can further obtain $x = g^a \pmod p$ and $y = g^b \pmod p$ as these values are exchanged between the two parties. However, the secret value is $g^{ab} \pmod p$ and thus the attacker must learn either a or b in order to exponentiate x or y and determine the secret. Assume without loss of generality that the attacker will attempt to obtain a – the attacker knows g, p, and $x = g^a \pmod p$. Retrieving a from these values alone is known as the *discrete logarithm problem*, as the operation is analogous to taking the traditional logarithm base g in $\mathbb{R}$. Classically, the discrete logarithm problem is believed to be computationally infeasible for a large modulus p (Diffie & Hellman, 1976).

Vulnerability to quantum computers. The discrete logarithm problem, much like the integer factorization problem, is not known to have an efficient classical solution for many cyclic groups, including certain cyclic groups of large prime order (Diffie & Hellman, 1976). Thus, the discrete logarithm problem for these groups is largely assumed to not be in the P complexity class, although this is an open problem in computer science as no proof has ever been given of the nonexistence of a polynomial-time algorithm. Thus, the discrete logarithm problem is generally regarded as computationally intractable on classical computers, rendering it a secure foundation for algorithms such as Diffie-Hellman.

An efficient solution to the discrete logarithm problem provides an efficient solution to Diffie-Hellman, as illustrated above – such a solution could compute a given only g, p, and x (all of which are visible to an attacker eavesdropping on a Diffie-Hellman negotiation), and the attacker can then compute x^a, the shared secret. Thus, breaking Diffie-Hellman is at least as easy as the discrete logarithm problem. However, Shor's quantum algorithm for integer factorization describes a modified version of the algorithm that solves the discrete logarithm problem as well (Shor, 1997). Thus, an attacker with a quantum computer of sufficient size could use Shor's algorithm to solve the discrete logarithm problem and compute a from these values alone, breaking the security of the Diffie-Hellman protocol.

Symmetric-Key Algorithms

Symmetric-key cryptosystems have vastly different mathematical structures in comparison to asymmetric-key cryptosystems (Schneier, 1996). As a result, they do not

present the same vulnerabilities to quantum-computer-based attacks. However, these cryptosystems are not invulnerable to quantum computer effects and considerations must be made in order to ensure that they remain secure even after the introduction of large quantum computers to the cryptanalysis industry.

A quantum algorithm known as Grover's algorithm provides the (currently known) most substantial enhancement to cryptanalysis for symmetric-key systems such as AES (Grover, 1996). Specifically, given a function f with a domain of cardinality n, Grover's algorithm allows for the preimage under f of an arbitrary value to be calculated with $O(\sqrt{n})$ trial evaluations of the function on average, whereas a normal "brute-force" procedure would require $O(n)$ trial evaluations of the function on average (Grover, 1996). This represents a quadratic speedup of any brute-force procedure for an attacker with the capability of running Grover's algorithm, and since the algorithm is broadly applicable to any black-box function, it can be used to produce a quadratic speedup in brute-forcing symmetric ciphers such as AES.

An attacker in possession of a corresponding plaintext and ciphertext for a symmetric encryption scheme can use Grover's algorithm to more efficiently discover the key used in the encryption over brute-forcing. This is reasonable to consider, as many forms of encrypted communication and data storage use attacker-influenceable, repeated, or predictable data (such as HTTP headers, for example.) The attacker will then conduct a brute-force attack over the keyspace to discover the key. For the sake of example, assume that the encryption being used is AES with a 128-bit key size. There are therefore 2^{128} keys that could be used, and a brute-force attacker must (on average) try half of these

keys (2^{127}) in a classical computing scenario to discover the correct key. In the worst case, every key must be tried. Grover's algorithm, however, allows this brute-force operation to be accomplished with $O(\sqrt{n})$ trials of AES, where $n = 2^{128}$ is the size of the keyspace. Thus, only 2^{64} AES operations will be required (on average) to discover the key with a high degree of probability – a quadratic speedup over classical brute-forcing.

The efficiency difference made by Grover's algorithm can make previously infeasible (or borderline infeasible) AES brute-forcing operations practical for a well-equipped attacker. The brute-force computation of a 128-bit AES key is far beyond the capability of even the most well-equipped state-level attackers and would require millions of years of continuous calculation even with a vast amount of computational resources (Schneier, 1996). However, the brute-forcing of a 64-bit AES key is a much less computationally expensive operation – attacks on symmetric keys of this size have been demonstrated to be practical (Güneysu, Kasper, Novotný, Paar, & Rupp, 2008). Thus, Grover's algorithm would potentially enable the brute-forcing of lower-end AES keys given the appropriate quantum computing resources. However, the quadratic speedup can be effectively nullified by doubling the key size used in the symmetric cipher – for instance, a 256-bit AES key will then require $O(2^{128})$ operations to brute-force. Thus, by using key sizes that are at a minimum twice the bit length of the minimum recommended bit length for symmetric encryption keys under a classical computing paradigm, the security of symmetric-key algorithms can be preserved. With appropriate modifications, these algorithms can continue to be used even after the development of large, general-purpose quantum computers.

Quantum Algorithms

A quantum algorithm is an algorithm designed for use on a quantum computer, leveraging quantum-mechanical data structures and quantum-mechanical physical phenomena to perform calculations that would otherwise not be possible or feasible using a traditional computer. While classical binary computers use bits to represent data and perform operation on those bits, quantum computers use *qubits* to represent data. While a classical bit can only take one of two states (written 0 and 1) at any given time, a qubit is a superposition of two states (written $|0\rangle$ and $|1\rangle$) that collapses to one of the two states upon observation (Rieffel & Polak, 2000). A qubit can be thought of as a complex number with unit length: a qubit a could be written $a = a_0|0\rangle + a_1|1\rangle$, where $|a_0|^2$ is the probability that $|0\rangle$ will be observed, and $|a_1|^2$ is the probability that $|1\rangle$ will be observed. Equivalently, we can think of a as a unit-length vector in a 2-dimensional Hilbert space.

Qubits can be strung together in quantum registers to represent larger quantities of data, just as bits can be strung together to represent large amounts of data in classical computing systems. The values in these registers are also superpositions (comprised of the individual superpositions in their qubits) and can also be viewed as unit-length vectors in Hilbert spaces of 2^k-dimension for a register of k qubits (Shor, 1997). Observation of the quantum register collapses the superposition, and one of the 2^k basis vectors of the 2^k-dimensional space is observed. This is analogous to how k bits in a classical computer can represent 2^k unique states. However, the ability of a quantum computer to represent data in superpositions and perform operations upon superpositions

allows for computational shortcuts (Rieffel & Polak, 2000). Appropriately designed algorithms can use constructive and destructive interference to make the observation of states corresponding to solutions to a problem occur with high frequency (Shor, 1997). This has the perceived effect of performing a calculation for multiple possible inputs simultaneously – while the outputs corresponding to each input are not individually accessible as part of the superposition, such an algorithm can produce a "correct" value in response to observation with much better success over random choice. This principle is what allows algorithms such as Shor's to perform a seemingly impossible amount of computational work in a relatively small number of quantum computer operations.

Shor's Algorithm

In 1994, mathematician Peter Shor developed and published a theoretical algorithm for the computation of an integer's prime factorization with polynomial time complexity (Shor, 1997). The algorithm was theoretical in the sense that it was an algorithm to be run on a quantum computer, which was a concept (at the time of the 1994 publication of Shor's paper) that had been well-studied but had never been implemented. Shor's algorithm was highly significant because no previous algorithm for the prime factorization of arbitrary integers in polynomial time had yet been described (Shor, 1997). Prior to the discovery of Shor's algorithm, the general number field sieve (GNFS) was one of the most efficient algorithms known for the prime factorization of arbitrary integers on a classical computer. However, the GNFS has the following sub-exponential time complexity for the factorization of an integer n (Pomerance, 1996):

$$O(\exp\left(\left(\sqrt[3]{\frac{64}{9}}\right)\cdot(\log n)^{\frac{1}{3}}\cdot(\log\log n)^{\frac{2}{3}}\right))$$

While this indicates that the GNFS is more efficient than any algorithm of exponential time complexity, the amount of computational effort required to factor very large, randomly generated compound integers (such as 2048-bit RSA keys) using the GNFS makes the algorithm essentially useless for these applications. Shor's algorithm accomplishes the factorization of these large integers in polynomial time. This effectively places the efficient factorization of RSA-sized numbers within the realm of possibility for a sufficiently large and appropriately designed quantum computer. An efficient method for large integer factorization fundamentally negates the security assumptions of the integer factorization problem. Therefore, a working implementation of Shor's algorithm for large integers poses an existential threat to RSA and other cryptosystems that rely on the computational intractability of the integer factorization problem and other related, foundational “hard” problems in number theory (Shor, 1997).

The algorithm. In the formulation of his algorithm, Shor (1997) assumed that quantum computers would be substantially more difficult to build and operate than classical computers. Thus, operation-for-operation, quantum computation would be more expensive than classical computation. As a result, it is advantageous for as many of the calculations in the algorithm as possible to be performed by a classical computer, leaving only the calculations that strictly require quantum-mechanical properties for the quantum computer to perform. While this increases the complexity of a Shor's-algorithm-based system for integer factorization (via the integration of non-quantum computing systems

with quantum computing systems,) the performance benefits of utilizing the least possible amount of quantum computing resources far outweigh this complexity penalty, as quantum computing systems are (as of yet) extremely difficult to design and build, particularly at scale.

To this end, Shor reduces the question of finding the prime factorization of an integer n to the calculation of the order of an arbitrary element x in the multiplicative group $\mathbb{Z}_n^*$. To simplify this analysis and to most closely emulate the cryptographic application of Shor's algorithm, we will assume that n is a *semiprime* – that is, that n is the product of exactly two prime numbers p and q:

$$n = pq$$

As the RSA algorithm constructs its public modulus as a semiprime in this way, this structure for n is the situation of greatest importance for the use of Shor's algorithm in the cryptanalysis of RSA. A black-box function that can perform the aforementioned order-finding operation can be shown to lead to a prime factorization of n with a high degree of probability in the following manner: given an arbitrary $x \in \mathbb{Z}_n^*$, use the black-box function to find r, the order of x in $\mathbb{Z}_n^*$. Then, observe that:

$$\left(x^{\frac{r}{2}} + 1\right)\left(x^{\frac{r}{2}} - 1\right) = x^r - 1$$

However, as $x^r = 1$ (since r is the order of x,) $x^r - 1 \equiv 0 \equiv n \pmod{n}$. Thus, $\left(x^{\frac{r}{2}} + 1\right)$ and $\left(x^{\frac{r}{2}} - 1\right)$ are factors of n. This procedure has two failure modes – if r is odd, $\frac{r}{2}$ is clearly not an integer and thus the calcuation is meaningless in $\mathbb{Z}_n^*$. Furthermore, if $x^{r/2} \equiv -1 \pmod{n}$, we easily see that the factorization obtained is trivial. Thus, if either

of these situations is encountered, we will need to repick a random x and repeat the procedure until a nontrivial factorization is found. Shor (1997) calculates that for the case of a semiprime, there is at least a 50% probability that the period of a random $x \in \mathbb{Z}_n^*$ will yield a factorization in this way. Thus, there is a more than 99% probability that a nontrivial factorization will be found after the completion of at least seven trials.

We have thus found that a black-box function capable of calculating the order of arbitrary elements in $\mathbb{Z}_n^*$ provides a simple (and classically computable) route to the factorization with high probability. This black-box is the core of Shor's algorithm and is the component of the algorithm that requires quantum computation (Shor, 1997).

The quantum algorithm manipulates two registers of $\log q$ qubits each – q is the unique power of 2 such that $n^2 < q < 2n^2$ (Shor, 1997). Assuming that both registers are initially set to $|0\rangle$ (the basis vector chosen to represent 0 in the Hilbert space in which one may visualize the quantum registers,) Shor (1997) sets the first register to the uniform distribution superposition of all values in $\mathbb{Z}_n$, so we have:

$$\frac{1}{\sqrt{q}}\sum_{a=0}^{q-1}|a\rangle|0\rangle$$

The second register is set to the superposition of all modular exponentiations of our random x to elements of $\mathbb{Z}_n$, so we have the fully initialized state:

$$\frac{1}{\sqrt{q}}\sum_{a=0}^{q-1}|a\rangle|x^a \pmod n\rangle$$

Shor (1997) notes that the calculation of the modular exponentiation superposition is the most expensive part of the algorithm and carries the largest time and space costs.

Once the quantum computer is initialized to the aforementioned state, the quantum Fourier transform is applied to the first register and the state of the system is then observed – let c be the observed value from the first register. As observation has collapsed the superposition, c is a basis vector in the Hilbert space represented by the quantum registers and thus represents a q-bit integer. Shor (1997) indicates that c approximates the desired period and shows that as a result of the calculation, there exists at most one integer d satisfying the inequality $\left|\frac{c}{q} - \frac{d}{r}\right| < \frac{1}{2q}$. Furthermore, certain values of c allow $\frac{d}{r}$ (which is not known) to be computed from $\frac{c}{q}$ (which is known) in polynomial time. If d and r are coprime, the calculation yields r, the desired order. Shor (1997) notes that not all values of c will permit this calculation, but due to interference properties of the quantum calculation, the observed value c can be used to find r with a high degree of probability. Specifically, Shor (1997) calculates that the algorithm can find r with a probability of $\frac{\delta}{\log\log r}$ for some constant δ, so if this procedure is repeated until r is found, we find that this portion of the procedure has a complexity of $O(\log\log r)$ and finds r with a high degree of probability. r is then used to compute the prime factorization of n efficiently by the procedure described previously (Shor, 1997).

Grover's Algorithm

Mathematician Lov Grover (1996) of Bell Labs published a paper describing a quantum algorithm for a fast database search. Classically, the best way to locate an item in an unsorted list or database of n items (about which no other information is known) is to sequentially traverse the list until the item is found. In the worst case, the item is

located at the "end" of the database and thus n items must be examined (Grover, 1996). However, assuming that the item is expected to be anywhere in the list with equal probability, the item will be in the first half of the list 50% of the time and in the last half of the list 50% of the time. Thus, on average, $\frac{n}{2}$ items will be examined before the desired item is found. It is clear that the difficulty of the search scales linearly with the length of the list, as $O(n)$ operations are required on average to find the required item. This technique is called a *brute-force* search and is guaranteed to eventually locate the item.

For many strong symmetric encryption algorithms (such as AES), brute-force search of the keyspace is the best-known method for determining the key that maps a known plaintext to a known ciphertext. However, AES with 128-bit keys has 2^{128} unique keys that can be used, and thus we can interpret the AES keyspace as a "database" with 2^{128} entries, exactly one of which is the correct key. A brute-force search of this database will require 2^{127} operations on average – an extraordinary amount of computation that appears to be (for the meantime) well out of the realm of possibility, even with the combined computing resources of the human race (Schneier, 1996). Grover (1996) describes a quantum algorithm for locating a desired item within this type of database using only $O(\sqrt{n})$ operations on average – a massive speedup that has the potential to endanger the security of smaller symmetric encryption keys.

The algorithm. Grover's algorithm primarily operates on a single quantum register – to search a database with n entries, Grover's algorithm requires $\log n$ qubits so that each of the indices of the database items can be represented (Grover, 1996). As in Shor's algorithm, the register is initialized to a superposition where every value has a

probability amplitude of $\frac{1}{\sqrt{n}}$ corresponding to a $\frac{1}{n}$ probability of observing this particular value. The intial machine state is thus:

$$\frac{1}{\sqrt{n}}\sum_{x=0}^{n-1}|x\rangle$$

Next, a black-box oracle operator is applied to the register. This operator inverts the probability amplitude of a state *only* if it matches the search value – all other states remain unchanged by the operator (Grover, 1996). This oracle operator may vary depending on the type of search that is being performed or the type of problem that is being solved – for the application of cracking an AES key, this oracle would likely evaluate a key to determine if it is the correct one.

Finally, a *diffusion operator* is applied to the register. This has the effect of inverting and amplifying the inverted-amplitude states of the superposition (Grover, 1996). The net effect of the two operators is that the target state is returned to a positive probability amplitude of a value greater than its original amplitude of $\frac{1}{\sqrt{n}}$, causing the remaining states to decrease evenly in amplitude to compensate (as the diffusion transformation is unitary and preserves the property that the sum of the squares of the amplitudes is 1.)

To complete the algorithm, these two operators are repeatedly applied to boost the probability amplitude of the desired state (Grover, 1996). This has the effect of making observation of the desired *target* state much more likely in comparison to *non-target* states following multiple iterations. Grover (1996) shows that the answer can be determined in this way using $O\left(\sqrt{n}\right)$ applications of the operators. Thus, for a 128-bit

AES key, only $O(2^{64})$ operations are required on average for the search, in comparison to $O(2^{128})$ operations for a pure brute-force search on a classical computer. The bit-length security of the key is effectively halved by this attack, placing the computation of cryptographic objects such as 128-bit AES keys within the realm of possibility for a well-equipped attacker with a quantum computer of appropriate size. However, the attack presented by Grover's algorithm is essentially nullified by doubling the key size, and thus as an (effective) security of at least 128 bits is currently recommended for AES keys (Barker, 2016), an effective strategy for securing AES encryption against possible quantum attacks is to use a minimum of 256 bits for AES keys. It is important to note that increased key length alone does not make Grover's algorithm any less effective, and smaller keys will still be potentially open to cryptanalysis. However, the effective security contributed by the increased key length can raise the computational effort required to perform Grover's algorithm to the point of infeasibility, even for a quantum computer. Thus, the same principle of key-doubling can resolve the corresponding threat for other symmetric-key systems as well.

Quantum Computers: The State of the Art

Quantum computers are inherently difficult to design, build, and operate. In a twist of irony, the quantum-mechanical physical properties that lend quantum computers their power also introduce fundamental instabilities that are difficult to overcome at small scales, let alone the larger scales required for useful, powerful quantum computation. Classical digital computers are able to store their fundamental unit of information, the *bit*, in forms such as the level of voltage present in an electronic circuit, the state of a

specially designed memory-cell circuit (such as that used in random-access memory), the magnetic polarity of a region on a magnetic disk (such as those used in traditional magnetic disk drives), or even in the form of physical high and low points engraved on the tracks of an optical media disk. Classical binary data can be easily read and manipulated in each of these formats because each format is merely a *representation* of the corresponding abstract binary data (consisting of 1's and 0's.) In order to perform the translation, the appropriate I/O device needs only to be able to reliably differentiate between the encoding of a 0 and the encoding of a 1 in the appropriate format.

Quantum computers, on the other hand, use the *qubit* as their fundamental unit of information. A qubit represents the superposition of two basis quantum states, termed $|0\rangle$ and $|1\rangle$ respectively (Rieffel & Polak, 2000). To realize a qubit as a physical object in a physical quantum computing system, it is therefore necessary to use some form of physical system that is capable of taking on a superposition of two quantum states, as described. Examples of such a system include the polarization of a photon (which is a superposition of the *horizontal* and *vertical* polarization states,) as well as the spin of an electron (which is a superposition of the spin-up and spin-down quantum states.) However, just as one of these superpositions will collapse and yield one of the basis states upon planned observation, interaction between the qubit and the environment will result in spontaneous decay of the qubit's superposition and a potential premature loss of information (Rieffel & Polak, 2000). This phenomenon is known as *quantum decoherence*, and in order to combat it, quantum computer systems are often stabilized at extremely low temperatures (often a few fractions of a degree above $0° K$) to minimize

decoherence effects. Even then, quantum computer systems require an intense amount of engineering effort to produce a stable system where qubits can maintain their state long enough for a calculation to be performed. Even in the most well-isolated systems, individual qubits spontaneously decay in timeframes on the order of nanoseconds to microseconds (Rieffel & Polak, 2000). Thus, there are extreme challenges to building quantum computers that can perform calculations that are *both* usefully large and also usefully long-lived. However, there are a growing number of instances where quantum computers have been successfully developed, built, and operated, steadily increasing the number and lifespan of system qubits that are available for computation.

IBM Q

IBM's quantum computer research division (*IBM Q*) announced in November 2017 that they had successfully developed and tested a prototype of a general-purpose 50-qubit prototype processor and that they would be immediately making a 20-qubit version of the architecture available to clients through a quantum-computing-as-a-service model (Gil, 2017). The original iterations of the IBM Q program (which debuted in May 2016) allowed the public to interact with a basic quantum computer system, and later versions of the service allowed users to access 5-qubit and 16-qubit quantum computers. The new 20-qubit system and prototype 50-qubit system have an average decoherence threshold of approximately 90 microseconds (Gil, 2017).

D-Wave

The Canadian quantum computing company D-Wave currently produces a specific type of quantum computer for commercial applications – the current D-Wave

flagship model is the D-Wave 2000Q, a *quantum annealing* quantum computer with a 2048 qubit processor (D-Wave Systems, 2017). However, D-Wave's quantum annealing system is far less general-purpose than competing implementations and is suited only to a certain class of problem involving minimization and optimization of functions. Furthermore, D-Wave's qubits are more unstable and more error-prone than competing implementations, contributing additional overhead for error detection and correction (Lee, 2017).

As a result of these effects, there has been significant controversy surrounding the ability of D-Wave's quantum computing systems to actually perform computation (as marketed) in a manner that is materially more efficient than the ability of a classical computer. Studies and investigations into these claims have yielded inconclusive results (Amin, 2015).

Post-Quantum Cryptosystems

It is clear that in light of the abilities of impending quantum computer research, new cryptographic algorithms will be required to replace those that can be compromised by quantum-computer-based attacks. These algorithms are commonly referred to as *post-quantum cryptosystems* to reflect their usage following the general availability of large quantum computers. While AES and other symmetric-key cryptosystems can generally compensate for the effects of algorithms such as Grover's algorithm by simply doubling the key size, RSA, Diffie-Hellman, and many other systems and protocols for asymmetric encryption and key exchange will not be effectively usable. This is due to the existence of algorithms (such as Shor's algorithm) that can solve the integer factorization and discrete

logarithm problems in polynomial time. Fortunately, cryptosystems have been devised that can serve as secure replacements for many of these applications (Augot et al., 2015). While they are not widespread in common use today, these algorithms and algorithms like these will undoubtedly grow in usage as the threat posed by quantum computing continues to grow.

Supersingular Isogeny Diffie-Hellman

Supersingular isogeny Diffie-Hellman (SIDH) is a modification of the Diffie-Hellman key-exchange protocol that operates on supersingular elliptic curves (De Feo, Jao, & Plût, 2011). Rather than operating on elements of a finite cyclic group or an elliptic curve group as is the case in the traditional DH and ECDH algorithms respectively, SIDH uses *isogenies* of elliptic curves to allow two parties to derive a shared secret key over an insecure channel in a manner that preserves the security of the key even when an eavesdropper has access to quantum computing resources (De Feo, Jao, & Plût, 2011).

The algorithm. In order to begin the supersingular isogeny Diffie-Hellman protocol between two parties wishing to calculate a shared secret (hereafter referred to as Alice and Bob), a number of parameters must be initially mutually agreed upon. First, Alice and Bob must agree on a prime p that is of the following form for some small primes ℓ_A, ℓ_B:

$$p = \ell_A^{e_A} \ell_B^{e_B} \cdot f \pm 1$$

f is chosen so that p is prime. Alice and Bob then select a field $\mathbb{F}_{p^2}$ of order p^2 and agree on a supersingular elliptic curve E over $\mathbb{F}_{p^2}$. Furthermore, Alice and Bob agree on base

points P_A, Q_A, P_B, Q_B on the elliptic curve – P_A, Q_A have order $\ell_A^{e_A}$ and P_B, Q_B have order $\ell_B^{e_B}$.

Alice now computes a random $\mathbb{Z}_{\ell_A^{e_A}}$-linear combination of P_A, Q_A. Alice uses this point as the generator of the kernel of a secret elliptic curve isogeny ϕ_A and completes her step of the algorithm by sending Bob $\phi_A(E)$, $\phi_A(P_B)$, and $\phi_A(Q_B)$. Bob performs the analogous calculation of a secret isogeny ϕ_B using a random $\mathbb{Z}_{\ell_B^{e_B}}$-linear combination of P_B, Q_B and completes the process by sending $\phi_B(E)$, $\phi_B(P_A)$, and $\phi_B(Q_A)$ to Alice (De Feo, Jao, & Plût, 2011).

In the final step of the algorithm, Alice computes the $\mathbb{Z}_{\ell_A^{e_A}}$-linear combination of $\phi_B(P_A)$ and $\phi_B(Q_A)$ using the coefficients chosen in her initial step of computing ϕ_A. Alice then uses this elliptic point as the generator of the kernel of a second isogeny ϕ_A' while Bob uses the same procedure to calculate ϕ_B' based on the points sent to him by Alice (De Feo, Jao, & Plût, 2011). The properties of isogeny graphs of supersingular elliptic curves cause Alice and Bob to obtain the same elliptic curve E_{AB} through the following calculation:

$$E_{AB} = \phi_A'\big(\phi_B(E)\big) = \phi_B'(\phi_A(E))$$

Alice and Bob may then take the j-invariant of E_{AB} as a shared secret key (De Feo, Jao, & Plût, 2011).

Security properties of the algorithm. The supersingular isogeny Diffie-Hellman key exchange protocol is structured very similarly to the standard Diffie-Hellman protocol but is considered to be secure even against an attacker in possession of a

quantum computer. In the course of the protocol, Alice and Bob must first agree on a common choice of initial elliptic curve and basis points, but the only information exchanged between Alice and Bob following the initialization of the protocol is the images of the elliptic curve E under the secret isogenies ϕ_A, ϕ_B, along with the images of the basis points under the opposing isogenies. In order to discover E_{AB} and thus the shared secret, the attacker must determine one of the secret isogenies given only E and its image. De Feo, Jao, and Plût (2011) note that this operation is difficult even on a quantum computer – specifically, that there is no known sub-exponential algorithm for this procedure for classical or quantum computers. Thus, with elliptic curves of sufficient size, the SIDH key exchange is believed to be suitable for use even in environments where quantum computing attacks are possible (De Feo, Jao, & Plût, 2011).

NTRU

NTRU (short for *N-th degree TRUncated polynomial ring*) is a formerly proprietary lattice-based asymmetric-key encryption scheme developed by researchers at Brown University in the late 1990s (Hoffstein, Pipher, & Silverman, 1998). NTRU is a fast alternative to cryptosystems such as RSA and operates on elements of polynomial quotient rings – the algorithm is strongly related to the shortest vector problem for lattices, which is not known to admit an efficient classical or quantum solution (Hoffstein, Pipher, & Silverman, 1998).

The algorithm. NTRU is formally a *family* of cryptosystems with variable parameters specifying aspects of the encryption and decryption process. Four subsets ($\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_m, \mathcal{L}_r$) of the polynomial quotient ring $\mathbb{Z}[x]/(x^n - 1)$ for some large prime $n \in$

$\mathbb{N}$ must be specified. Additional parameters to the cryptosystem include relatively prime integers p, q with $q \gg p$ (Hoffstein, Pipher, & Silverman, 1998).

An NTRU private key is a random polynomial $f \in \mathcal{L}_f$, but it must meet the condition that $f_p = f^{-1} \pmod{p}$ and $f_q = f^{-1} \pmod{q}$ exist. f_p is typically stored alongside f, although it is easily computable from f using the extended Euclidean algorithm. The corresponding NTRU public key is computed from f by randomly selecting $g \in \mathcal{L}_g$ and calculating $h = g \cdot f_q \pmod{q}$. h is the public key and may be published (Hoffstein, Pipher, & Silverman, 1998).

Plaintext messages in NTRU are represented as elements $m \in \mathcal{L}_m$. The encoding of a message from binary data or another format to the polynomial m will be dependent on $\mathcal{L}_m$. To encrypt m by the public key h, a random secret *hiding* polynomial $r \in \mathcal{L}_r$ is chosen and the ciphertext e is computed by (Hoffstein, Pipher, & Silverman, 1998):

$$e = prh + m \pmod{q}$$

To decrypt the message using the private key, a few intermediate operations are required. Observe that:

$$\begin{aligned} fe \pmod{q} &= f \cdot (prh + m) \pmod{q} && (e = prh + m \pmod{q}) \\ &= f \cdot \left(prgf_q + m\right) \pmod{q} && (h = gf_q \pmod{q}) \\ &= ff_q prg + fm \pmod{q} \\ &= prg + fm \pmod{q} && (ff_q = 1 \pmod{q}) \end{aligned}$$

Thus, let $a = fe \pmod{q}$. We therefore see that $a = prg + fm \pmod{q}$. Then, by computing $b = a \pmod{p}$, we have $b = fm \pmod{p}$ as $prg \equiv 0 \pmod{p}$. Then, simply observe that:

$$bf_p = ff_p m \pmod{p} = m \pmod{p}$$

as $ff_p \equiv 1 \pmod{p}$. Thus, by taking the ciphertext and multiplying by f modulo q, then multiplying by f_p modulo p, the plaintext will be recovered (Hoffstein, Pipher, & Silverman, 1998). For some choices of parameters to the NTRU cryptosystem, there is the possibility that certain combinations of public keys and plaintexts will yield ciphertexts that cannot be decrypted correctly due to information loss via the taking of residues modulo p and q. By correctly choosing $q \gg p$ and using coefficients of a in the interval $[\frac{-q}{2}, \frac{q}{2}]$ (rather than $[0, q-1]$) during the calculation, the original message is guaranteed to be recovered correctly (Hoffstein, Pipher, & Silverman, 1998).

Security properties of the algorithm. While NTRU has not been the subject of as intense cryptographic analysis as many other cryptosystems, many combinations of NTRU parameters are currently believed to be secure and are not known to admit any substantial cryptanalytic attacks. In terms of security against quantum computer attacks, reversing the NTRU encryption without possession of one of the private key polynomials is closely related to the lattice shortest vector problem. Variants of this problem are considered to be NP-hard and therefore the lattice shortest vector problem is generally considered to be computationally infeasible to solve directly even with the aid of quantum computing resources (Hoffstein, Pipher, & Silverman, 1998). Thus, with appropriate choices of parameters, NTRU is a reasonable replacement for asymmetric-key algorithms such as RSA even under the widespread adoption of quantum computer-aided cryptanalytic techniques (Augot et al., 2015).

Conclusion

The landscape of cryptography is changing quickly in response to the formidable threat posed by the advent of strong quantum computing technology. Because cryptography is only as useful as it is secure, it is critical that the encryption technologies currently in use across the information security and broader computing industries are evaluated so that their weaknesses against quantum-computer-related attacks can be well understood and thus mitigated.

While quantum computers had long been a purely theoretical concept, new leaps in engineering technology and ability have enabled the construction of small quantum computers. The majority of these machines are far too small and limited to pose any reasonable threat to any form of cryptography that is actually in modern practice. However, newer, larger, and more powerful quantum computing systems are continually in development and the day will undoubtedly come soon that this technology possesses the ability to mount a real attack against real cryptography. At that time, it will be crucial that algorithms capable of replacing older, vulnerable cryptographic techniques be well-studied and well-tested. This will allow the transition into the world of post-quantum cryptography to take place seamlessly and with minimal disruption to the computing industry. In a world where the global economy has become increasingly and inextricably linked with computing, information security, and ultimately cryptography, it is critical to pursue these avenues of research so that the principles of strong information security, data integrity, and human privacy can be protected in the midst of a rapidly changing future.

References

Amin, M. H. (2015). Searching for quantum speedup in quasistatic quantum annealers. *Physical Review A, 92*(5). Retrieved from https://arxiv.org/pdf/1503.04216.pdf

Augot, D., Batina, L., Bernstein, D. J., Bos, J., Buchmann, J., Castryck, W., . . . Yang, B. (2015). Initial recommendations of long-term secure post-quantum systems. *PQCRYPTO. EU. Horizon, 2020*. Retrieved from http://pqcrypto.eu.org/docs/initial-recommendations.pdf

Barker, E. (2016). Recommendation for key management – part 1: general. *NIST Special Publication 800-57 Part 1*. doi:10.6028/NIST.SP.800-57pt1r4

De Feo, L., Jao, D., & Plût, J. (2011). Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto 2011: Proceedings of the International Conference on Post-Quantum Cryptography, 2011 Conference*, 19-34. doi:10.1007/978-3-642-25405-5_2

Dierks, T., & Rescorla, E. (2008). The transport layer security (TLS) protocol version 1.2. *IETF RFC 5246*. doi:10.17487/RFC5246

Diffie, W., & Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory, 22*(6), 644-654. doi:10.1109/TIT.1976.1055638

D-Wave Systems. (2017). *D-Wave announces upgrades to D-Wave 2000Q quantum computer* [Press release]. Retrieved from https://www.dwavesys.com/press-releases/d-wave-announces%C2%A0upgrades-d-wave-2000q-quantum-computer

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search.

Proceedings, 28th Annual ACM Symposium on the Theory of Computing, May 1998, 212-219. Retrieved from https://arxiv.org/abs/quant-ph/9605043

Gil, D. (2017, Nov 10). The future is quantum. *IBM Research*. Retrieved from https://www.ibm.com/blogs/research/2017/11/the-future-is-quantum/

Güneysu, T., Kasper, T., Novotný, M., Paar, C., & Rupp, A. (2008). Cryptanalysis with COPACOBANA. *IEEE Transactions on Computers, 57*(11). doi:10.1109/TC.2008.80

Hoffstein, J., Pipher, J., Silverman, J. H. (1998). NTRU: A ring-based public key cryptosystem. *International Algorithmic Number Theory Symposium 1998,* 267-288. doi:10.1007/BFb0054868

Lee, C. (2017, Jan 26). Explaining the upside and downside of D-Wave's new quantum computer. *Ars Technica*. Retrieved from https://arstechnica.com/science/2017/01/explaining-the-upside-and-downside-of-d-waves-new-quantum-computer/

Pomerance, C. (1996). A tale of two sieves. *Notices of the AMS, 43*(12), 1473-1485. Retrieved from http://www.ams.org/notices/199612/pomerance.pdf

Rieffel, E., & Polak, W. (2000). An introduction to quantum computing for non-physicists. *ACM Computing Surveys, 32*(3), 300-335. doi:10.1145/367701.367709

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM, 21*(2), 120-126. doi:10.1145/359340.359342

Schneier, B. (1996). *Applied cryptography: Protocols, algorithms, and source code in C*

(2nd ed.). Hoboken, NJ: Wiley.

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing, 26*(5), 1484-1509. doi:10.1137/S0097539795293172