# Trail of Bits Blog

# A Guide to Post-Quantum Cryptography

- **POST**
- OCTOBER 22, 2018
- 2 COMMENTS

For many high-assurance applications such as TLS traffic, medical databases, and blockchains, forward secrecy is absolutely essential. It is not sufficient to prevent an attacker from immediately decrypting sensitive information. Here the threat model encompasses situations where the adversary may dedicate many years to the decryption of ciphertexts after their collection. One potential way forward secrecy might be broken is that a combination of increased computing power and number-theoretic breakthroughs make attacking current cryptography tractable. However, unless someone finds a polynomial time algorithm for factoring large integers, this risk is minimal for current best practices. We should be more concerned about the successful development of a quantum computer, since such a breakthrough would render most of the cryptography we use today insecure.

## Quantum Computing Primer

Quantum computers are not just massively parallel classical computers (https://www.scottaaronson.com/papers/npcomplete.pdf). It is often thought that since a quantum bit can occupy both 0 and 1 at the same time, then an n-bit quantum computer can be in $2^n$ states simultaneously and therefore compute NP-complete problems extremely fast. This is not the case, since measuring a quantum state destroys much of the original information. For example, a quantum system has complete knowledge of both an object's momentum and location, but any measurement of momentum will destroy information about location and vice versa. This is known as the Heisenberg uncertainty principle (https://en.wikipedia.org/wiki/Uncertainty_principle). Therefore, successful quantum algorithms consist of a series of transformations of quantum bits such that, at the end of the

computation, measuring the state of the system will not destroy the needed information. As a matter of fact, it has been shown (https://arxiv.org/pdf/quant-ph/9701001.pdf) that there cannot exist a quantum algorithm that simultaneously attempts all solutions to some NP-complete problem and outputs a correct input. In other words, any quantum algorithm for solving hard classical problems must exploit the specific structure of the problem at hand. Today, there are two such algorithms that can be used in cryptanalysis.

The ability to quickly factor large numbers would break both RSA and discrete log-based cryptography. The fastest algorithm for integer factorization is the general number field sieve, which runs in sub-exponential time. However, in 1994 Peter Shor developed a quantum algorithm (Shor's algorithm (https://www.scottaaronson.com/blog/?p=208)) for integer factorization that runs in polynomial time, and therefore would be able to break any RSA or discrete log-based cryptosystem (including those using elliptic curves). This implies that all widely used public key cryptography would be insecure if someone were to build a quantum computer.

The second is Grover's algorithm, which is able to invert functions in $O(\sqrt{n})$ time. This algorithm would reduce the security of symmetric key cryptography by a root factor, so AES-256 would only offer 128-bits of security. Similarly, finding a pre-image of a 256-bit hash function would only take $2^{128}$ time. Since increasing the security of a hash function or AES by a factor of two is not very burdensome, Grover's algorithm does not pose a serious threat to symmetric cryptography. Furthermore, none of the pseudorandom number generators suggested for cryptographic use would be affected by the invention of a quantum computer, other than perhaps the $O(\sqrt{n})$ factor incurred by Grover's algorithm.

# Types of Post-Quantum Algorithms

Post-quantum cryptography is the study of cryptosystems which can be run on a classical computer, but are secure even if an adversary possesses a quantum computer. Recently, NIST initiated (https://csrc.nist.gov/Projects/Post-Quantum-Cryptography) a process for standardizing post-quantum cryptography and is currently reviewing first-round submissions. The most promising of these submissions included cryptosystems based on lattices, isogenies, hash functions, and codes.

Before diving more deeply into each class of submissions, we briefly summarize the tradeoffs inherent in each type of cryptosystem with comparisons to current (not post-quantum) elliptic-curve cryptography. Note that codes and isogenies are capable of producing digital signatures, but no such schemes were submitted to NIST.

| | **Signatures** | **Key Exchange** | **Fast?** |
|---|---|---|---|
| Elliptic Curves | 64 bytes | 32 bytes | ✓ |
| | | | |

| | | | |
|---|---|---|---|
| Lattices | 2.7kb | 1 kb | ✓ |
| Isogenies | ✗ | 330 bytes | ✗ |
| Codes | ✗ | 1 mb | ✓ |
| Hash functions | 41 kb | ✗ | ✓ |

Table 1: Comparison of classical ECC vs post-quantum schemes submitted to NIST

In terms of security proofs, none of the above cryptosystems reduce to NP-hard (or NP-complete) problems. In the case of lattices and codes, these cryptosystems are based on slight modifications of NP-hard problems. Hash-based constructions rely on the existence of good hash functions and make no other cryptographic assumptions. Finally, isogeny-based cryptography is based on a problem that is conjectured to be hard, but is not similar to an NP-hard problem or prior cryptographic assumption. It's worth mentioning, however, that just as we cannot prove any classical algorithm is not breakable in polynomial time (since P could equal NP), it could be the case that problems thought to be difficult for quantum computers might not be. Furthermore, a cryptosystem not reducing to some NP-hard or complete problem shouldn't be a mark against it, per se, since integer factorization and the discrete log problem are not believed to be NP-complete.

## Lattices

Of all the approaches to post-quantum cryptography, lattices are the most actively studied and the most flexible. They have strong security reductions and are capable of key exchanges, digital signatures, and far more sophisticated constructions like fully homomorphic encryption (https://blog.cryptographyengineering.com/2012/01/02/very-casual-introduction-to-fully/). Despite the extremely complex math needed in both optimizations and security proofs for lattice cryptosystems, the foundational ideas only require basic linear algebra. Suppose you have a system of linear equations of the form

(https://trailofbits.files.wordpress.com/2018/10/image6.png)

Solving for **x** is a classic linear algebra problem that can be solved quickly using Gaussian elimination. Another way to think about this is that we have a mystery function,

(https://trailofbits.files.wordpress.com/2018/10/image2.png)

where given a vector **a**, we see the result of **ax**, without knowing **x**. After querying this function enough times we can learn `f` in a short amount of time (by solving the system of equations above). This way we can reframe a linear algebra problem as a machine learning problem.

Now, suppose we introduce a small amount of noise to our function, so that after multiplying **x** and **a**, we add an error term **e** and reduce the whole thing modulo a (medium-sized) prime q. Then our noisy mystery function looks like

(https://trailofbits.files.wordpress.com/2018/10/image10.png)

Learning this noisy mystery function has been mathematically proven to be extremely difficult. The intuition is that at each step in the Gaussian elimination procedure we used in the non-noisy case, the error term gets bigger and bigger until it eclipses all useful information about the function. In the cryptographic literature this is known as the Learning With Errors (https://cims.nyu.edu/~regev/papers/lwesurvey.pdf) problem (LWE).

The reason cryptography based on LWE gets called lattice-based cryptography is because the proof that LWE is hard relies on the fact that finding the shortest vector in something called a lattice is known to be NP-Hard. We won't go into the mathematics of lattices in much depth here, but one can think of lattices as a tiling of n-dimensional space

(https://trailofbits.files.wordpress.com/2018/10/image5.jpg)

Lattices are represented by coordinate vectors. In the example above, any point in the lattice can be reached by combining $\mathbf{e_1}$, $\mathbf{e_2}$, and $\mathbf{e_3}$ (via normal vector addition). The shortest vector problem (SVP) says: given a lattice, find the element whose length as a vector is shortest. The intuitive reason this is difficult is because not all coordinate systems for a given lattice are equally easy to work with. In the above example, we could have instead represented the lattice with three coordinate vectors that were extremely long and close together, which makes finding vectors close to the origin more difficult. As a matter of fact, there is a canonical way to find the "worst possible" representation of a lattice. When using such a representation, the shortest vector problem is known to be NP-hard.

Before getting into how to use LWE to make quantum-resistant cryptography, we should point out that LWE itself is not NP-Hard. Instead of reducing directly to SVP, it reduces to an approximation of SVP that is actually conjectured to *not* be NP-Hard. Nonetheless, there is currently no polynomial (or subexponential) algorithm for solving LWE.

Now let's use the LWE problem to create an actual cryptosystem. The simplest scheme was created by Oded Regev in his original paper (https://cims.nyu.edu/~regev/papers/qcrypto.pdf) proving the hardness of the LWE problem. Here, the secret key is an n-dimensional vector with integer entries mod q, i.e. the LWE secret mentioned above. The public key is the matrix **A** from the previous discussion, along with a vector of outputs from the LWE function

(https://trailofbits.files.wordpress.com/2018/10/image8.png)

(https://trailofbits.files.wordpress.com/2018/10/image7.png)

An important property of this public key is that when it's multiplied by the vector `(-sk,1)`, we get back the error term, which is roughly `0`.

To encrypt a bit of information `m`, we take the sum of random columns of **`A`** and encode `m` in the last coordinate of the result by adding 0 if `m` is `0` and `q/2` if `m` is `1`. In other words, we pick a random vector **`x`** of 0s or 1s, and compute

Intuitively, we've just evaluated the LWE function (which we know is hard to break) and encoded our bit in the output of this function.

Decryption works because knowing the LWE secret will allow the recipient to get back the message, plus a small error term

When the error distribution is chosen correctly, it will never distort the message by more than `q/4`. The recipient can test whether the output is closer to `0` or `q/2 mod q` and decode the bit accordingly.

A major problem with this system is that it has very large keys. To encrypt just one bit of information requires public keys with size $n^2$ in the security parameter. However, an appealing aspect of lattice cryptosystems is that they are extremely fast.

Since Regev's original paper there has been a massive body of work around lattice-based cryptosystems. A key breakthrough for improving their practicality was the development of Ring-LWE, which is a variant of the LWE problem where keys are represented by certain polynomials. This has led to a quadratic decrease in key sizes and sped up encryption and decryption to use only `n*log(n)` operations (using Fast Fourier techniques).

Among the many lattice-based cryptosystems being considered for the NIST PQC standard, two that are especially worth mentioning are the Crystals constructions, Kyber (https://pq-crystals.org/kyber/) and Dilithium (https://pq-crystals.org/dilithium/index.shtml).

Kyber is a key-encapsulation mechanism (KEM) which follows a similar structure to the system outlined above, but uses some fancy algebraic number theory to get even better performance than Ring-LWE. Key sizes are approximately 1kb for reasonable security parameters (still big!) but encryption and decryption time is on the order of .075 ms. Considering this speed was achieved in software, the Kyber KEM seems promising for post-quantum key exchange.

Dilithium is a digital signature scheme based on similar techniques to Kyber. Its details are beyond the scope of this blog post but it's worth mentioning that it too achieves quite good performance. Public key sizes are around 1kb and signatures are 2kb. It is also quite performant. On Skylake processors the average number of cycles required to compute a signature was around 2 million. Verification took 390,000 cycles on average.

# Codes

The study of error correcting codes has a long history in the computer science literature dating back to the ground-breaking work of Richard Hamming and Claude Shannon. While we cannot even begin to scratch the surface of this deep field (https://www.cs.cmu.edu/~venkatg/teaching/codingtheory/) in a short blog post, we give a quick overview.

When communicating binary messages, errors can occur in the form of bit flips. Error-correcting codes provide the ability to withstand a certain number of bit flips at the expense of message compactness. For example, we could protect against single bit flips by encoding 0 as 000 and 1 as 111. That way the receiver can determine that 101 was actually a 111, or that 001 was a 0 by taking a majority vote of the three bits. This code cannot correct errors where two bits are flipped, though, since 111 turning into 001 would be decoded as 0.

The most prominent type of error-correcting codes are called linear codes, and can be represented by `k x n` matrices, where `k` is the length of the original messages and `n` is the length of the encoded message. In general, it is computationally difficult to decode messages without knowing the underlying linear code. This hardness underpins the security of the McEliece public key cryptosystem (http://www.math.unl.edu/~s-jeverso2/McElieceProject.pdf).

At a high level, the secret key in the McEliece system is a random code (represented as a matrix **G**) from a class of codes called Goppa codes (https://en.wikipedia.org/wiki/Goppa_code). The public key is the matrix **SGP** where **S** is an invertible matrix with binary entries and **P** is a permutation. To encrypt a message **m**, the sender computes **c = m(SGP) + e**, where **e** is a random error vector with precisely the number of errors the code is able to correct. To decrypt, we compute $\mathbf{cP^{-1} = mSG + eP^{-1}}$ so that **mS** is a codeword of **G** that can correct the added error term **e**. The message can be easily recovered by computing $\mathbf{mSS^{-1}}$.

Like lattices, code-based cryptography suffers from the fact that keys are large matrices. Using the recommended security parameters (http://pqcrypto.eu.org/docs/initial-recommendations.pdf), McEliece public keys are around 1 mb and private keys are 11 kb. There is currently ongoing work (https://eprint.iacr.org/2012/409.pdf) trying to use a special class of codes called quasi-cyclic moderate density parity-check codes that can be represented more succinctly than Goppa codes, but the security of these codes is less well studied than Goppa codes.

# Isogenies

The field of elliptic-curve cryptography is somewhat notorious for using quite a bit of arcane math. Isogenies (https://2017.pqcrypto.org/school/slides/Isogeny_based_crypto.pdf) take this to a whole new level. In elliptic-curve cryptography we use a Diffie-Hellman type protocol to acquire a shared secret, but instead of raising group elements to a certain power, we walk through points on an elliptic curve. In isogeny-based cryptography, we again use a Diffie-Hellman type protocol but instead of walking through points on elliptic curve, we walk through a sequence of elliptic curves themselves.

*From An Introduction to Supersingular Isogeny-Based Cryptography (https://ecc2017.cs.ru.nl/slides/ecc2017school-costello.pdf)*

An *isogeny* is a function that transforms one elliptic curve into another in such a way that the group structure of the first curve is reflected in the second. For those familiar with group theory, it is a group homomorphism with some added structure dealing with the geometry of each curve (https://en.wikipedia.org/wiki/Morphism_of_algebraic_varieties). When we restrict our attention to supersingular elliptic curves (https://en.wikipedia.org/wiki/Supersingular_elliptic_curve) (which we won't define here), each curve is guaranteed to have a fixed number of isogenies from it to other supersingular curves.

Now, consider the graph created by examining all the isogenies of this form from our starting curve, then all the isogenies from those curves, and so on. This graph turns out to be highly structured in the sense that if we take a random walk starting at our first curve, the probability of hitting a specific other curve is negligibly small (unless we take exponentially many steps). In math jargon, we say that the graph generated by examining all these isogenies is an expander graph (https://people.seas.harvard.edu/~salil/pseudorandomness/expanders.pdf) (and also Ramanujan (https://mast.queensu.ca/~murty/ramanujan.pdf)). This property of expansion is precisely what makes isogeny-based cryptography secure.

For the Supersingular Isogeny Diffie-Hellman (https://en.wikipedia.org/wiki/Supersingular_isogeny_key_exchange) (SIDH) scheme, secret keys are a chain of isogenies and public keys are curves. When Alice and Bob combine this information, they acquire curves that are different, but have the same j-invariant (https://en.wikipedia.org/wiki/J-invariant). It's not so important for the purposes of cryptography what a j-invariant is, but rather that it is a number that can easily be computed by both Alice and Bob once they've completed the key exchange.

Isogeny-based cryptography has extremely small key sizes compared to other post-quantum schemes, using only 330 bytes for public keys. Unfortunately, of all the techniques discussed in this post, they are the slowest, taking between 11-13 ms for both key generation and shared secret computation (https://blog.cloudflare.com/sidh-go/). They do, however, support perfect forward secrecy, which is not something other post-quantum cryptosystems possess.

# Hash-Based Signatures

There are already many friendly introductions to hash-based signatures (https://blog.cryptographyengineering.com/2018/04/07/hash-based-signatures-an-illustrated-primer/), so we keep our discussion of them fairly high-level. In short, hash signatures use inputs to a hash function as secret keys and outputs as public keys. These keys only work for one signature though, as the signature itself reveals parts of the secret key. This extreme inefficiency of hash-based signatures led to use of Merkle trees (https://en.wikipedia.org/wiki/Merkle_tree) to reduce space consumption (yes, the same Merkle trees used in Bitcoin).

Unfortunately, it is not possible to construct a KEM or a public key encryption scheme out of hashes. Therefore hash-based signatures are not a full post-quantum cryptography solution. Furthermore, they are not space efficient; one of the more promising signature schemes, SPHINCS (https://sphincs.cr.yp.to/sphincs-20150202.pdf), produces signatures which are 41kb and public/private keys that are 1kb. On the other hand, hash-based schemes are extremely fast since they only require the computation of hash functions. They also have extremely strong security proofs, based solely on the assumption that there exist hash functions that are collision-resistant and preimage resistant. Since nothing suggests current widely used hash functions like SHA3 or BLAKE2 are vulnerable to these attacks, hash-based signatures are secure.

# Takeaways

Post-quantum cryptography is an incredibly exciting area of research that has seen an immense amount of growth over the last decade. While the four types of cryptosystems described in this post have received lots of academic attention, none have been approved by NIST and as a result are not recommended for general use yet. Many of the schemes are not performant in their original form, and have been subject to various optimizations that may or may not affect security. Indeed, several attempts to use more space-efficient codes for the McEliece system have been shown to be insecure (https://eprint.iacr.org/2009/509.pdf). As it stands, getting the best security from post-quantum cryptosystems requires a sacrifice of some amount of either space or time. Ring lattice-based cryptography is the most promising avenue of work in terms of flexibility (both signatures and KEM, also fully homomorphic encryption), but the assumptions that it is based on have only been studied intensely for several years. Right now, the safest bet is to use McEliece with Goppa codes since it has withstood several decades of cryptanalysis.

However, each use case is unique. If you think you might need post-quantum cryptography, get in touch (https://www.trailofbits.com/contact/) with your friendly neighborhood cryptographer. Everyone else ought to wait until NIST has finished its standardization process.

By blperez1 Posted in Cryptography (https://blog.trailofbits.com/category/cryptography/)

## 2 thoughts on “A Guide to Post-Quantum Cryptography”

1. Pingback: We crypto now | Trail of Bits Blog
2. Pingback: What do La Croix, octonions, and Second Life have in common? | Trail of Bits Blog