Contents

I Introduction

I.A Problem Statement

I.B Motivation

I.C Scope of the Paper

II Background/Intro

The emergence of quantum computers into the minds of mainstream media will lead towards a shift in how computation is thought about and done from now into the future. Quantum computers are like regular computers except instead of storing their informations a 1 or a 0, they store quantum q-bits, which can represent a wide variety of complex states. Using complex and specific arrangements of these q-bits and logic gates, one can perform operations that can be very fast, and can break common number theory problems like the discrete logarithm problem, and the factorization of a large number. By completing these tasks efficiently, they can undermine the security of many modern crypto systems we use today.

Since security is always a back and forth, arms race between the attacker and the defender, the next step is to create a new generation of post-quantum cryptographic algorithms, that are resistant to attacks by quantum computers. Thankfully, some algorithms we have in place already, such as symmetric key cryptography can be very secure against quantum computers given a large key size. Furthermore, NIST, the National Institute of Standards and Technology, has standardized algorithms that help secure and are resistant to threats in a quantum-computing world. Standardizing things like Key Encapsulation Mechanisms (KEM)‘s to make keys secure, and using special Digital Signature Algorithms to authenticate who you are comminating with.

There is already work being done to implement many of these systems into real world use cases, as we know, there is never a one size fits all solution to cryptography. A big barrier to widespread post-quantum crypto adoption are the infrastructure/hardware costs of implementing these new algorithms. Since modern quantum computers aren’t and won’t be on the scale to break any security for years to come, it might seem like there is no need to rush or worry, however, attackers can employ strategies such as harvest now, decrypt later, which could mean sensitive

information not encrypted properly even today could be at risk. Not all is lost, as Google and other companies have already started testing and experiments changing algorithms over to post-quantum resistant ones with good amounts of success.

This paper goes over the current landscape of quantum computers, and its relation to networking and security, understanding how they operate and break modern cryptography, as well as the current successes and barriers to keeping information safe in a post-quantum world.

II.A So what are Quantum Computers?

Starting at the very core of a computer we find the key difference between regular computers and quantum computers that causes the differences everywhere else. In classical computers, the smallest unit of storage and computation is a bit. Either 1 or 0, on or off. In quantum computers their smallest unit is called a q bit. (Chae et al., 2024)

Just like a regular bit can be in a state of 0 or 1, the two possible states a Q-bit can be in are $|0\rangle$ and $|0\rangle$. This "$|\ \ \rangle$" is called a *ket* or Dirac notation, and is commonly used to describe q-bits and their state (Nielsen & Chuang, 2010). What makes the q-bits different than regular bits is that they can exist as linear combinations (Nielsen & Chuang, 2010) of these two states $|0\rangle$ and $|1\rangle$, such as

$$|x\rangle = \alpha|0\rangle + \beta|1\rangle$$

.

A good way of understanding these Q-bits is to understand them similar to how we understand probabilistic algorithms. In probabilistic algorithms, instead of being in a specified state, the algorithm describes a probabilistic distribution which lays out the possible states the system can be in, and through steps, the chance of bein in each state changes (Shor, 1997).

Q-Bits act similarly. Instead of being in a state of 1 or 0, their linear combination of $|0\rangle$ and $|1\rangle$ helps define the probability of being in the $|0\rangle$ or $|1\rangle$ state when measured.

Diving in further, we can look into the common analogy of Schrodinger cat. In the experiment, a cat is in a sealed box with a bottle of some poisonous gas. After a certain amount of time, there is a 50% chance that the bottle gets broken (usually by measuring random radioactivity), releasing the

gas and killing the cat. It is almost as if the cat is both dead and alive, 50% dead, 50% chance alive, until we are able to open the box and confirm which one is the case.

By observing or measuring the system, we force it to collapse to one of the two final states. You can thing about q-bits in this same way where measurement of the q-bit collapses the bit "With a probability determined by the squared magnitude of the coefficients a and b in the superposition" (Chae et al., 2024).

Because of this collapse, you can think of all possible states the q-bit could be in as points along a sphere, all one unit away from the origin, and when you measure the q-bit, it will collapse into either $|0\rangle$ or $|1\rangle$.

So regardless of where the metaphorical arrow points on the circle, we will only get a measurement of 0 or 1 probabilistically, meaning, further steps are needed to make guaranteed outputs which is where logic gates come into play.

II.B Computing Gates

In classical computing we manipulate bits by passing them through gates that act like functions. By combining these gates, we can create complex behaviors such as adders, flip-flops, all the way up to the hardware structure found inside modern computers. Quantum computing follows a similar idea of applying operations on our quantum bits by passing them through *quantum gates*, however in the world of quantum, we need to take into account the probabilistic nature of the q-bits superposition.

The goal of quantum gates is to transform q-bits such that when we measure them at the end of our series of gates, the output contains useful information we can make judgments based on. Depending on the algorithm, this could mean manipulating our bit so that its final state measures as $|0\rangle$ or $|1\rangle$ with near-certainty, or taking repeating measurements of our q-bit after a set of operations, to "sample" from the probability distribution the q-bit represents.

One example of how powerful the right combination of these quantum gates is the example of Grover's algorithm. Grover's algorithm applies a clever sequence of these gates to improve runtime of the the "Database search" problem. In this problem one is searching through a database of N items for a single marked item. On classical computers, you have to search through the database one

by one. The marked item could be the first you check or the last. On average you have to search over $\frac{N}{2}$ items to find your marked item. Since this scales polynomially with a factor of N we give it O(N) polynomial runtime. Grover's Algorithm uses quantum gates to reduce this problem to O($\sqrt{n}$), which is a quadratic speed up over the classical algorithm (Grover, 1996).

II.C Cryptographic Theory

In 1994 Peter Shor showed one of the most famous applications of quantum gates and quantum computing: solving "Two number theory problems which have been studied extensively but for which no polynomial-time algorithms have yet been discovered are finding discrete logarithms and factoring integers" (Shor, 1997).

Because there are no polynomial-time methods to solve these problems, if we increase the size of the input, the complexity grows exponentially making large examples of these problems computationally infeasible on modern computers.

For example, RSA as its security, bases its strength off of the inability to factor large integers, and Diffie-Hellman key exchange bases its strength on the discrete logarithm problem. Shor showed that these problems can be solved in polynomial time on a quantum computer which effectively breaks the security assumptions of their security (Shor, 1997).

To illustrate, consider RSA. The system defines two types of keys: a public key (N, e) and a private key (p, q, d) where p and q are large prime factors of N.

$$\text{pk} = (N, e)$$

$$\text{sk} = (p, q, d)$$

To encrypt our message m we compute our cipher text c as follows.

$$c \coloneqq m^e \bmod N$$

then to decrypt c back into m we compute

$$m \coloneqq c^d \bmod N$$

Further more, e and d are modular inverses of each other in a new modulus $\varphi(N)$. This $\varphi(N)$ is also directly related to p and q which are our 2 big factors of N.

$$e \cdot d = 1 \bmod \varphi(N)$$

$$(p-1)(q-1) = \varphi(N)$$

We could find d if we knew $\varphi(N)$ by using the extended euclidean algorithm to compute the modular inverse. So we need to find the factors $p * q = N$

Because $\varphi(N)$ can be directly calculated based on the factors p and q, an attacker who can efficiently factor N can compute $\varphi(N)$ and with that compute the private key d (by using the extended euclidean algorithm to compute the modular inverse).

II.C.1 Prime Factorization

So how do we find our factors p and q? The best approach on modern computers is to use brute force and guess and check,

Lets start with an initial take a guess for a factor of N called g. If $\gcd(g, N) \neq 1$ we already found a non-trivial factor of N. If not, lets try another guess. Shor's insight is a process of creating better guesses for factors of N. This problem of finding a better guess can be reduced down to the problem of order finding: a problem quantum computers can solve in polynomial time (Amico et al., 2019).

It all starts with this equation.

$$g^r = 1 \bmod N$$

This r is called the order, or the period, of this modular base N, because, as we multiply g by itself, we get a sequence of numbers mod N. What this equation is saying is that these powers of g will repeat after r successive multiplication.

Since g and N are co-prime we know that there must exist some r such that the equation is true. (Or we could say since $g^0 = 1$ and there are only N possible values, by the pidgenhole principle they must repeat, and therefore 1 must repeat.) (OR For every prime p, the multiplicative group (mod p) is cyclic,(Hardy & Wright, 2008)) Finding this particular r is the computational order solving problem, and it turns out quantum computers are great at solving it.

The equation from the order solving problem is used to create two better guesses for our prime factors by factoring the left side into two numbers as follows.

$$g^r = 1 \bmod N$$

$$\left(g^{\frac{r}{2}} + 1\right) * \left(g^{\frac{r}{2}} - 1\right) = 0 \bmod N$$

This expression appears like two numbers multiply to N! The key problem is computing r. This is where quantum computing gates can be applied.

Shor's algorithm uses a quantum logic gate called the Quantum Fourier Transform. When this gate is applied over multiple q-bits that represent the powers of g. This gate extract the "order", our r value, with high probability. Once we get this r value, we still need to run a quick sanity check. r needs to be even if we plan to raise $g^{\frac{r}{2}}$, and we also need to check that $g^{\frac{r}{2}} \pm 1$, isn't just some multiple of N (Amico et al., 2019). If we are trying to factor N, a guess of $2N$ isn't very helpful at all! By repeating those steps till we get a r that satisfies those conditions, we have created a number that shares factors with N, which we can use to break the RSA Crypto System.

II.C.2 Discrete Log Problem

Similar to integer factorization, the discrete logarithm problem is a hard mathematical problem that forms the backbone of certain crypto systems such as the Diffie-Hellman key exchange. The discrete logarithm problem is the problem of finding an exponent k such that $b^k = a$ in some $\bmod m$.

By choosing your modulo "group" effectively, the difficulty of solving this equation for k is strong enough that public-key encryption algorithms like ElGamal (which is based on Diffie-Hellman) base their security on its hardness (Menezes et al., 1997).

One might observe that many of the steps in our previous section look like the discrete log problem—so it's no surprise that Shor was also able to break the discrete logarithm problem with quantum computers in a similar way.

The fastest algorithm known for finding discrete logarithms is discussed in (Gordon, 1993) which runs sub-exponential time, which is good, but not polynomial like in (Shor, 1997) that showed how to find discrete logarithms on a quantum computer with two modular exponentiations and two quantum Fourier transform.

II.D Quantum Computers

II.D.1 Background

II.D.2 Cryptographic Applications

III Current Landscape

III.A Families of Post Quantum Algorithms

III.A.1 Algorithms that Aren't Broken

While it is true that with a sufficiently powerful computer, one could break algorithms whose security depends on the difficulty of factoring large integers or solving the discrete logarithm problem, such as RSA, we already have other algorithms that are not yet known to be broken by quantum algorithms (Bernstein, 2009). In fact, symmetric algorithms, or secret key cryptography, are generally accepted to not really be affected by quantum computers. Though there will be a need to increase the key size (which is more or less true across the board), today's best secret key algorithms will seem to also be the best secret key algorithms in a quantum world (Bernstein, 2009). Work has been done to show exactly how much of an impact quantum computing will have on AES, for example, and while quantum computers will still do better than classical ones in theory, AES remains a useful post quantum algorithm, given an increase in key size (Jang et al., 2022).

List of asymmetric algorithms that are (or can be made to be with large enough key sizes) quantum resistant:

- Merkle's hash-tree public-key signature system
- McEliece's hidden-Goppa-code public-key encryption system
- Hoffstein–Pipher–Silverman "NTRU" public-key-encryption system
- Patarin's HFE^{v-} public-key-signature system

All of the above are from prior to the year 2000 (Bernstein, 2009).

List of new post quantum algorithms:

- CRYSTALS-KYBER
  - Learning-with-errors module lattice based (Regev, 2009)
  - encryption system

- CRYSTALS-DILITHIUM
  - Fiat-Shamir with aborts (Lyubashevsky, 2009)
  - signature system

III.A.2 New Developments

III.B NIST Standards

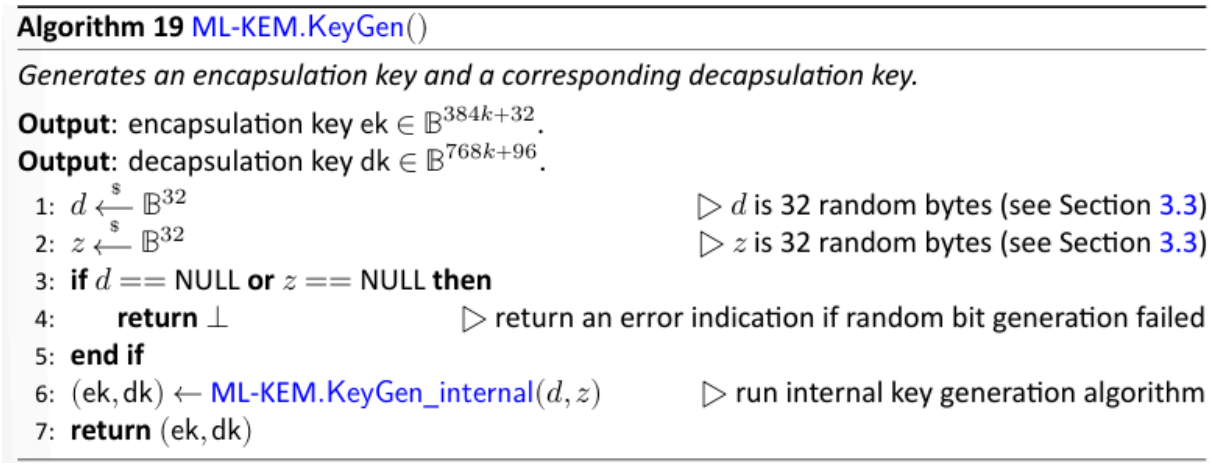
The future of technology is in quantum computing and with that comes many security risks to the cryptosystems already in place such as public key cryptography and digital signatures. These current systems rely on mathematical schemes, such as integer factorization or discrete logs, that are difficult to break but with quantum computing could be broken. In 2016 the National Institute of Standards and Technology (NIST) established a public post quantum cryptography standardization process where other people and companies can suggest and propose new algorithms for public key cryptography and digital signatures that are quantum computing resistant. In August 13, 2024, NIST finally published three Federal Information Processing Standards (FIPS) regarding post quantum cryptography with these proposed new algorithms. The three new standards are specifically about new security algorithms for public key cryptography and digital signatures.

III.B.1 FIPS 203

The first of three standards is FIPS 203 which is a module lattice based key encapsulation mechanism standard (ML-KEM). A key encapsulation mechanism (KEM) is a "set of algorithms that under certain conditions can be used by two parties to establish a shared secret key over a public channel" (Standards & Technology, 2024a). This key along with a symmetric key cryptographic algorithm can be used to secure communication with encryption and authentication. There are three algorithms used in KEM, the probabilistic key generation algorithm, the probabilistic encapsulation algorithm, and the deterministic decapsulation algorithm. The standard denotes the algorithms by KeyGen, Encaps, and Decaps respectively.

The KeyGen algorithm produces both an encapsulation key and a decapsulation key using complete randomness internally. The encapsulation key is made public and the decapsulation key is kept private for the user running the KeyGen algorithm. The KeyGen algorithm can be seen in Figure 1. Other users of the public key can perform seed consistency, encapsulation key,

decapsulation key, or pair-wise consistency checks, but none of them "guarantee that the key pair was properly generated" (Standards & Technology, 2024a).



Algorithm 19 ML-KEM.KeyGen()

Generates an encapsulation key and a corresponding decapsulation key.

Output: encapsulation key ek $\in \mathbb{B}^{384k+32}$.
Output: decapsulation key dk $\in \mathbb{B}^{768k+96}$.

1: $d \xleftarrow{\$} \mathbb{B}^{32}$ ▷ d is 32 random bytes (see Section 3.3)
2: $z \xleftarrow{\$} \mathbb{B}^{32}$ ▷ z is 32 random bytes (see Section 3.3)
3: **if** $d ==$ NULL **or** $z ==$ NULL **then**
4: **return** $\perp$ ▷ return an error indication if random bit generation failed
5: **end if**
6: $(\text{ek}, \text{dk}) \leftarrow$ ML-KEM.KeyGen_internal(d, z) ▷ run internal key generation algorithm
7: **return** (ek, dk)

Figure 1: ML_KEM KeyGen Algorithm (Standards & Technology, 2024a)

Basically, the algorithm is creating two random strings of 32 bytes and passing that into the KeyGen_internal algorithm which then outputs the encapsulation key, denoted ek, and the decapsulation key, denoted dk. The KeyGen_internal involves the K-PKE.KeyGen algorithm which is a much longer and more in-depth algorithm. The algorithm takes one of the random 32-byte strings and outputs what will eventually become the decapsulation key. PKE is a public key encryption scheme which is an asymmetric encryption scheme to send secret data over a public channel. Both PKE and KEM are used because the PKE is not approved to be a stand-alone scheme for protection against quantum cryptography. The PKE KeyGen algorithm also uses randomness to generate the encapsulation and decapsulation keys and the algorithm can be seen in Figure 2.

Algorithm 13 K-PKE.KeyGen(d)

Uses randomness to generate an encryption key and a corresponding decryption key.

Input: randomness $d \in \mathbb{B}^{32}$.
Output: encryption key $\text{ek}_{\text{PKE}} \in \mathbb{B}^{384k+32}$.
Output: decryption key $\text{dk}_{\text{PKE}} \in \mathbb{B}^{384k}$.

$(\rho, \sigma) \leftarrow G(d\|k)$ ▷ expand 32+1 bytes to two pseudorandom 32-byte seeds[1]
$N \leftarrow 0$
for $(i \leftarrow 0; i < k; i$++$)$ ▷ generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$
  for $(j \leftarrow 0; j < k; j$++$)$
    $\hat{\mathbf{A}}[i, j] \leftarrow \text{SampleNTT}(\rho\|j\|i)$ ▷ j and i are bytes 33 and 34 of the input
  end for
end for
for $(i \leftarrow 0; i < k; i$++$)$ ▷ generate $\mathbf{s} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{s}[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$ ▷ $\mathbf{s}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
for $(i \leftarrow 0; i < k; i$++$)$ ▷ generate $\mathbf{e} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{e}[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(\sigma, N))$ ▷ $\mathbf{e}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
$\hat{\mathbf{s}} \leftarrow \text{NTT}(\mathbf{s})$ ▷ run NTT k times (once for each coordinate of $\mathbf{s}$)
$\hat{\mathbf{e}} \leftarrow \text{NTT}(\mathbf{e})$ ▷ run NTT k times
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$ ▷ noisy linear system in NTT domain
$\text{ek}_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{\mathbf{t}})\|\rho$ ▷ run ByteEncode_{12} k times, then append $\hat{\mathbf{A}}$-seed
$\text{dk}_{\text{PKE}} \leftarrow \text{ByteEncode}_{12}(\hat{\mathbf{s}})$ ▷ run ByteEncode_{12} k times
return $(\text{ek}_{\text{PKE}}, \text{dk}_{\text{PKE}})$

Figure 2: K_PKE KeyGen Algorithm (Standards & Technology, 2024a)

The Encaps algorithm uses the encapsulation key generated from the KeyGen algorithm and generates a copy of the shared public key and ciphertext using randomness. The user using the public encapsulation key is the one who runs this algorithm, but they must first check the encapsulation key. Once that has been done, they can run the Encaps algorithm which can be seen in Figure 3.

Algorithm 20 ML-KEM.Encaps(ek)

Uses the encapsulation key to generate a shared secret key and an associated ciphertext.

Checked input: encapsulation key ek $\in \mathbb{B}^{384k+32}$.
Output: shared secret key $K \in \mathbb{B}^{32}$.
Output: ciphertext $c \in \mathbb{B}^{32(d_u k+d_v)}$.

1: $m \xleftarrow{\$} \mathbb{B}^{32}$ ▷ m is 32 random bytes (see Section 3.3)
2: **if** $m ==$ NULL **then**
3: **return** $\perp$ ▷ return an error indication if random bit generation failed
4: **end if**
5: $(K, c) \leftarrow$ ML-KEM.Encaps_internal(ek, m) ▷ run internal encapsulation algorithm
6: **return** (K, c)

Figure 3: ML_KEM Encaps Algorithm (Standards & Technology, 2024a)

The algorithm creates a random string of 32 bytes and passes that into the Encaps_internal algorithm which then outputs the shared secret key, denoted K, and the ciphertext, denoted c, using the checked encapsulation key from the KeyGen algorithm. The random string becomes the plaintext which after passing through the algorithm becomes the ciphertext. The Encaps_internal involves the K-PKE.Encrypt algorithm which takes the plaintext string, the checked encapsulation key and another random string. It then outputs what will eventually become the ciphertext. The K-PKE.Encrypt algorithm can be seen in Figure 4.

Algorithm 14 K-PKE.Encrypt$(\text{ek}_{\text{PKE}}, m, r)$

Uses the encryption key to encrypt a plaintext message using the randomness r.

Input: encryption key $\text{ek}_{\text{PKE}} \in \mathbb{B}^{384k+32}$.
Input: message $m \in \mathbb{B}^{32}$.
Input: randomness $r \in \mathbb{B}^{32}$.
Output: ciphertext $c \in \mathbb{B}^{32(d_u k + d_v)}$.

$N \leftarrow 0$
$\hat{\mathbf{t}} \leftarrow \text{ByteDecode}_{12}(\text{ek}_{\text{PKE}}[0:384k])$ ▷ run ByteDecode$_{12}$ k times to decode $\hat{\mathbf{t}} \in (\mathbb{Z}_q^{256})^k$
$\rho \leftarrow \text{ek}_{\text{PKE}}[384k:384k+32]$ ▷ extract 32-byte seed from ek_{PKE}
for ($i \leftarrow 0$; $i < k$; i++) ▷ re-generate matrix $\hat{\mathbf{A}} \in (\mathbb{Z}_q^{256})^{k \times k}$ sampled in Alg. 13
  for ($j \leftarrow 0$; $j < k$; j++)
    $\hat{\mathbf{A}}[i,j] \leftarrow \text{SampleNTT}(\rho \| j \| i)$ ▷ j and i are bytes 33 and 34 of the input
  end for
end for
for ($i \leftarrow 0$; $i < k$; i++) ▷ generate $\mathbf{y} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{y}[i] \leftarrow \text{SamplePolyCBD}_{\eta_1}(\text{PRF}_{\eta_1}(r, N))$ ▷ $\mathbf{y}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
for ($i \leftarrow 0$; $i < k$; i++) ▷ generate $\mathbf{e_1} \in (\mathbb{Z}_q^{256})^k$
  $\mathbf{e_1}[i] \leftarrow \text{SamplePolyCBD}_{\eta_2}(\text{PRF}_{\eta_2}(r, N))$ ▷ $\mathbf{e_1}[i] \in \mathbb{Z}_q^{256}$ sampled from CBD
  $N \leftarrow N + 1$
end for
$e_2 \leftarrow \text{SamplePolyCBD}_{\eta_2}(\text{PRF}_{\eta_2}(r, N))$ ▷ sample $e_2 \in \mathbb{Z}_q^{256}$ from CBD
$\hat{\mathbf{y}} \leftarrow \text{NTT}(\mathbf{y})$ ▷ run NTT k times
$\mathbf{u} \leftarrow \text{NTT}^{-1}(\hat{\mathbf{A}}^\top \circ \hat{\mathbf{y}}) + \mathbf{e_1}$ ▷ run NTT^{-1} k times
$\mu \leftarrow \text{Decompress}_1(\text{ByteDecode}_1(m))$
$v \leftarrow \text{NTT}^{-1}(\hat{\mathbf{t}}^\top \circ \hat{\mathbf{y}}) + e_2 + \mu$ ▷ encode plaintext m into polynomial v
$c_1 \leftarrow \text{ByteEncode}_{d_u}(\text{Compress}_{d_u}(\mathbf{u}))$ ▷ run ByteEncode$_{d_u}$ and Compress$_{d_u}$ k times
$c_2 \leftarrow \text{ByteEncode}_{d_v}(\text{Compress}_{d_v}(v))$
return $c \leftarrow (c_1 \| c_2)$

Figure 4: K_PKE Encrypt Algorithm (Standards & Technology, 2024a)

The Decaps algorithm uses the decapsulation key generated from the KeyGen algorithm and generates a copy of the shared secret key. The user using the private decapsulation key is the one who runs this algorithm, but they must first check the decapsulation key and the ciphertext. Once that has been done, they can run the Decaps algorithm which can be seen in Figure 5.

Algorithm 21 ML-KEM.Decaps(dk, c)

Uses the decapsulation key to produce a shared secret key from a ciphertext.

Checked input: decapsulation key $\text{dk} \in \mathbb{B}^{768k+96}$.
Checked input: ciphertext $c \in \mathbb{B}^{32(d_u k+d_v)}$.
Output: shared secret key $K \in \mathbb{B}^{32}$.

1: $K' \leftarrow$ ML-KEM.Decaps_internal(dk, c) ▷ run internal decapsulation algorithm
2: **return** K'

Figure 5: ML_KEM Decaps Algorithm (Standards & Technology, 2024a)

The algorithm takes the checked decapsulation key and ciphertext and passes them into the Decaps_internal algorithm which then outputs a shared secret key, denoted K'. The Decaps_internal involves the K-PKE.Decrypt algorithm which takes the decryption key, and the checked ciphertext. It then outputs what will eventually become the shared secret key. This algorithm also uses the K_PKE.Encrypt algorithm once again to check the correctness of the ciphertext. The K-PKE.Decrypt algorithm can be seen in Figure 6.

Algorithm 15 K-PKE.Decrypt$(\text{dk}_{\text{PKE}}, c)$

Uses the decryption key to decrypt a ciphertext.

Input: decryption key $\text{dk}_{\text{PKE}} \in \mathbb{B}^{384k}$.
Input: ciphertext $c \in \mathbb{B}^{32(d_u k+d_v)}$.
Output: message $m \in \mathbb{B}^{32}$.

1: $c_1 \leftarrow c[0 : 32d_u k]$
2: $c_2 \leftarrow c[32d_u k : 32(d_u k + d_v)]$
3: $\mathbf{u}' \leftarrow \text{Decompress}_{d_u}(\text{ByteDecode}_{d_u}(c_1))$ ▷ run Decompress_{d_u} and ByteDecode_{d_u} k times
4: $v' \leftarrow \text{Decompress}_{d_v}(\text{ByteDecode}_{d_v}(c_2))$
5: $\hat{\mathbf{s}} \leftarrow \text{ByteDecode}_{12}(\text{dk}_{\text{PKE}})$ ▷ run ByteDecode_{12} k times
6: $w \leftarrow v' - \text{NTT}^{-1}(\hat{\mathbf{s}}^{\top} \circ \text{NTT}(\mathbf{u}'))$ ▷ run NTT k times; run NTT^{-1} once
7: $m \leftarrow \text{ByteEncode}_1(\text{Compress}_1(w))$ ▷ decode plaintext m from polynomial v
8: **return** m

Figure 6: K_PKE Decrypt Algorithm (Standards & Technology, 2024a)

The steps to run all these algorithms between two parties, let's call Alice and Bob, are as follows. Alice runs the KeyGen algorithm to produce a decapsulation key, which Alice keeps private, and an encapsulation key, which is made public and available to Bob. Bob now takes this encapsulation key and runs the Encaps algorithm to generate a copy of the shared secret key along with the associated ciphertext. Bob sends the ciphertext to Alice who uses it as well as the decapsulation key to run the Decaps algorithm. Alice now has her own copy of the shared secret key

and assuming the two copies are the same, Alice and Bob can now communicate and send information privately and be secure from quantum computers.

There are three parameter sets for ML-KEM, those being ML_KEM-512, ML_KEM-768, and ML_KEM-1024. They are in order of increasing security strength but decreasing efficiency. 512 is defined as security category 1, 768 as security category 3, and 1024 as security category 5, which just means higher security as category number increases. 512 has a decapsulation failure rate of 2^(-138.8), 768 of 2^(-164.8), and 1024 of 2^(-174.8). It is a trade-off between security and performance and since it is available to both private and commercial organizations the scope is broad in terms of whether security or performance is preferred. ML_KEM can be implemented in hardware, firmware, or software and is based on the CRYSTALS-Kyber algorithm mentioned earlier. As of now ML-KEM is secure against quantum computers.

III.B.2 FIPS 204

The second standard is FIPS 204 which is a module lattice based digital signature algorithm (ML-DSA). A digital signature algorithm (DSA) is a "set of algorithms that can be used to generate and verify digital signatures" (Standards & Technology, 2024b). A digital signature is used to confirm the user signing the information, and to detect unauthorized modifications. Once the digital signature has been applied, it is non-repudiation since the user who signed the information cannot later claim to not be associated with the information. A digital signature appears as a sting of bits and is either on transmitted or stored data. There are three algorithms used in DSA, the key generation algorithm, the signing algorithm, and the verification algorithm. The standard denotes the algorithms by KeyGen, Sign, and Verify respectively.

The KeyGen algorithm works a little differently than in the previous standard. The algorithm produces a public key and a private key using a completely random seed. The KeyGen algorithm can be seen in Figure 7. The algorithm creates a random 32-byte string and then passes that into a different KeyGen_internal algorithm. This algorithm creates the public key, denoted pk, and the private key denoted sk. PKE is not needed in ML-DSA, so the public private key pair is less complicated. The sign algorithm uses the private key from the KeyGen algorithm, a given message containing the information wishing to be signed, and another dummy random string of 32 bytes. The sign algorithm can be seen in Figure 8. The message is formatted and then along with the

dummy string and private key is passed into the sign_internal algorithm. The sign_internal algorithm creates the signature used on the message. The verify algorithm uses the public key from the KeyGen algorithm, the same given message, and the signature from the sign algorithm. The verify algorithm can be seen in Figure 9. The message is formatted like before and then along with the public key and signature is passed into the verify_internal algorithm. The verify_internal algorithm returns a Boolean to verify the validity of the signature.

Algorithm 1 ML-DSA.KeyGen()

Generates a public-private key pair.

Output: Public key $pk \in \mathbb{B}^{32+32k(\text{bitlen }(q-1)-d)}$
and private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot\text{bitlen }(2\eta)+dk)}$.

1: $\xi \leftarrow \mathbb{B}^{32}$ ▷ choose random seed
2: **if** $\xi =$ NULL **then**
3: **return** $\perp$ ▷ return an error indication if random bit generation failed
4: **end if**
5: **return** ML-DSA.KeyGen_internal (ξ)

Figure 7: ML_DSA KeyGen Algorithm (Standards & Technology, 2024b)

Algorithm 2 ML-DSA.Sign(sk, M, ctx)

Generates an ML-DSA signature.

Input: Private key $sk \in \mathbb{B}^{32+32+64+32\cdot((\ell+k)\cdot\text{bitlen }(2\eta)+dk)}$, message $M \in \{0,1\}^*$,
context string ctx (a byte string of 255 or fewer bytes).
Output: Signature $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen }(\gamma_1-1))+\omega+k}$.

1: **if** $|ctx| > 255$ **then**
2: **return** $\perp$ ▷ return an error indication if the context string is too long
3: **end if**
4:
5: $rnd \leftarrow \mathbb{B}^{32}$ ▷ for the optional deterministic variant, substitute $rnd \leftarrow \{0\}^{32}$
6: **if** $rnd =$ NULL **then**
7: **return** $\perp$ ▷ return an error indication if random bit generation failed
8: **end if**
9:
10: $M' \leftarrow$ BytesToBits(IntegerToBytes$(0,1) \parallel$ IntegerToBytes$(|ctx|,1) \parallel ctx) \parallel M$
11: $\sigma \leftarrow$ ML-DSA.Sign_internal(sk, M', rnd)
12: **return** σ

Figure 8: ML_DSA Sign Algorithm (Standards & Technology, 2024b)

Algorithm 3 ML-DSA.Verify(pk, M, σ, ctx)

Verifies a signature σ for a message M.

Input: Public key $pk \in \mathbb{B}^{32+32k(\text{bitlen }(q-1)-d)}$, message $M \in \{0,1\}^*$,
signature $\sigma \in \mathbb{B}^{\lambda/4+\ell\cdot 32\cdot(1+\text{bitlen }(\gamma_1-1))+\omega+k}$,
context string ctx (a byte string of 255 or fewer bytes).
Output: Boolean.

1: **if** $|ctx| > 255$ **then**
2: **return** $\perp$ ▷ return an error indication if the context string is too long
3: **end if**
4:
5: $M' \leftarrow$ BytesToBits(IntegerToBytes$(0,1) \parallel$ IntegerToBytes$(|ctx|,1) \parallel ctx) \parallel M$
6: **return** ML-DSA.Verify_internal(pk, M', σ)

Figure 9: ML_DSA Verify Algorithm (Standards & Technology, 2024b)

The steps to run all these algorithms between two parties, let's call Alice and Bob, are as follows. Alice in this case is the user who wishes to sign the information and Bob is the user receiving the information and verifying that the information is signed by the right user and has not been altered with. Alice runs the KeyGen algorithm to produce a public and private key. Alice keeps the private key and makes the public key available to Bob. Alice uses the private key and a message and runs the sign algorithm to produce a signature used on the message. Alice sends this message with the signature to Bob. Bob now takes the public key and the signed message and runs the verify algorithm. If the algorithm results in true then Bob has an unaltered message from Alice, and it is verified that Alice is the one who signed it. These three steps are called commitment, challenge, and response.

The public key is made public so anyone who can get access to it can verify the message and signatory. The private key is needed to generate the signature, so only the person who owns it can sign, which makes faking a signature very hard. Digital signatures are required for all sorts of applications such as electronic mail, transfers of funds, data exchange and storage, and distribution. ML_DSA can be implemented in hardware, firmware, or software and is based on the CRYSTALS-Dilithium algorithm mentioned earlier. As of now ML_DSA is secure against quantum computers.

III.B.3 FIPS 205

III.B.4 Applications

The standards have been published so there are already companies designing with these standards in mind. A few examples of companies doing so are Terra Quantum, Nexus Public Key

Infrastructure (PKI), Microsoft, and Amazon Web Services (AWS). Terra Quantum added all three new FIPS standards to their open source TQ42 cryptography library. They did this to "ensure that [their] users are equipped with cryptographic tools that are not only cutting-edge but also recognized and vetted by one of the world's leading authorities technology standards" (Quantum, 2024). Anyone can now view these algorithms along with hundreds of others in the library to build their network and protect it from all sorts of threats by quantum and non-quantum computers. The TQ42 library also includes the FALCON algorithm which is what the next FIPS post quantum cryptography standard is based on. When that standard is finalized, TQ42 will already be prepared to add it to their library.

Nexus PKI provides public key infrastructure services and has already updated them in compliance with the new FIPS standards. They also offer the opportunity to test their services against FIPS 203, 204, and 205, to offer a better understanding of not only their services but also the standards as well. Nexus PKI uses these standards to "provide a solid security foundation for strong authentication, email encryption, digital signing, and securing IoT devices and applications" for all their users (Nexus, 2024). Microsoft has added FIPS 204 to their SymCrypt library to also protect their users from quantum cryptography. This will give their users the ability to test the standard out in their own applications (Thipsay, 2024). Microsoft is working on developing new DSA algorithms and is a part of the team that developed FIPS 203, 204, and 205 for NIST. AWS has done the same thing with FIPS 203. AWS incorporated all three parameter sets of ML-KEM. AWS has successfully used this algorithm in Hybrid Post-Quantum TLS in AWS KMS, Hybrid Post-Quantum TLS in AWS Secrets Manager, and AWS Transfer Family (Jake Massimo, 2025). AWS has also used FIPS 203 to update their secure file transfer protocol (Services, 2025).

III.B.5 Future Directions

The future of post quantum cryptography standards is more standards for stronger and more efficient algorithms. Quantum computers will become more powerful, so security measures need to be able to counter these advances. NIST is already ahead in this area having released FIPS 203, 204, and 205. NIST also has a couple more standards in the draft stages and are planned to be released soon. One being FIPS 206: Fast-Fourier Transform (FFT) over NTRU-Lattice Digital Signature Algorithm (FN-DSA). FIPS 206 will provide a third option for digital signatures that are secure

against quantum computers. FN-DSA will be based on the FALCON algorithm which is a lattice-based signature scheme that uses FFTs and NTRU lattices. FIPS 206 will have smaller signatures and public keys but as of now is “difficult to implement” (Perlner, 2024). The private and public key generation will be performed by a lattice basis row rotation, and the signing will be done by a FFT and LDL tree (Perlner, 2024). There are many similarities when compared to FIPS 204 and 205, but the main differences are that FIPS 206 will only allow randomized signing, forbid export of seeds, and discourage the export of FFT basis and LDL tree (Perlner, 2024). FIPS 206 might be the recommended DSA for smaller signatures.

The other planned FIPS standard to be released is FIPS 207: hamming quasi-cyclic key encapsulation mechanism (HQC-KEM). Right now, the plan for FIPS 207 is to serve as a backup KEM to ML-KEM (Townsend, 2025). Some of the predicted benefits of HQC-KEM is that it has long term keys so that allows for small ciphertexts and fast encapsulation and decapsulation (Robinson, 2025). HQC is also already being used by Mullvad VPN, Rosenpass VPN, and Crypto4A-HSMs so it has been tested for security purposes (Robinson, 2025). The concern surrounding FIPS 207 is the large public key size and whether it will be widely used. FIPS 206 is much closer to being drafted and passed, FIPS 207 was just introduced in the 6th PQC Standardization Conference in September of 2025.

IV In Networking

IV.A Applications

IV.A.1 Quantum Cryptography

Aside from just ensuring that we have classical cryptographic algorithms that can resist quantum computers, we can also potentially improve networking by taking advantage of quantum computing and quantum information in the implementation of networks (Miao et al., 2025).

Will this be feasible anytime soon? Probably not, given the infrastructure cost, but it might be possible to implement at smaller scales in the near future.

Bernstein also mentions quantum cryptography, which is much like a stream cipher, but using a direct fiber link over which q-bits can be transmitted. He does not, however, think it is a reasonable alternative to post quantum classical cryptography, and notes that at least one proponent of

quantum cryptography is a company selling quantum networking hardware called Magiq (and indeed, I found another: Aliro).

IV.A.2 Quantum Physical Layer

IV.A.3 Quantum Routers and Routing Algorithms (?)

IV.B The Post-Quantum Threat Model

Before considering the best way to prepare for quantum computing in computer networks, one must understand the **threat model** in a post-quantum world; That is, what attackers might exist, what capabilities they might possess, what assets they may target, and what kinds of network traffic are at risk. All these questions must be fully fleshed out before any semblance of standards or "best-practices" can be established. As quantum computing is still in its infancy, it is difficult to provide answers to all these questions with a large degree of clarity. However, given past and current technological trends, a few conclusions can be drawn.

Firstly, accessibility to quantum computers, at least initially, is likely to be limited. The latest in quantum computing is that there are a number of issues with affordably scaling quantum computers (Kolodziejczyk, 2023). Recent estimates, for example, calculate that an "estimated 20 million noisy qubits with error rates below 10^{-3}, operating for several hours" (Erol, 2025) would be required to break RSA-2048, which would suggest that access might be limited to well-funded nation-states or organizations. As previously discussed, the existence of quantum computing is an existential threat to classical public-key cryptography. This means that, without sufficient preparation, these "early-adopters" of quantum computing might have a first-mover advantage (Kolodziejczyk, 2023), wherein existing public-key infrastructure might be rendered obsolete overnight. An attacker in this scenario would be able to forge certificates, impersonate websites, derive secret keys, decrypt network traffic, and wreak all manner of havoc on previously secure networks. A full impact assessment of all effected algorithms is shown in Figure 10.

Impact Assessment:

- **RSA**: All key sizes (1024, 2048, 4096-bit) become vulnerable once CRQCs arrive
- **ECC**: Shorter keys (256-bit ECC ≈ 128-bit classical security) break even faster
- **Diffie-Hellman**: Key exchange protocols compromised, affecting TLS handshakes globally
- **Digital Signatures**: ECDSA, EdDSA become forgeable, undermining authentication

2.2. Grover's Algorithm: Weakening Symmetric Cryptography

Grover's algorithm [5] provides quadratic speedup for unstructured search, reducing an N-element search from O(N) to O(√N). For symmetric encryption:

- **AES-128**: Effective security reduced from 128-bit → ~64-bit
- **AES-256**: Remains secure at ~128-bit effective security
- **SHA-256**: Collision resistance weakened but remains practical

Figure 10: An impact assessment of post-quantum computing on classical cryptographic algorithms (Erol, 2025).

If this is the case, the current outlook is not optimistic: Without sufficient preparation for post-quantum computing, critical network infrastructure could be compromised, sensitive communications exposed, and trust in existing PKI and secure protocols undermined. However, even if a widespread switch to post-quantum cryptography is made with haste, the threat against existing data remains: So-called "harvest now, decrypt later" attacks could occur which slowly amass a collection of encrypted data today, with the intention of decrypting it once quantum-capable machines become available (Moody et al., 2024). This means that any sensitive information transmitted or stored under pre-quantum cryptographic standards could remain compromised for years or even decades. Intellectual property, government documents, and healthcare information could all be targets of such an attack. As such, any data requiring long-term secrecy should not be transmitted using cryptographic schemes that remain vulnerable to future quantum attacks.

In summary, attacks such as HNDL (Harvest Now Decrypt Later) demonstrate that quantum computing is not merely a distant threat in the future, but an active, present-day risk that requires immediate mitigation. While only a small number of well-funded adversaries are likely to possess quantum capabilities in the near future, the systems they would target, including government communications and long-lived sensitive data, represent critical components of modern infrastructure. Furthermore, protocols that rely on public key cryptography such as TLS are

particularly vulnerable given their dependance on primitives which are broken by quantum computing.

IV.C Implications and Considerations

Despite the urgency of their implementation, post-quantum cryptographic algorithms pose a number of novel challenges to the network designer. To provide a point of comparison, the average ECDH (Elliptic-Curve Diffie-Hellman) key size is on the scale of tens of bytes; Post-quantum algorithms, on the other hand, range from hundreds to even thousands of bytes (Marin, 2023). In the context of networking, these larger key sizes mean more packet fragmentation. The immediate impact of this is additional round trips, but an increase in the number of packets places more load on routers and switches and raises the likelihood of packet loss or retransmission, further compounding delays. While this may not be noticeable degradation in performance on the fastest of connections, the added latency on slower connections can rapidly accumulate. This effect is particularly evident in TLS, where larger key sizes may introduce multiple extra round trips per handshake, potentially crippling performance on slower connections.

Beyond the increase in network traffic, post-quantum algorithms also demand far more computational resources than traditional cryptography. Similar to network overhead, a “feast or famine” scenario can arise: performance degradation may be negligible on high-end hardware but becomes a significant challenge in resource-constrained environments. For example, one study comparing post-quantum algorithms found that while high-performance servers incurred less than 5% additional latency, certain algorithms on low-power devices required up to 12× more computation to achieve equivalent NIST security levels (Abbasi et al., 2025). So, while the backbone of the internet may handle post-quantum cryptography with minimal impact, consumer devices and IoT hardware could face substantial performance bottlenecks.

Another challenge with post-quantum cryptography in networking is its integration with protocols that don’t support fragmentation: UDP, for example, has no such mechanism for packet reassembly. If the key sized produced by an algorithm exceeds the MTU, then it is completely unusable. As such, any application of key-exchange over UDP cannot simply “drop in” post-quantum algorithms as a replacement without utilizing a higher level protocol that runs on top of

UDP such as QUIC. Even with such a protocol, however, many firewalls or NAT policies block fragmented UDP packets as an anti-DOS measure (Eggert et al., 2017).

Much of the discussion around post-quantum cryptography has centered around TLS and the internet, but this is not the only domain which relies on strong cryptographic algorithms: IPsec, WPA/WPA2, and VPNs are all examples of technologies which utilize cryptography to ensure confidentiality, integrity, and authenticity. Should classical cryptographic algorithms be proven insecure, these protocols would require a full redesign, contending with both previously identified issues and entirely new challenges.

Finally, as with any advancements that require changes in the internet's backbone, post-quantum cryptography would undoubtedly face deployment issues on ossified infrastructure. Despite the native hardware support for classical cryptography, post-quantum algorithms face slow adoption, even on cutting-edge hardware. In addition, the lagging pace of infrastructure upgrades means that even as post-quantum algorithms mature, deployment will be uneven. Larger keys and certificates increase memory usage, which may be trivial on modern servers but can exceed the limitations of older hardware: An experiment by Cloudflare, for example, found that "there are clients or middleboxes that break when certificate chains grow by more than 10kB or 30kB" (Westerbaan, 2024).

IV.D Transition Strategies

While post-quantum computing completely breaks classical public-key cryptography, most symmetric algorithms are affected far less, typically experiencing only a reduction in effective security. As such, the current migration strategy for many symmetric cryptography algorithms, such as AES, is to simply increase the key size (Erol, 2025).

Despite this, strengthening symmetric cryptography is only one piece of the puzzle: Even if the encryption itself is resistant to quantum computers, this resistance means nothing if the key itself can be derived, or the recipient can be impersonated. This is why effective post-quantum public key schemes play a crucial role in preventing HNDL (Harvest Now Decrypt Later) and other such attacks. However, for many of the reasons outlined, transitioning to quantum-resistant algorithms is a challenging process with dedicated mitigation strategies.

As discussed, one of the primary concerns when switching to post-quantum algorithms is key size. If the key-establishment handshake becomes too large, packet fragmentation can compound with network latency to result in a massive decrease in performance. However, several networking protocols are flexible, and can be tuned to optimize for larger handshakes. In TCP, for instance, one such parameter is the initial congestion window size, or `initcwnd`, which controls how many packets the sender can transmit before receiving an ACK. In (Sikeridis et al., 2020), it was found that the total handshake duration using post-quantum algorithms could be reduced by up to 50% by increasing `initcwnd`. Their results are shown in Figure 11:

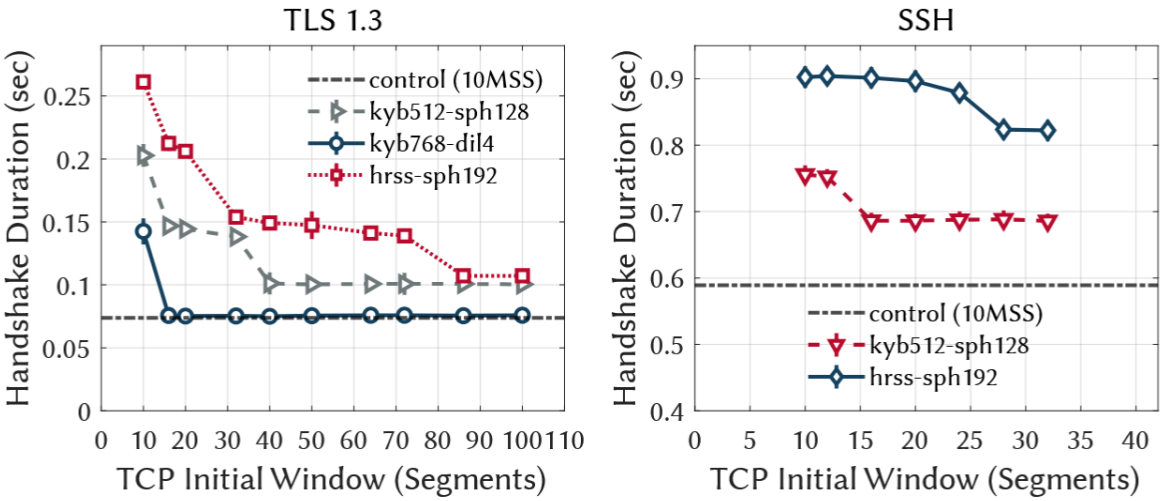


Figure 11: The impact of `initcwnd` on TLS 1.3 and SSH total handshake duration (Sikeridis et al., 2020).

Other protocols contain similar levers, such as DTLS's max handshake size. In addition, more aggressive session reuse or caching mechanisms could greatly expedite the total handshake duration.

Beyond protocol-level tuning, algorithm selection also plays a role in performance. Similar to classical cryptography, not all quantum-resistant algorithms have the same characteristics, with each varying in strength, computational cost, and key size. An overview of the most common algorithm categories and their advantages and disadvantages is shown in Figure 12.

Category	Functional Basis	Advantages	Disadvantages
Lattice-Based	lattice-related problems	high security level, fast and convenient execution, algorithmic simplicity	vulnerable implementation on conventional computers
Code-Based	error-correcting codes	guaranteed and high security level	huge public key, additional data needed for verifying digital signatures
Hash-Based	hash functions	algorithmic simplicity, fast and convenient execution	huge digital signature, security dependency on utilized hash functions
Multivariate	multivariate polynomial equations	simple and efficient implementation, fast and convenient execution, small digital signature size	huge public key, lack of proven security level
Isogeny-Based	isomorphic elliptic curves	small public key size, small digital signature size, low communication cost	vulnerability to some quantum algorithms, mathematical complexity
MPC-in-the-Head	combination of ZKP and MPC protocols	small public key size	huge digital signature

Figure 12: The advantages and disadvantages of various categories of post-quantum cryptographic algorithms (Hosseini & Pilaram, 2024).

As such, there is no "one-size-fits-all" category of algorithm; Choosing the right type for a specific deployment scenario can help mitigate performance issues. In TLS, for instance, lattice-based algorithms are typically chosen, as they typically produce smaller outputs which are less prone to fragmentation and more suitable for networking environments.

On the hardware front, although not as mature as classical cryptography, advancements have been made in specialized accelerators to offset the computational overhead of post-quantum algorithms. Lattice-based algorithms, in particular, rely heavily on polynomial multiplication which accounts for nearly a third of the executed instructions (Zeng et al., 2024). Using dedicated hardware accelerators for NTT, an algorithm used for polynomial multiplication, as much as a 56% performance improvement has been observed for ML-KEM, the NIST standard post-quantum key-exchange algorithm (Dam et al., 2025).

Finally, the battle against the ossification of the internet's backbone is a long but steady battle. One such trick for dealing with ossification is building in backwards compatibility at the protocol level: A similar practice was done with TLS, wherein a 1.3 handshake will masquerade as 1.2, including many of the old headers for the purpose of preventing crashes on legacy clients

(Westerbaan & Rubin, 2022). A similar approach is taken today with post-quantum cryptography, where hybrid or fallback schemes are typically provided. In addition, techniques like protocol "greasing", where random unused fields are deliberately sent in transmissions, can help detect and deter problematic implementations before they become widespread (Westerbaan & Rubin, 2022).

IV.E Case Studies

IV.E.1 CECPQ1

In a 2016 experiment to investigate the feasibility of post-quantum cryptography on a large scale, Google announced that a small fraction of Google Chrome web browser's would temporarily switch to a post-quantum key exchange algorithm in addition to the typically used elliptic curve default (Google, 2016). The algorithm selected for the experiment was "New Hope", a lattice-based key-exchange which uses a problem known as "Ring Learning With Errors", or "Ring-LWE" (Alkim et al., 2015). To facilitate its use within TLS, Google developed CECPQ1, a key-agreement algorithm with X25519, the existing elliptic curve, as a backstop in case of failure. The stated goals for this experiment were to

> "a) provide the research community a target because it would be very useful to know whether the ring structure in R-LWE is a mistake sooner rather than later
>
> b) to test the real-world impact on latency and compatibility with the larger handshake messages that any post-quantum scheme seems likely to need."
>
> — (Langley, 2016a)

After several months, Google announced plans to end the experiment and discontinue CECPQ1 support in Chrome. The results were somewhat promising: Despite anticipated compatibility issues with middle boxes, CECPQ1′s deployment was reportedly seamless. In addition, the median connection latency only increased by 1 ms. The devices most affected were those with already slow connections, which took an additional hit due to CECPQ1′s increased message sizes: The slowest 5% saw an increase in latency of 20 ms, and the bottom 1% experienced an average of 150 ms extra latency (Langley, 2016b). Despite this increase in latency, Google's experiment with CECPQ1 demonstrated the feasibility of widespread deployment of post-quantum cryptography.

IV.E.2 CECPQ2

Building on the insights gained from the CECPQ1 experiment, Google collaborated up with Cloudflare to test CECPQ2, an altered version of its Q1 counterpart. The primary goal of CECPQ2 was to continue testing a combined elliptic curve/post-quantum key-exchange algorithm at scale while tuning CECPQ1 to perform better in TLS 1.3 handshakes.

TLS 1.3 differs from 1.2 in that the client sends a set of unsolicited public s to the TLS server, whereas only one would be sent in 1.2. This is particularly relevant for post-quantum key-exchange algorithms, which already suffer from larger key sizes. To test their impact on TLS 1.3, Google injected a dummy TLS extension into the handshake based on three different algorithm families key size estimations: Supersingular isogenies at 400 bytes, structured lattices at 1,100, and unstructured lattices at 10,000. After probing 2,500 websites to observe how they were affected, it was revealed that 21 of these websites to failed under the 10,000 byte load (Langley, 2018). Despite this, the algorithms with smaller key sizes (Supersingular isogenies) consistently were measured to have longer key derivation times. The question posed by the results of these experiments was clear: Does the faster key derivation of lattice-based algorithm outstrip the increased network overhead resulting from its larger key sizes?

To further investigate this tradeoff, Google tested two variations of CECPQ2: CECPQ2, and CECPQ2b. Whereas CECPQ2 employed HRSS, an algorithm of the structured lattice family, CECPQ2b utilized SIKE, an isogeny-based algorithm. After deploying both to Cloudflare's TLS stack, metrics for both were collected. After analysis, it was found that the HRSS-based CECPQ2 generally outperformed Q2b across the majority of connection types, although Q2b had a significant advantage for slower connections due to its smaller message size, although the report noted that improvements in hardware or algorithmic optimizations may change this in the future (Kwiatkowski & Valenta, 2019).

V Conclusion

With the creation of better and better quantum computers, we don't just get a new technology to play around with, we get a device capable of challenging the very structure of network security we rely on, by cracking down on problems that modern computers just cant solve.

To address risks quantum computers will have on security, many real-world approaches have been devised to “Shor” up holes in security algorithms like ElGamal, but they have been slowed due to deployment and infrastructure costs. The transition of global infrastructure toward quantum-resistant security wont be as easy as a one size fits all algorithm or approach though, many systems will have to change, and we will have to monitor tradeoffs of this security with tradeoffs of performance or complexity.

Ultimately, the transition to post-quantum cryptography is one that many modern companies and industries like NIST and Google are showing promising steps in, in order to shape and better secure the data of tomorrow.

Bibliography

Abbasi, M., Cardoso, F., Váz, P., Silva, J., & Martins, P. (2025). A Practical Performance Benchmark of Post-Quantum Cryptography Across Heterogeneous Computing Environments. *Cryptography*, *9*(2), 32. https://doi.org/10.3390/cryptography9020032

Alkim, E., Ducas, L., Pöppelmann, T., & Schwabe, P. (2015,). *Post-quantum key exchange - a new hope*. https://eprint.iacr.org/2015/1092

Amico, M., Saleem, Z. H., & Kumph, M. (2019). Experimental study of Shor's factoring algorithm using the IBM Q Experience. *Physical Review a*, *100*(1). https://doi.org/10.1103/physreva.100.012305

Bernstein, D. J. (2009). Introduction to post-quantum cryptography. In D. J. Bernstein, J. Buchmann, & E. Dahmen (Eds.), *Post-Quantum Cryptography* (pp. 1–14). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-88702-7_1

Chae, E., Choi, J., & Kim, J. (2024). An elementary review on basic principles and developments of qubits for quantum computing. *Nano Convergence*, *11*, 11. https://doi.org/10.1186/s40580-024-00418-5

Dam, D.-T., Nguyen, K.-D., Le, D.-H., & Pham, C.-K. (2025). Accelerating Post-Quantum Cryptography: A High-Efficiency NTT for ML-KEM on RISC-V. *Preprints*. https://doi.org/10.20944/preprints202511.1018.v1

Eggert, L., Fairhurst, G., & Shepherd, G. (2017). *UDP Usage Guidelines* (Best Current Practice No. RFC8085). https://www.rfc-editor.org/rfc/rfc8085

Erol, V. (2025). Quantum Readiness in Cryptography: A Maturity-Based Framework for Post-Quantum Transition. *Preprints*. https://doi.org/10.20944/preprints202509.2584.v1

Google. (2016,). *Experimenting with Post-Quantum Cryptography*. https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

Gordon, D. (1993). Discrete Logarithms in GF(P) Using the Number Field Sieve. *Siam Journal on Discrete Mathematics - SIAMDM*, *6*, 124–138. https://doi.org/10.1137/0406010

Grover, L. K. (1996,). *A fast quantum mechanical algorithm for database search*. https://arxiv.org/abs/quant-ph/9605043

Hardy, G. H., & Wright, E. M. (2008). *An Introduction To The Theory Of Numbers*. Oxford University Press. https://doi.org/10.1093/oso/9780199219858.001.0001

Hosseini, S. M., & Pilaram, H. (2024,). *A Comprehensive Review of Post-Quantum Cryptography: Challenges and Advances*. https://eprint.iacr.org/2024/1940

Jake Massimo, P. K. D. K. M. A., Matthew Campagna. (2025,). *Submission for NIST Workshop on Guidance for Key Encapsulation Mechanisms (KEM)*. https://csrc.nist.gov/csrc/media/Events/2025/workshop-on-guidance-for-kems/documents/papers/building-post-quantum-cloud-services-paper.pdf

Jang, K., Baksi, A., Kim, H., Song, G., Seo, H., & Chattopadhyay, A. (2022,). *Quantum Analysis of AES*. https://doi.org/10.62056/ay11zo-3y

Kolodziejczyk, B. (2023). *Scaling Quantum Computing Technologies – Opportunities, Challenges and Policy* [Technical report]. https://sdgs.un.org/sites/default/files/2023-05/A2-%20Bart%20Kolodziejczyk%20-%20Scaling%20quantum%20computing%20technologies.pdf

Kwiatkowski, K., & Valenta, L. (2019, October 30). *The TLS Post-Quantum Experiment*.

Langley, A. (2016a, July 7). *Intent to Implement and Ship: CECPQ1 for TLS*. https://groups.google.com/a/chromium.org/g/security-dev/c/DS9pp2U0SAc

Langley, A. (2016b, November 28). *CECPQ1 results*. https://www.imperialviolet.org/2016/11/28/cecpq1.html

Langley, A. (2018, April 11). *Post-quantum confidentiality for TLS*. https://www.imperialviolet.org/2018/04/11/pqconftls.html

Lyubashevsky, V. (2009). Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In M. Matsui (Ed.), *Advances in Cryptology – ASIACRYPT 2009: Advances in Cryptology – ASIACRYPT 2009*.

Marin. (2023, May 17). *Post-Quantum Cryptography (PQC) and Network Connectivity: Challenges and Impacts*. https://postquantum.com/post-quantum/pqc-network-impacts/

Menezes, A. J., Oorschot, P. C. van, & Vanstone, S. A. (1997). *Handbook of Applied Cryptography* (1st ed.). CRC Press. https://doi.org/10.1201/9780429466335

Miao, C., Léger, S., Li, Z., Lee, G., Jiang, L., & Schuster, D. I. (2025). Implementation of a Quantum Addressable Router Using Superconducting Qubits. *PRX Quantum*, *6*(4), 40335. https://doi.org/10.1103/pq3x-cmw9

Moody, D., Perlner, R., Regenscheid, A., Robinson, A., & Cooper, D. (2024). *Transition to Post-Quantum Cryptography Standards* [NIST Internal Report (IR) 8547 ipd]. https://doi.org/10.6028/NIST.IR.8547.ipd

Nexus. (2024,). *Rely on Nexus for your PKI's Post-Quantum Readiness*. https://nexus.ingroupe.com/post-quantum-cryptography-nexus-pki/

Nielsen, M. A., & Chuang, I. L. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.

Perlner, R. (2024,). *FIPS 206 Status Update*. https://csrc.nist.gov/csrc/media/presentations/2025/fips-206-fn-dsa-(falcon)/images-media/fips_206-perlner_2.1.pdf

Quantum, T. (2024, July 30). *Terra Quantum Adds NIST-Compliant FIPS Standards to OSS PQC Library*. https://terraquantum.swiss/news/terra-quantum-adds-nist-compliant-fips-standards-to-oss-pqc-library/

Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, *56*(6). https://doi.org/10.1145/1568318.1568324

Robinson, A. (2025, September). *FIPS 207: HQC-KEM*. https://csrc.nist.gov/presentations/2025/fips-207-hqc-kem

Services, A. W. (2025, May 21). *AWS Transfer Family Announces ML-KEM Quantum-Resistant Key Exchange for SFTP*. https://aws.amazon.com/about-aws/whats-new/2025/05/aws-transfer-family-quantum-resistant-key-exchange/

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. https://doi.org/10.1137/s0097539795293172

Sikeridis, D., Kampanakis, P., & Devetsikiotis, M. (2020). *Assessing the Overhead of Post-Quantum Cryptography in TLS 1.3 and SSH*. https://doi.org/10.1145/3386367.3431305

Standards, N. I. of, & Technology. (2024b). *Module-Lattice-Based Digital Signature Standard* [NIST Federal Information Processing Standards Publication]. https://doi.org/10.6028/NIST.FIPS.204

Standards, N. I. of, & Technology. (2024a). *Module-Lattice-Based Key-Encapsulation Mechanism Standard* [NIST Federal Information Processing Standards Publication]. https://doi.org/10.6028/NIST.FIPS.203

Thipsay, A. (2024, September 9). *Microsoft's Quantum-Resistant Cryptography is Here*. https://techcommunity.microsoft.com/blog/microsoft-security-blog/microsofts-quantum-resistant-cryptography-is-here/4238780

Townsend, K. (2025, March 17). *NIST Announces HQC as Fifth Standardized Post Quantum Algorithm*. https://www.securityweek.com/nist-announces-hqc-as-fifth-standardized-post-quantum-algorithm/

Westerbaan, B. (2024, March 5). *The state of the post-quantum Internet*. https://blog.cloudflare.com/pq%E2%80%912024/

Westerbaan, B., & Rubin, C. D. (2022, October 3). *Defending against future threats: Cloudflare goes post-quantum*. https://blog.cloudflare.com/post-quantum-for-all/

Zeng, C., He, D., Feng, Q., Peng, C., & Luo, M. (2024). The implementation of polynomial multiplication for lattice-based cryptography: A survey. *Journal of Information Security and Applications*, *83*, 103782. https://doi.org/https://doi.org/10.1016/j.jisa.2024.103782