Post-quantum cryptography

Are we getting there any time soon?

T.Gregersen

Todays menu

Introduction

The impact on cryptography today

- Symmetric cryptography
- Asymmetric means

The quantum threat

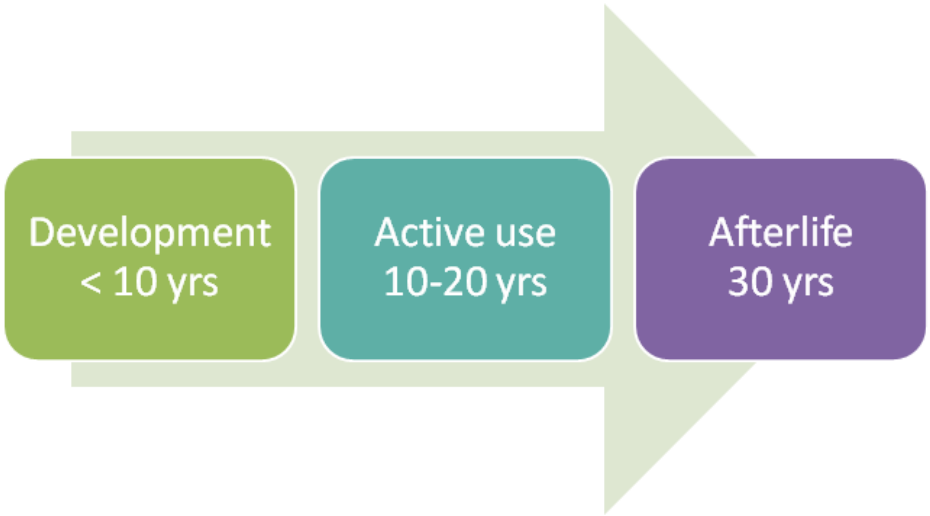
- Grovers algorithm
- Shors algorithm
- The relevant consequences

Post quantum cryptography

- Code based techniques
- Lattice-based systems
- Hash-based signatures

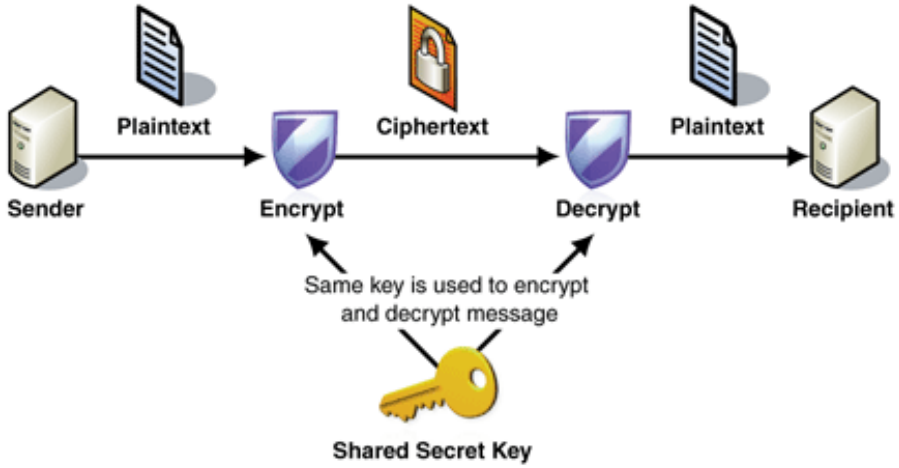
Are we about to land?

- Quantum computers *might* be a problem for our reliance on cryptography in todays protocols.
- The idea of these machines have been around for a long time, not sure if they will come to full fruition. But let us assume that this does happen.



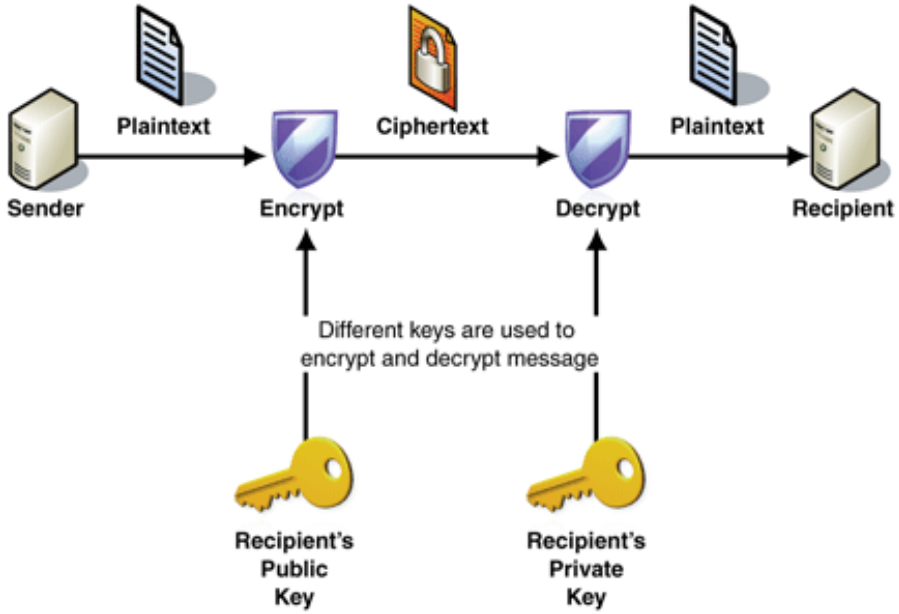
  - The introduction of new primitives will usually take an extensive amount of time until we are using them.
  - This means we should start the debate on which ones to use early!

- The basis of most solutions:



- There is Grovers algorithm, but this problem is possible to handle through extending key sizes.

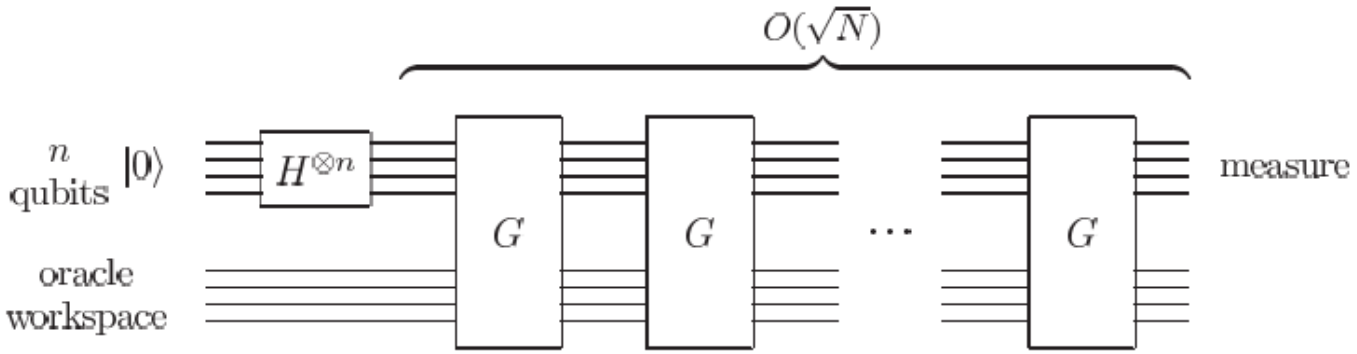
- Key establishment, signatures:



- Shors algorithm might turn out to be disastrous, this depends on the scale and to what extent we may control quantum circuits.

- The targets of today: RSA, DH, ECDH.
- We are assuming the existence of hard computational problems to build on (factorization, finding logarithms).
- The relevant quantum algorithms attack the fundamental problems with some interesting limitations.

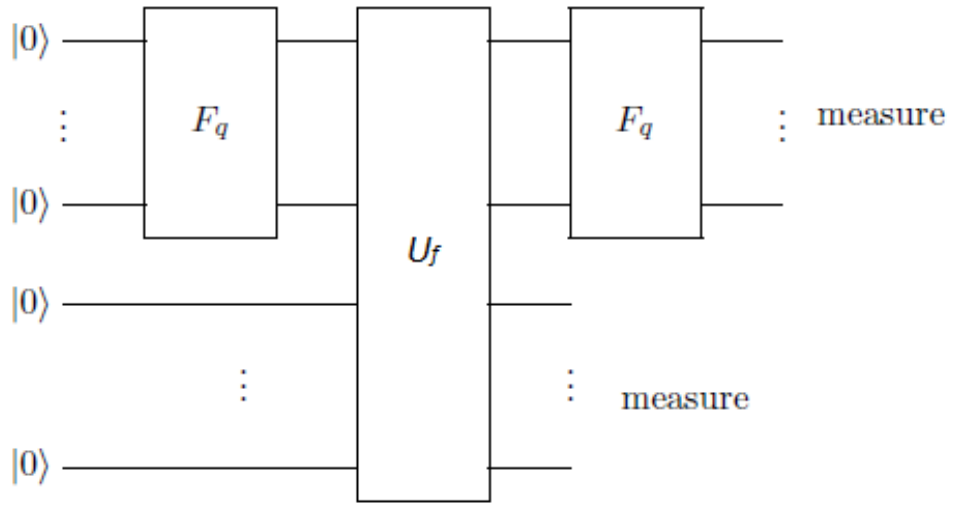
- An unstructured search construction which may be used to speed up any process where this is helpful (finding inverse images, exploring key spaces, collision searching...).
- Here is the relevant diagram:



- What we need for this to happen:
  - Scalability of memory
  - Qubits that can be initialized to arbitrary values
  - Quantum gates that are faster than decoherence time
  - Universal gate set
  - Qubits that can be read easily
- Not at all trivial, we do not yet know if all of these can be handled arbitrarily.

- Its kernel uses a quantum Fourier tranform (QFT) which shows improvement over its classical counterpart.
- It can be used for factoring numbers, finding discrete logarithms by solving a hidden subgroup problem (but not for *all* types of groups). We rely on finding periods of functions so that Fourier analysis comes to the aid.

- How do we use the QFT to find the period? Here is the relevant diagram:



- A pessimistic view forces a doubling the number of bits in symmetric keys, but not yet sure if this is really necessary[1]).
- Asymmetric algorithms are potentially hurt beyond practical use, this is where we need the first solutions!
- There are multiple initiatives to solve these issues, the way ahead is guided by several voices:
  - National Institute of Standards and Technology (NIST) has gone through an extended process through several rounds of narrowing the field of potential candidates for US standards[2].
  - PQCRYPTO (EU) a separate initiative to evaluate candidates independently [3].

[1] https://arxiv.org/abs/1512.04965
[2] https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
[3] https://pqcrypto.eu.org/

- What properties should replacement algorithms have?
  - Keys/signatures/ciphertext should that show some sort of efficiency with regards to space/time constraints.
  - Constructions that show strong security foundations.
  - Implementation should be straightforward with ability to handle side channels.
- There are candidates, but combining all this isn't necessarily easy.

- NIST started with 69 algorithms for key establishment and signatures. Over time, many have fallen to cryptanalysis.
- Examples of KEMs (based on coding theory, lattices or isogeny graphs):
  - BIKE/Classic McEliece/HQC/LedaCrypt/NTS-KEM/ ROLLO/RQC.
  - CRYSTALS-KYBER/FrodoKEM/LAC/NewHope/NTRU/ NTRU Prime/Round5/SABER/Three Bears.
  - SIKE.
- Examples of signature algorithms (based on lattices, multivariate polynomials, ZKPs and cryptographic hash functions):
  - CRYSTALS-DILITHIUM/FALCON/qTesla.
  - GeMSS/LUOV/MQDSS/Rainbow.
  - Picnic.
  - SPHINCS+.

- Before the coming annoncements, we are down to the following candidates:
  - Primary candidates for key establishment: Classic McEliece (BIKE/HQC secondary candidates kept for further research), NTRU/Kyber/SABER (Frodo/NTRUPrime kept for further research). SIKE is also kept for further research.
  - Primary candidates for signatures: SPHINCS+, FALCON/Dilithium, Rainbow (GeMSS is kept for further research). Picnic also kept for further research.

- It is likely we will keep several of them depending on how we want to use them (they have different pros and cons).
- The security analysis of each family are at different stages depending on the workload put into each of them. Some are old, others are new and of lesser standing.
- Let us have a look under the hood of some of these and see how they differ in practice.

- The McEliece/Niederreiter-system (1978/1986) based on error correcting codes.
- An error correcting code $\mathcal{C}$ is a method of adding redundancy to information so that we may detect and correct errors.
- A *linear* error correcting code encodes information as a linear subspace of some ambient space.
- There is an associated decoding algorithm $\mathcal{D}_{\mathcal{C}}$ to reverse the encoding process.

- Code-based cryptography supplies us with trapdoor one-way functions based on the fact that decoding general codes is hard if we do not know which code was used.
- On the other hand, the codes we use will usually have effective decoding algorithms up to some boundary on the number of errors. This means that the knowledge of the particular code will let us decode easily.
- In summary, the security of the setup we follow will hinge on two properties:
  1. Random codes are difficult to decode.
  2. The structure of the code we use may be obscured and difficult to differentiate from random.

- Not all instances of the decoding problem are hard: This depends on the code family and parameters.There are situations where one may solve this in polynomial time.
- This means we have to be careful in our choice. It is slightly miraculous that the original choice from the 70's is still alive with minor modifications.

- We fix notation and start with an $[n, k]$-code C over a field $\mathbf{F}_q$, a k-dimensional vector subspace of $\mathbf{F}_q^n$.
- C is defined by a $k \times n$ generator matrix G. We map a k-bit message $\mathbf{m}$ to a code word by

$$\mathbf{v} = \mathbf{m}G.$$

- Let $n - r = k$. As we know, C can also be defined through an $r \times n$ parity check matrix H so that

$$GH^T = 0.$$

$\mathbf{v} \in \mathbf{F}_q^n$ is a code word if and only if $\mathbf{v}H^T = 0$.

- To generate a key pair for the McEliece PKE, we generate
  - A generator matrix G $(k \times n)$ with an efficient decoding algorithm and the capability of correcting t errors.
  - A $k \times k$ matrix $S \in GL_k(\mathbf{F}_q)$.
  - A $n \times n$ permutation matrix P.
- The public key is then

$$(\hat{G} := SGP, t)$$

and the private key is

$$(S, G, P)$$

- The public key is equivalent to G, but obfuscated to hide the code (and hence the associated decoding algorithm) we are using.

- Alice will now encrypt a message $\mathbf{m} \in \mathbf{F}_q^k$:
  - She first encodes the message $\mathbf{m}\hat{G}$
  - She randomly chooses an error vector $\mathbf{e} \in \mathbf{F}_q^n$ of weight t and forms the ciphertext

$$\mathbf{c} = \mathbf{m}\hat{G} + \mathbf{e}.$$

- Bob decrypts as follows:
  - He first computes $\mathbf{c}P^{-1}$ to find

$$\mathbf{c}P^{-1} = \mathbf{m}\hat{G}P^{-1} + \mathbf{e}P^{-1} = \mathbf{m}SG + \mathbf{e}P^{-1}.$$

  - As $\mathbf{e}P^{-1}$ has weight t, he may decode $\mathbf{m}SG + \mathbf{e}P^{-1}$ to $\mathbf{m}S$ and right multiply with S^{-1} to find $\mathbf{m}$.

- As we can see, this leaves us with a choice of code for the generator matrix G. McEliece chose *binary Goppa codes* in his original proposal, and these are still viable choices under slight modifications to account for cryptanalysis[4].
- This choice leads to very large keys however, so a lot of effort has gone into finding codes that are more space efficient, but still secure. This turns out to be a much harder task than it seems, but there are still other candidates that *might* be of use[5][6].

[4] https://classic.mceliece.org/
[5] https://bikesuite.org/
[6] http://pqc-hqc.org/

- Not long after McEliece published his proposal, Niederreiter published a related encryption scheme[7].
- Here, one employs the dual code of G, encoding a message as an error pattern. This leads to a scheme that is more space-efficient at the cost of a slighty longer encryption process.
- In his original proposal, Niederreiter suggested we work with Generalized Reed-Solomon codes. However, this setup turned out to be weak[8].

[7] Niederreiter: Knapsack-type cryptosystems and algebraic coding theory

[8] Sidelnikov/Shestakov: On insecurity of cryptosystems based on generalized Reed-Solomon codes

- To generate a key pair for the Niederreiter PKE, we generate
  - A $r \times n$ parity check matrix H (recall that $r = n - k$) with an efficient decoding algorithm and the capability of correcting t errors.
  - A $r \times r$ matrix $S \in GL_r(\mathbf{F}_q)$.
  - A $n \times n$ permutation matrix P.
- The public key is then

$$(\hat{H} := SHP, t)$$

  and the private key is

$$(S, H, P)$$

- Again, the public key is equivalent to H, but hides the code we are using.

- Alice will now encrypt a message $\mathbf{m} \in \mathbf{F}_q^k$:
  - She first encodes the message as an error vector $\mathbf{e} \in \mathbf{F}_q^n$ such that $w(\mathbf{e}) = t$ (constant weight encoding algorithms come in different flavors[9]).
  - She then computes the ciphertext

$$\mathbf{c} = \hat{H}\mathbf{e}^T.$$

- Bob decrypts as follows:
  - He first computes

$$S^{-1}\mathbf{c} = HP\mathbf{e}^T.$$

  - As $P\mathbf{e}^T$ has weight t, he may perform syndrome decoding to find $P\mathbf{e}^T$ and right multiply with P^{-1} to find $\mathbf{e}^T$.

[9] Sendrier: Encoding information into constant-weight words

- Some notable differences between the McEliece/Niederreiter protocols:
  - The key size is different: G is of dimension $k \times n$ versus $(n-k) \times n$ for H, so we may choose H to make space problems less serious.
  - Coding messages as constant weight vectors

$$\mathbf{m} \longmapsto \mathbf{e}$$

makes the Niederreiter scheme more time consuming.

- There are two main ways of attacking the basic McEliece/Niederreiter schemes:
  1. We may try to figure out which particular code is in use from the public key.
  2. We may try to decode ciphertexts directly without knowing which code was in use.

  The available attacks are also split into structural versus generic attacks depending on the use of particular structure of the underlying code.
- In the first case there are *support splitting* algorithms, but these are not the fastest.
- For message recovery attacks, there are *information set decoding* (ISD)-algorithms. These are not very fast either, but faster than the former family. We will describe the simplest ways these apply.

- The basics of ISD-attacks goes back to the work of Prange in 1962[10].
- The idea is the following: We are given a coded message $\mathbf{c} = \mathbf{m}G + \mathbf{e}$ with $wt(\mathbf{e}) = t$.
  1. Choose $I \subset \{1, \ldots, n\} : |I| = k$ randomly (we are hoping for an error-free vector in this subset of coordinates of $\mathbf{c}$). Then form

     $$\mathbf{c}_I = \mathbf{m}G_I + \mathbf{e}_I$$

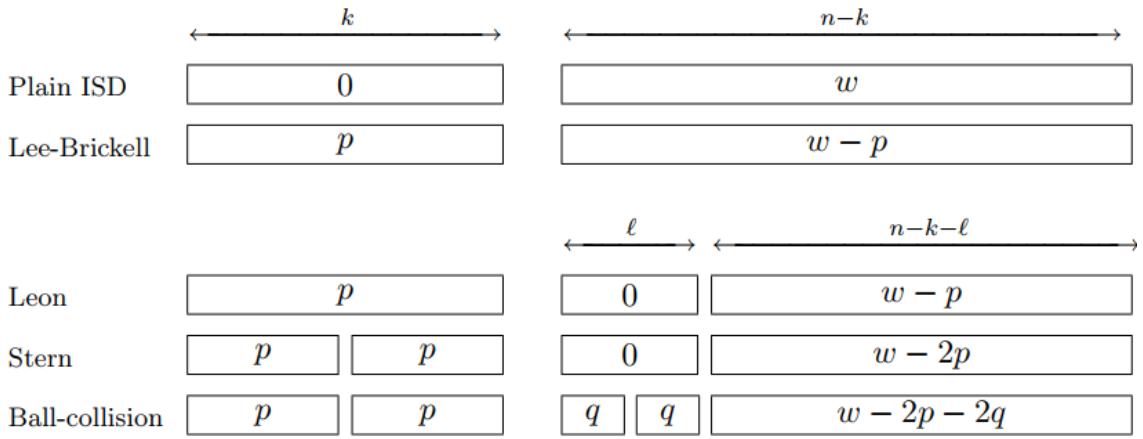
     (G_I consists of the matrix formed by the k columns we chose, likewise for $\mathbf{e}_I$).
  2. If $\mathbf{e}_I = 0$ and G_I is invertible, we find

     $$\mathbf{m} = \mathbf{c}_I G_I^{-1}.$$

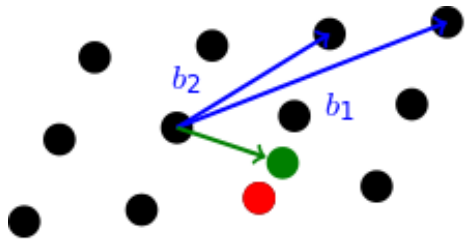
  3. If $\mathbf{e}_I \neq 0$ and G_I is *not* invertible, we try a new subset I of coordinates.

[10] Prange: The use of information sets in decoding cyclic codes

- There have been many refinements to ISD-analysis, each finding optimization techniques: It can be done by limiting the types of errors that occur in $\mathbf{e}$, the searching through the I-sets, the calculation of G_I^{-1}, etc.
- The error patterns of the separate ISD-attacks can be visualized such as this:



- May be based on several problems arising from the theory of lattices: LWE, LWR, NTRU.
- Security reduction to problems associated with the geometry of lattices:



- Are more space effective than code-based systems, but are usually structured so that security issues may arise. Further research is on the cards.
- Let us consider a simplistic version of NTRU to get a feeling for this.

- The NTRU-system was presented at CRYPTO '96 by Hoffstein, Pipher and Silverman. It consists of an encryption as well as a signature algorithm.
- To define NTRUencrypt, we start with some special polynomial rings. First, we fix primes N and p. Then, we let

$$\mathcal{R} := \mathbb{Z}[X]/(X^N - 1)$$

and

$$\mathcal{R}_p := \mathbb{Z}_p[X]/(X^N - 1).$$

- The projection $\mathbb{Z} \longrightarrow \mathbb{Z}_p$ sets up a homomorphism

$$\mathcal{R} \longrightarrow \mathcal{R}_p$$

by reducing coefficients modulo p.

- Considering inverse images under

$$\mathcal{R} \longrightarrow \mathcal{R}_p$$

leads to ambiguities. This acts as a lifting problem: There are several candidate preimages.

- Choosing which range of values we lift to gives us a chosen form of uniqueness: The *center lift* of $\mathbf{a}(x) \in \mathcal{R}_p$ is the unique element $\overline{\mathbf{a}}(x) \in \mathcal{R}$ such that its coefficients satisfy

$$-\frac{p}{2} < \overline{a}_i \leq \frac{p}{2}.$$

- Now, on to the description of NTRU. We will describe key generation, encryption and decryption.

  At the outset, we fix a quadruple (N, p, q, d) where N and p are primes and $\gcd(N, q) = \gcd(p, q) = 1$. These are all public parameters and will determine a chosen level of security.

  From here, we will need the rings $\mathcal{R}_p$ and $\mathcal{R}_q$.

- We need to bound the polynomials we are going to use. Let

$$\mathcal{T}(d_1, d_2) = \{\mathbf{a}(x) \in \mathcal{R} : \begin{cases} d_1 \text{ coefficients} = 1 \\ d_2 \text{ coefficients} = -1 \\ \text{other coefficients} = 0 \end{cases} \}$$

- Alice wants to set up key generation. She generates

$$\mathbf{f}(x) \in \mathcal{T}(d+1, d) \text{ and } \mathbf{g}(x) \in \mathcal{T}(d, d)$$

randomly.

- To generate the public key, Alice calculates $\mathbf{f}_p^{-1}(x) \in \mathcal{R}_p$ and $\mathbf{f}_q^{-1}(x) \in \mathcal{R}_q$, the inverses of $\mathbf{f}(x)$ in the respective rings (if they exist that is). She then calculates

$$\mathbf{h}(x) = \mathbf{f}_q^{-1}(x) \star \mathbf{g}(x) \in \mathcal{R}_q.$$

This serves as Alices's public key.

- The pair $(\mathbf{f}(x), \mathbf{f}_p^{-1}(x))$ forms Alice's private key.

- To encrypt a message, Bob first encodes his message as a polynomial $\mathbf{m}(x) \in \mathcal{R}$ with coefficients in the interval $\langle -\frac{p}{2}, \frac{p}{2}]$ (the center lift of some element in $\mathcal{R}_p$).
- He then chooses an ephemeral key $\mathbf{r}(x) \in \mathcal{T}(d, d)$ randomly and calculates

$$\mathbf{e}(x) \equiv p\mathbf{h}(x) \star \mathbf{r}(x) + \mathbf{m}(x) \pmod{q}$$

which is the ciphertext.

- On receiving the ciphertext, Alice first calculates

$$\begin{aligned}\mathbf{a}(x) &\equiv \mathbf{f}(x) \star \mathbf{e}(x) \\ &\equiv \mathbf{f}(x) \star (p\mathbf{h}(x) \star \mathbf{r}(x) + \mathbf{m}(x)) \\ &\equiv p\mathbf{g}(x) \star \mathbf{r}(x) + \mathbf{f}(x) \star \mathbf{m}(x) \pmod{q}.\end{aligned}$$

- She then computes

$$\mathbf{b}(x) \equiv \mathbf{a}(x) \equiv \mathbf{f}(x) \star \mathbf{m}(x) \pmod{p}$$

(note the modulus switch) and finally

$$\mathbf{c}(x) \equiv \mathbf{f}_p^{-1}(x) \star \mathbf{f}(x) \star \mathbf{m}(x) \equiv \mathbf{m}(x) \pmod{p}$$

to recover the plaintext.

- There is a snag we didn't mention: The decryption process is error-prone: If the coefficients of $\mathbf{a}(x)$ are too big, we might not get

$$\mathbf{c}(x) \equiv \mathbf{f}_p^{-1}(x) \star \mathbf{f}(x) \star \mathbf{m}(x) \equiv \mathbf{m}(x) \pmod{p}.$$

- We may handle this by assuming bounds like

$$q > (6d + 1)p$$

to limit the growth of coefficients.

- The imposed bound makes sure decryption is performed without failure. It is possible to use a less extreme bound at the cost of a positive probability for errors.

- So what is the most obvious way of attacking NTRU? We may always resort to brute force!
- From the formulation of NTRU, we know that there is a relation

$$\mathbf{h}(x) = \mathbf{f}_q^{-1}(x) \star \mathbf{g}(x) \in \mathcal{R}_q$$

or reformulated,

$$\mathbf{f}(x) \star \mathbf{h}(x) \equiv \mathbf{g}(x) \pmod{q}.$$

- The key recovery problem is finding $\mathbf{f}(x)$ and $\mathbf{g}(x)$ given $\mathbf{h}(x)$.
- Note that a solution pair need not be unique: Observe that for any solution $(\mathbf{f}(x), \mathbf{g}(x))$, $(x^k \star \mathbf{f}(x), x^k \star \mathbf{g}(x))$ $(0 \leq k < N)$ is also a solution to the key equation.

- An attacker can check if a candidate $\mathbf{f}(x)$ is correct by evaluating

$$\mathbf{f}(x) \star \mathbf{h}(x) \pmod{q}$$

and checking if this polynomial has coefficients in $\{-1, 0, 1\}$ (ternary). This will be a correct answer with high probability.

- Odlyzko has come up with a Meet-In-The-Middle attack which lowers the complexity of the brute force method.
- The idea is that one may split $\mathbf{f}(x)$ into a sum of two simpler polynomials $\mathbf{f}_i(x)$ (of degree at most $\frac{N}{2} - 1$ and of degrees between $\frac{N}{2}$ and $N - 1$ respectively), store the results of

$$\mathbf{f}_1(x) \star \mathbf{h}(x), -\mathbf{f}_2(x) \star \mathbf{h}(x) \pmod{q},$$

and check if

$$\mathbf{f}_1(x) \star \mathbf{h}(x) \approx -\mathbf{f}_2(x) \star \mathbf{h}(x) \pmod{q}$$

- This leads to finding a candidate key pair since we know that

$$\mathbf{f}_1(x) \star \mathbf{h}(x) \equiv \mathbf{g}(x) - \mathbf{f}_2(x) \star \mathbf{h}(x) \pmod{q}$$

for a valid key pair ($\mathbf{g}(x)$ has small coefficients).

- There is another way to attack the key recovery problem of NTRU. The relation

$$\mathbf{f}(x) \star \mathbf{h}(x) = \mathbf{g}(x) \pmod{q}$$

can be interpreted using a lattice.

- Consider the $2N \times 2N$-matrix

$$M_{\mathbf{h}}^{NTRU} = \begin{bmatrix} I & \mathbf{h} \\ \mathbf{0} & qI \end{bmatrix}$$

where I is an identity matrix, $\mathbf{h}$ is the matrix obtained by rotating the coefficients of the underlying polynomial $\mathbf{h}(x)$.

- We can form a lattice by considering integer combinations of the rows of $M_{\mathbf{h}}^{NTRU}$. We will refer to it as the *NTRU-lattice*.
- Now, let

$$\mathbf{f}(x) \star \mathbf{h}(x) = \mathbf{g}(x) \pmod{q}$$

and $\mathbf{u}(x)$ be chosen so that

$$\mathbf{f}(x) \star \mathbf{h}(x) = \mathbf{g}(x) + q\mathbf{u}(x).$$

An easy check shows that

$$(\mathbf{f}, -\mathbf{u}) M_{\mathbf{h}}^{NTRU} = (\mathbf{f}, \mathbf{f} \star \mathbf{h} - q\mathbf{u}) = (\mathbf{f}, \mathbf{g}).$$

This means that $(\mathbf{f}, \mathbf{g})$ is an element in the NTRU-lattice.

- Since $\mathbf{f}$ and $\mathbf{g}$ are very sparse vectors, this makes for a *short* vector in $\mathbb{R}^{2N}$.
- The problem of finding short vectors in a lattice is classical and it turns out that this is very hard given proper parameters.

- Recall that a lattice $L \subset \mathbb{R}^n$ is a subgroup (under addition) isomorphic to $\mathbb{Z}^n$. It can be represented as

$$L = \{a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n \mid a_i \in \mathbb{Z}\},$$

  where the $\mathbf{v}_i$ are linearly independent.
- Lattices have been studied for many different reasons. Apart from cryptography, they show up in number theory, Lie algebras, group theory and physics among other areas.
- We will look into important computational problems that can be applied in cryptography.

- First, we define the *the shortest vector problem* (SVP) in L: This is the problem of finding a nonzero vector $\mathbf{v} \in L \subset \mathbb{R}^n$ that minimizes $\|\mathbf{v}\|$.
- Secondly there is the *the closest vector problem* (CVP) in L: Given a $\mathbf{w} \in \mathbb{R}^n$, this is the problem of finding a vector $\mathbf{v} \in L$ such that $\|\mathbf{v} - \mathbf{w}\|$ is minimal.
- In both cases, there need not be a unique answer. The difficulty of both problems increase rapidly when n grows.

  The length of the shortest vector in a lattice L is usually denoted $\lambda_1(L)$ (the first successive minimum). The minimal distance from a target vector $\mathbf{w}$ to the lattice is denoted $d(L, \mathbf{w})$.

- The *approximate shortest vector problem* (SVP_γ) in L is defined using a given approximation factor. Here, we wish to find a nonzero vector $\mathbf{v} \in L \subset \mathbb{R}^n$ such that

$$\|\mathbf{v}\| \leq \gamma \lambda_1(L)$$

  where $\mathbf{v}_{shortest}$ solves the SVP in L.
- Similarly, there is the *approximate closest vector problem* (CVP_γ), we wish to find a nonzero vector $\mathbf{v}' \in L \subset \mathbb{R}^n$ such that

$$\|\mathbf{v}' - \mathbf{w}\| \leq \gamma d(L, \mathbf{w}).$$

- The difficulty of these problems vary from trivial to extremely hard depending on γ.

- Let us turn to the SVP. How short is the shortest vector of a lattice L? First of all, there is *Hermite's theorem* stating that for every L of dimension n, there is a nonzero vector $\mathbf{v} \in L$ satisfying

$$\|\mathbf{v}\| \leq \sqrt{n} \det(L)^{1/n}.$$

- Here, $\det(L)$ is the determinant of the matrix with basis vectors of L as rows. This turns out to be an invariant of L.

- It is possible to improve this with the *Gaussian heuristic*, saying that a random lattice will have some element $\mathbf{v}$ satisfying

$$\|\mathbf{v}\| \approx \sigma(L)$$

where

$$\sigma(L) = \sqrt{\frac{n}{2\pi e}}(\det(L))^{1/n}.$$

- There is also a Gaussian heuristic for the CVP: Here, a random lattice $L \subset \mathbb{R}^n$ with some random vector $\mathbf{w} \in \mathbb{R}^n$ will have some $\mathbf{v} \in L$ satisfying

$$\|\mathbf{v} - \mathbf{w}\| \approx \sigma(L).$$

- These are truly heuristic measures:
  From a given lattice, it is not necessarily easy to determine exactly how short a shortest or how close a closest vector will be.
- In practice, experiments show that the Gaussian heuristic is closer to the truth than Hermite's boundary.
- Finding the private key $(\mathbf{f}(x), \mathbf{g}(x))$ is possible if we can solve the SVP in $L_{\mathbf{h}}^{NTRU}$ with high probability. This is why we are interested in lattice reduction algorithms such as LLL/BKZ which is the go-to solution today.

- Hash-based signatures are, as the name suggests, based on cryptographic hash-functions to establish security.
- The story begins with the advent of one-time signatures. This leads to obvious size problems, so the next steps are all about efficiency.
- In our case, we will have a look at some propositions for one-time signatures, and then move on to combinatory techniques and tree-based structures.
- Although quantum algorithms have not been studied for eons, none have been found that break the security of such functions beyond practical use.

- The Lamport one-time signature scheme uses a cryptographic hash function to produce signatures.
- Let us assume we have a hash function

$$f : X \to Y.$$

- If we want to sign one bit b, we randomly pick $(x_0, x_1) \in X^2$ (the secret key) and compute $(y_0 = f(x_0), y_1 = f(x_1)) \in Y^2$ (the public key).
- The signing rule is the following: $sig = x_0$ if $b = 0$ and $sig = x_1$ if $b = 1$.
- To verify the signature, the receiver checks that $f(sig) = y_b$.

- This is easy to generalize to larger messages.
- If we want to sign a k-bit message

$$m = b_0 \cdots b_{k-1},$$

we repeat the one-bit procedure for each bit, randomly picking $(x_{i0}, x_{i1}) \in X^2$ $(0 \leq i < k)$. Then we compute $(y_{i0} = f(x_{i0}), y_{i1} = f(x_{i1})) \in Y^2$ for each i.

- The signing rule is as before

$$sig_i = \begin{cases} x_{i0} & \text{if } b_i = 0 \\ x_{i1} & \text{if } b_i = 1. \end{cases}$$

- To verify the signature, the receiver checks that $f(sig_i) = y_{ib}$ for each i.

- There are obvious drawbacks deriving from this scheme:
  - As k grows larger, the size of the signatures and keys grow large too. There are ways to deal with this.
  - We may only use a signature once since an attacker may forge a valid signature otherwise (the attacker will have a choice of value for each time a reuse occurs). There are ways to deal with this too.

- The Winternitz signature scheme is a way to create a trade-off between space and time starting with the Lamport signature scheme.
- The basic idea is that we may form groups of message bits and sign these instead of individual bits to shorten the number of signatures. From here, let us assume we want to sign w bits at a time.

- At first glance, this seems to be a bad idea:
  In Lamport's scheme, there were two lists of bitstrings, one for each bit-value of the message. If we want to do the same for w bits at a time, we would have to create 2^w lists for each group we want to sign, causing a blow-up in the size of secret/public keys.
- Winternitz came up with an elegant solution to this. Instead of generating a large set of random lists, he suggests we generate them from hashing as needed.

- This begins with generating a list of random seeds and utilizing a given hash-function f to make hash-tables:

$$(x_i^j), 0 \leq j < 2^w.$$

Here, we pass from each j-level to the next by hashing, i.e, we let $x_i^j = f(x_i^{j-1})$.

- The secret key is then the very first list

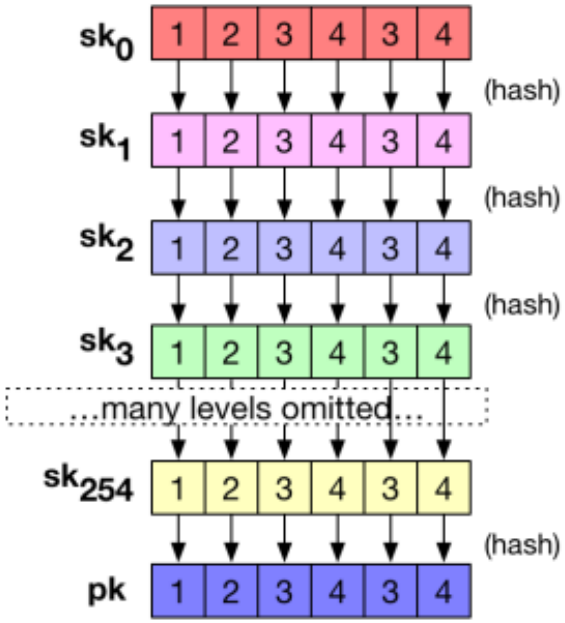
$$(x_i^0)$$

which is generated randomly.

- Now, we let the public key be the list obtained at the very end. In other words, the list

$$(y_i = f^{2^w-1}(x_i^0)).$$

- From the setup, we see that this makes for lists of secret/public keys that are much smaller than in Lamport's signature.
- If we use any of the possible values for the secret key, we get to the public key value by hashing an appropriate number of times:



- In the illustration, $w = 8$. To sign a byte, we would choose an element from the appropriate level: A byte of value 0 would be signed with an element from sk_0, a byte of value 1 would be signed with an element from sk_1 and so on.
- The problem for this basic setup is the relation between the lists: An attacker knows that the secret keys are related through the hash function f.

- This means the attacker may increment the value of any message byte, hash an appropriate number of times and get a valid signature!
- The solution lies in the introduction of a checksum which is paramount to the security of the scheme!

- Now, we want to sign message

$$m = b_0 \cdots b_{k-1}.$$

For this, we split the message into groups of size w, padding it from the left with 0s so we get $t_1 = \lceil k/w \rceil$ blocks $B_1, \ldots, B_{t_1}$.

- At this point, we treat the blocks B_i as integers in $\{0, \ldots 2^w - 1\}$ and form the checksum

$$C = \sum_{i=1}^{t_1} (2^w - B_i).$$

Notice that any increase in the B_i will change the sum.

- Since $C \leq t_1 2^w$, its binary representation has length at most

$$\lfloor \log_2(t_1 2^w) \rfloor + 1 = \lfloor \log_2(t_1) \rfloor + w + 1.$$

- We pad C from the left with the minimum number of zeros so that the padded checksum (in binary) is divisible by w. Thereafter, we form the final blocks $B_{t_1+1}, \ldots, B_t$ where $t = t_1 + t_2$ and

$$t_2 = \left\lceil \frac{\lfloor \log_2(t_1) \rfloor + w + 1}{w} \right\rceil.$$

- Having chosen a cryptographic hash function f (with s-bit output size) and w, we choose t random s-bit strings

$$(x_i), 0 \leq i < t$$

as our secret key.

- Then, we publish the list of strings

$$(y_i = f^{2^w - 1}(x_i)), 0 \leq i < t$$

as our public key.

- For a given message, we pad, get an extended $m = B_1 \| \cdots \| B_{t_1}$. Then we form the checksum $C = B_{t_1+1} \| \cdots \| B_t$.
- Our signature will be

$$sig = (sig_1 \| \cdots \| sig_t)$$

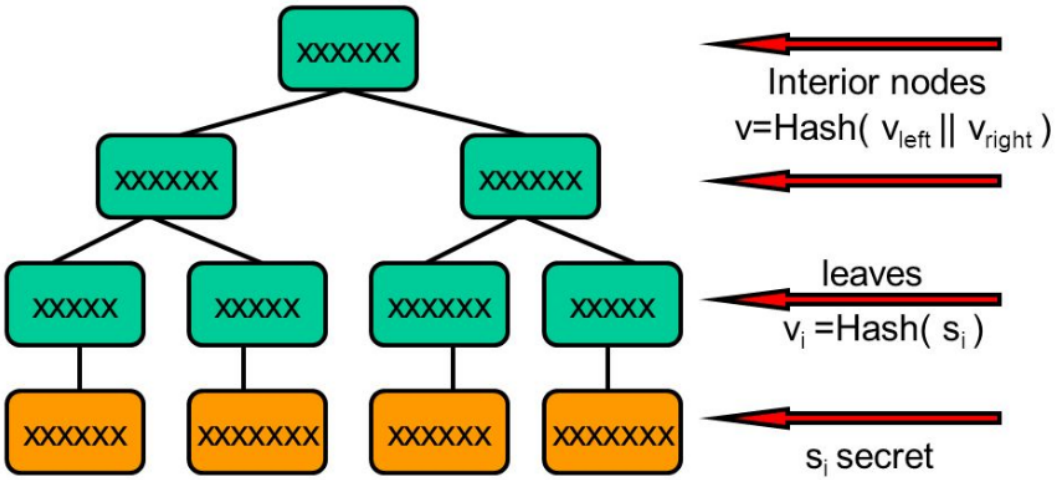
where $sig_i = f^{B_i}(x_i)$.

- For the verifier, the job is now easy. He first computes the blocks B_i for $0 \leq i < t$ as before.
- Then he checks, for each i, that

$$f^{2^w-1-B_i}(sig_i) = f^{2^w-1-B_i}(f^{B_i}(x_i)) = f^{2^w-1}(x_i) = y_i.$$

- The Lamport signature (or the other one-time versions for that matter) had another big drawback: We could only use a signature once.
- Obviously, we could generate a large number of one-time signatures (OTS) and concatenate all the public keys into *one* single public key. This key would then be very large.
- Merkle came up with a solution to this: Tree-based hashing. We are going to set up a binary tree that permits verification of a given set of signatures for *one* public key with a much smaller footprint.

- A priori, we decide on a number of messages to be signed, $N = 2^n$ say.
- Then, we choose a OTS-scheme with an associated cryptographic hash function f.
- Once this is done, we create N separate OTS pairs $x_i, y_i = f(x_i)$. Then, we create a Merkle tree as in the following figure:



- First of all, the public key is the element at the very top (the root).
- How is the tree constructed? We employ the hash function f:
  - At the very bottom, we hash the secret keys x_i. These hashes serve as the bottom leaves.
  - Then, pairs of leaves are hashed to obtain a superior node.
  - This continues until we reach the top node which serves as the public key.

- Here is how we create a signature from a message m:
  - First, a bottom leaf is chosen with an associated secret/public key pair (x_i, y_i). This is then used to create the first part of the signature sig_0 from the message m.
  - After this, the rest of the signature consists of all nodes needed to find the unique path to the public key hashing oneself up the tree.
  - This path consists of $n+1$ nodes A_i, and we use neighboring nodes B_i to move to the next level so that $A_{i+1} = f(A_i \| B_i)$.
- The final signature is the concatenation

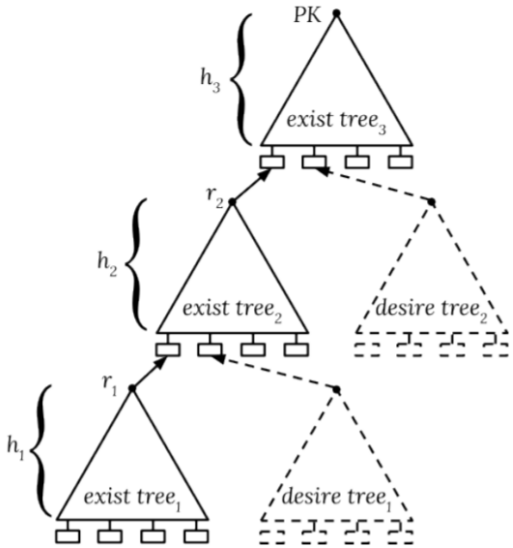
$$sig = (sig_0 \| B_2 \| B_3 \cdots \| B_{n-1}).$$

- Verifying a signature is now simple:
  - The receiver begins by checking that message m produces signature sig_0.
  - If this is so, he then computes $f(y_i)$ and hashes his way to the top level of the tree, checking that the correct public key is produced.

- The advantages of Merkle trees come with some caveats: Computational effort and signature length.
  - To generate the public key, 2^n OTS keys must be generated.
  - Then, every node of the tree must be computed. This means we need to compute $2^{n+1} - 1$ hash operations, one for each node.
  - Generating a signature required the B_i-nodes. If the nodes of the tree are not stored, these will have to be regenerated for every signature.

- Generating the tree is very expensive, and generating a very large tree is impractical.
- However, saving all $2^{n+1} - 1$ nodes would quickly lead to storage problems.
- Hence, dealing with both these problems needs a creative strategy.

- There have been several suggestions of how to improve the situation:
  - Instead of computing one big tree, we could generate subtrees of smaller size and produce tree chains. The leaves of the main tree are used to sign roots of lower level trees which contain OTS.



- Here, the signature consists of:
  - A chosen signing key at the bottom
  - The signing keys used to sign the roots connecting the trees
  - The paths taken through each tree
- Using multiple levels, we can vary the size of the trees that need to be generated when signing, so storage and time consumption can be adjusted to suitable levels.

- Some hash-based signature schemes are *stateful* in that the signer must keep track of the number of messages that has been signed before:
  - Recall that we may not use a OTS at the bottom two times!
  - This is more arduous than it seems: Prone to programming failures, hardware failures, other glitches..
- There are two proposed schemes for this setup: LMS and XMSS:
  - Both are in line for standardization in IETF/NIST processes.

- One of the signature schemes suggested for the NIST PQC project, is SPHINCS+, a *stateless* hash-based signature scheme.

  Here, the idea is to use dynamic trees, except that we randomly choose which leaf OTS we use instead of choosing them in order. The idea is that a large tree will make reusing an OTS highly improbable and do away with any state to keep track of.

- Some observations:
  - Keys/signatures/ciphertext can become large in many of these algorithms (figures in bytes for Classic McEliece[11] og SPHINCS+[12]):

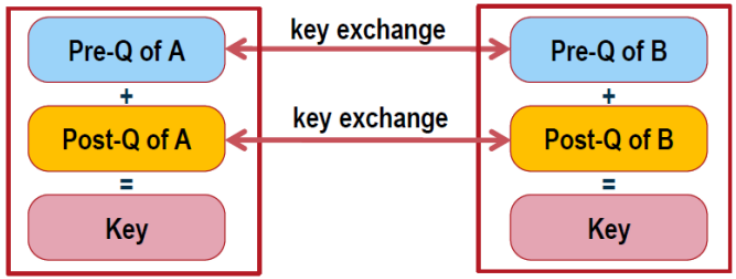
	Public key	Private key	Ciphertext	Session key
mceliece348864	261120	6452	128	32
mceliece460896	524160	13568	188	32
mceliece6688128	1044992	13892	240	32
mceliece6960119	1047319	13908	226	32
mceliece8192128	1357824	14080	240	32

	public key size	secret key size	signature size
SPHINCS$^+$-128s	32	64	8 080
SPHINCS$^+$-128f	32	64	16 976
SPHINCS$^+$-192s	48	96	17 064
SPHINCS$^+$-192f	48	96	35 664
SPHINCS$^+$-256s	64	128	29 792
SPHINCS$^+$-256f	64	128	49 216

[11] https://classic.mceliece.org/nist/mceliece-20190331.pdf

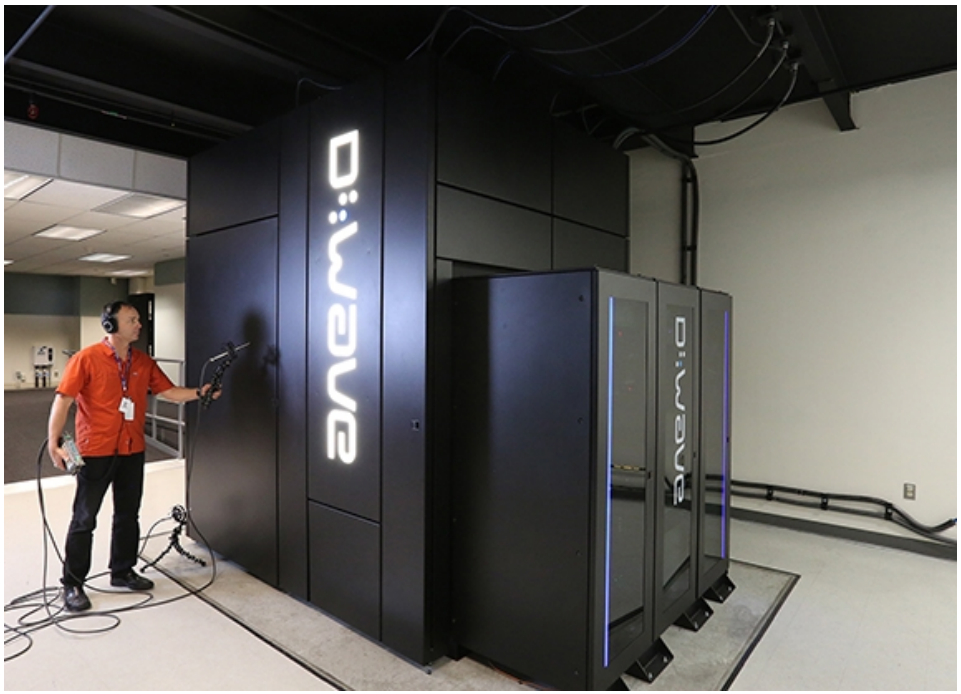
[12] https://sphincs.org/data/sphincs+-round2-specification.pdf

- We may consider hybrids of quantum resistant *and* classical primitives.



- This adds complexity to the protocols we use and we must analyse what consequences this has in terms of security and efficiency.
- Depending on the limitations in our protocols, we must choose which primitives it is possible to fit.

- There is lots of research and development to realize quantum circuits, but how advanced are they really?
- PR is abundant with many nice pictures...[13]:



[13] https://www.dwavesys.com

- With many possible paths of constructions, it seems silly not to take any of them seriously.
- We need more research to see stable quantum circuits that scale (logical versus physical qubits).
- There is ample funding in this area (Google, IBM, Lockheed Martin..).

- It is not easy to determine *when* a fully functional quantum computer is ready.
- At this point, the players in the field estimate sometime near 2030.
- For us, there is the more important question of finding good replacements for todays primitives which need integration and testing.

Rounding up

- Quantum algorithms forces us to look for new cryptographic primitives.
- We are not sure exactly when we need them, but we are closing in on the solutions.
- Planning ahead for their introduction is imperative, so see to it that they fit where applicable.

Thank you very much!