The effect of quantum algorithms

CBDC 2022

T.Gregersen

Todays menu

Introduction

The impact on cryptography today

Symmetric cryptography

Asymmetric means

The quantum threat

Grover's algorithm

Shor's algorithm

The relevant consequences

Post quantum cryptography

Code based techniques

Hash-based signatures

Are there consequences for digital currencies?

- Cryptography is omnipresent: This is how we handle confidentiality, authenticity and integrity of information in most cases.
- Quantum computers *might* prove to be a problem for our reliance on cryptography, the building blocks (primitives) of today are in question.
- The idea of these machines have been around for a long time, they might not 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.
    - There has been an extensive amount of work to analyse what to use in the future, we should all consider the consequences.

Introduction

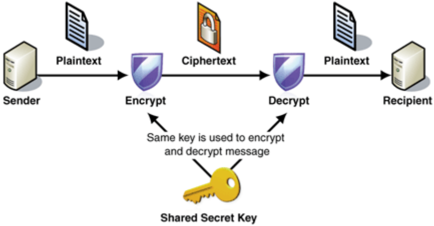
The impact on cryptography today
- Symmetric cryptography
- Asymmetric means

The quantum threat
- Grover's algorithm
- Shor's algorithm
- The relevant consequences

Post quantum cryptography
- Code based techniques
- Hash-based signatures

Are there consequences for digital currencies?

- The basis of most solutions:



- There is Grover's algorithm, but this is possible to handle through extending key sizes.

Introduction

The impact on cryptography today

- Symmetric cryptography
- Asymmetric means

The quantum threat

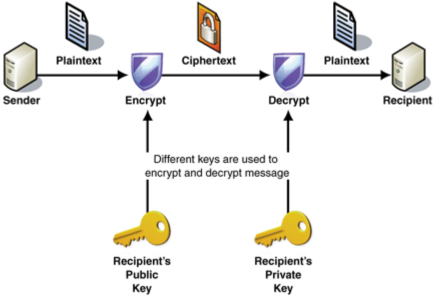
- Grover's algorithm
- Shor's algorithm
- The relevant consequences

Post quantum cryptography

- Code based techniques
- Hash-based signatures

Are there consequences for digital currencies?

- Key establishment, signatures:



- Shor's algorithm might turn out to be disastrous, this depends on the scale of and to what extent we may control quantum circuits (there are several possible constructions).

- Todays smorgasboard of primitives: RSA, DH, ECDH (and associated signature algorithms).
- 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 limitations).

Introduction

The impact on cryptography today

- Symmetric cryptography
- Asymmetric means

The quantum threat

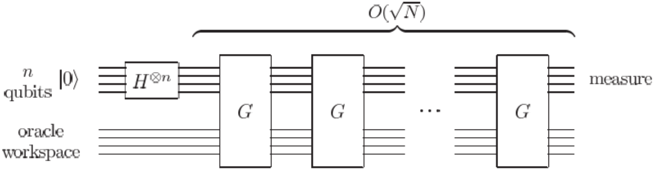
- Grover's algorithm
- Shor's algorithm
- The relevant consequences

Post quantum cryptography

- Code based techniques
- Hash-based signatures

Are there consequences for digital currencies?

- 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 a diagram showing the basic workflow:



Introduction

The impact on cryptography today

- Symmetric cryptography
- Asymmetric means

The quantum threat

- Grover's algorithm
- Shor's algorithm
- The relevant consequences

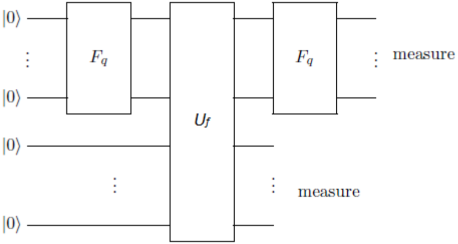
Post quantum cryptography

- Code based techniques
- Hash-based signatures

Are there consequences for digital currencies?

- 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:



Introduction

The impact on cryptography today

- Symmetric cryptography
- Asymmetric means

The quantum threat

- Grover's algorithm
- Shor's algorithm
- The relevant consequences

Post quantum cryptography

- Code based techniques
- Hash-based signatures

Are there consequences for digital currencies?

- A pessimistic view forces us to expand the size of keys, but it is not yet clear by how much.
- 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 in the works:
  - 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[1].

[1] https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

- What properties should replacement algorithms have?
  - Keys/signatures/ciphertext should show some sort of efficiency with regards to space/time constraints.
  - Constructions that show strong security foundations.
  - Implementation issues should be addressed to avoid as many practical problems as possible.
- There are candidates, but combining all this isn't necessarily easy.

- NIST started with a bunch of candidate algorithms for key establishment and signatures. Over time, many have fallen to cryptanalysis.
- Examples of KEMs:
  - 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:
  - CRYSTALS-DILITHIUM/FALCON/qTesla.
  - GeMSS/LUOV/MQDSS/Rainbow.
  - Picnic.
  - SPHINCS+.

- Choices have been made:
  - Kyber has been chosen for standardization (key establishment).
  - Dilithium/Falcon/SPHINCS+ have been chosen for standardization (signatures).
- There will be a fourth round of selection, BIKE/ClassicMcEliece/HQC are still open for consideration.

- A single primitive for standardization is preferable, but considering the properties of the candidates, choices might depend on use cases (they have different pros and cons).
- The security analysis of each family are at different stages: Some are old, others are new and have a lighter track record.
- Only time will tell which primitives will survive in the long run, but we are gaining confidence in the choices made.

- The McEliece/Niederreiter-system (1978/1986) based on error correcting codes.
- An error correcting code $\mathcal{C}$ is a method for adding redundancy to information so that we may detect and correct errors.
- 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.

- 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[2].
- 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[3][4].

[2] https://classic.mceliece.org/

[3] https://bikesuite.org/

[4] http://pqc-hqc.org/

- Hash-based signatures are, as the name suggests, based on cryptographic hash-functions to establish security.
- 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

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

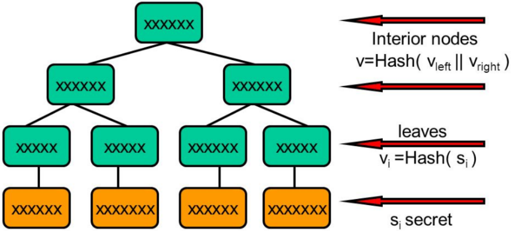
- To verify the signature, the receiver checks that $f(sig_i) = y_{ib}$ for each i.

- 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.

- The Lamport signature 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.

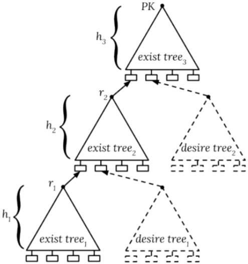
- Here is how we create a signature from a message m:
  - First, a bottom leaf is chosen with an associated secret/public key pair (x_i, y_i). This is then used to create the first part of the signature sig_0 from the message m.
  - After this, the rest of the signature consists of all nodes needed to 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.

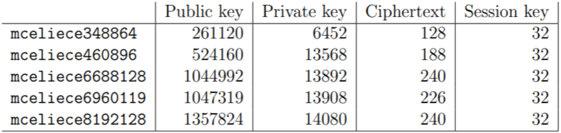
- The advantages of Merkle trees come with some caveats: Computational effort and signature length.
  - To generate the public key, 2^n OTS keys must be generated.
  - Then, every node of the tree must be computed. This means we need to compute $2^{n+1} - 1$ hash operations, one for each node.
  - Generating 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.

- 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.

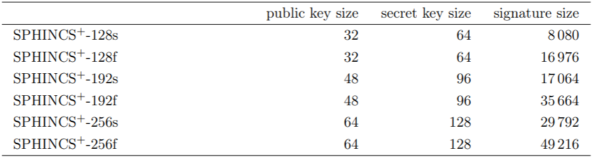


- Using multiple levels, we can vary the size of the trees that need to be generated when signing, so storage and time consumption can be adjusted to suitable levels.
- One of the signature schemes suggested for the NIST PQC project, is SPHINCS+, a *stateless* hash-based signature scheme.

- Some observations:
  - Keys/signatures/ciphertext can become large in many of these algorithms (figures in bytes for Classic McEliece[5] og SPHINCS+[6]):



	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



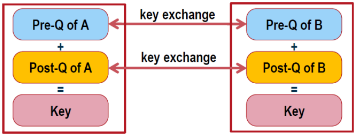
	public key size	secret key size	signature size
SPHINCS$^+$-128s	32	64	8 080
SPHINCS$^+$-128f	32	64	16 976
SPHINCS$^+$-192s	48	96	17 064
SPHINCS$^+$-192f	48	96	35 664
SPHINCS$^+$-256s	64	128	29 792
SPHINCS$^+$-256f	64	128	49 216

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

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

- These primitives highlight the differences in size compared with todays workhorses.
- Protocol designers have to make trade-offs and choose algorithms that match their application scenarios.
- In time, we will have primitives to rely on, but we may also find ourselves in a situation where standard primitives fall away to cryptanalysis. Be open to changes if need be!

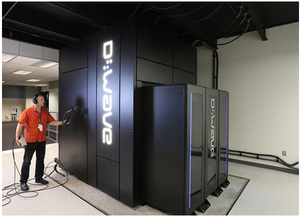
- Until we are sure we have quantum safe cryptography, we may consider hybrids of quantum resistant *and* classical primitives.



- This adds even more 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.

- Designing a protocol for digital currencies will need signatures to identify parties and authorize transactions, hence quantum safe solutions should be incorporated.
- There might be other implications (I am no expert), so make sure to do research into what problems this might imply.

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



- What we need for this to happen:
  - Scalability of memory
  - Qubits that can be initialized to relevant values
  - Quantum gates that are faster than some 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.

- With many possible paths of constructions, it seems silly not to take the threat of quantum computing 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, Honeywell..).

- 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!