Introduction to Post-Quantum Cryptography

Matthias J. Kannwischer
Radboud University, Nijmegen, The Netherlands
matthias@kannwischer.eu

March 2019, DHBW Stuttgart
About me

- Matthias Kannwischer
  - kannwischer.eu
- PhD Candidate at Radboud University
  - Nijmegen, The Netherlands
- Supervisor: Peter Schwabe
  - cryptojedi.org
- 2012-2015: BSc CS at DHBW Stuttgart
- 2015-2017: MSc ITSec at TU Darmstadt
- I work on Post-Quantum Cryptography...
Post-quantum cryptography (PQC) is public-key cryptography (PKC) that is resistant against attacks by classical computers and quantum computers.

- PQC is aka quantum-resistant cryptography or quantum-safe cryptography
- PKC is also called asymmetric cryptography
- Crypto means cryptography!
Let’s take one step back:

What is cryptography and why do we need it?
Cryptography Use-Cases

- Transport Layer Security (TLS) – e.g. in HTTPS
- IPSec – e.g. for VPNs
- End-to-end messenger encryption: Signal, WhatsApp, ...
- Encrypted calls: Skype, Facetime, ...
- Harddisk encryption
- E-mail encryption/signatures: PGP, S/MIME
- SIM-cards
- Software updates
- Credit- and debit cards: Chip-payments, EMV, EMV2
- Content protection: DRM, PayTV, Games
- Tor browser
- Access control
- ...

- Some protect us, others protect industry from us
Cryptography Use-Cases

- Crypto is everywhere...
“Quantum computers break all cryptography” — The Internet

- not actually true... as we will see...
What is a Quantum Computer?

Quantum Computer or Microbrewery?

Slide credit: Steven Galbraith (@EllipticKiwi)
Why do we care?

- Quantum computers solve a lot of problems
- **But:** Complexity-based cryptography is based upon the hardness of certain problems
- Two quantum algorithms threaten the security of our communications:
  - Shor’s algorithm (1994)
  - Grover’s algorithm (1996)
A fast quantum mechanical algorithm for database search

Lov K. Grover
3C-404A, Bell Labs
600 Mountain Avenue
Murray Hill NJ 07974
lkgrover@bell-labs.com

Summary
Imagine a phone directory containing $N$ names arranged in completely random order. In order to find someone's phone number with a probability of $\frac{1}{2}$, any classical algorithm (whether deterministic or probabilistic) will need to look at a minimum of $\frac{N}{2}$ names. Quantum mechanical systems can be in a superposition of states and simultaneously examine multiple names. By properly adjusting the phases of various operations, successful computations reinforce each other while others interfere randomly. As a result, the desired phone number can be obtained in only $O(\sqrt{N})$ steps. The algorithm is within a small constant factor of the fastest possible quantum mechanical algorithm.

This paper applies quantum computing to a mundane problem in information processing and presents an algorithm that is significantly faster than any classical algorithm can be. The problem is this: there is an unsorted database containing $N$ items out of which just one item satisfies a given condition - that one item has to be retrieved. Once an item is examined, it is possible to tell whether or not it satisfies the condition in one step. However, there does not exist any sorting on the database that would aid its selection. The most efficient classical algorithm for this is to examine the items in the database one by one. If an item satisfies the required condition stop; if it does not, keep track of this item so that it is not examined again. It is easily seen that this algorithm will need to look at an average of $\frac{N}{2}$ items before finding the desired item.
1996: Grover’s algorithm

- Grover’s algorithm allows to search a database with $n$ elements in $\sqrt{n}$ steps
- Imagine you have a 10 digit (random) password
  - Brute forcing this classically would take $10^{10}$ steps
  - Grover can do this in $10^5$ steps
- Breaking AES-128 (block cipher with a 128-bit key)
  - Classically: $O(2^{128})$
  - Quantum: $O(2^{64}) \rightarrow$ maybe still ok? on the edge
- **Fix:** Double the key length and it’s fine
  - Symmetric key crypto is still fine
  - Switch to AES-256
  - And, please stop using DES, 3DES, SHA-1, ...
Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer

Peter W. Shor

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

Keywords: algorithmic number theory, prime factorization, discrete logarithms, Church’s thesis, quantum computers, foundations of quantum mechanics, spin systems, Fourier transforms
1994: Shor’s algorithm

- Shor’s algorithm allows factoring large numbers and solving the discrete logarithm problem
  - breaks RSA
  - breaks DSA, ElGamal, Diffie-Hellman..
  - breaks elliptic curve crypto (ECDSA, ECDH, ...)

- **This cannot be fixed by using larger keys!**
- Before we look at what this means for crypto...
  - Shor’s algorithm requires thousands of qubits
  - When will we have that?
When will a large enough quantum computer exist?

- Some say never...
- Some say the NSA already has one...
- Most believe in the next 10-20 years...
When will a large enough quantum computer exist?
Let’s look at a wide-spread example: TLS in HTTPS
Pre-Quantum Cryptography
Security overview

This page is secure (valid HTTPS).

- **Certificate - valid and trusted**
  The connection to this site is using a valid, trusted server certificate issued by Let's Encrypt Authority X3.
  
  View certificate

- **Connection - secure (strong TLS 1.2)**
  The connection to this site is encrypted and authenticated using TLS 1.2 (a strong protocol), ECDHE_RSA with P-256 (a strong key exchange), and AES_256_GCM (a strong cipher).

- **Resources - all served securely**
  All resources on this page are served securely.
Certificate Viewer: kannwischer.eu

This certificate has been verified for the following usages:

SSL Server Certificate

Issued To

Common Name (CN)       kannwischer.eu
Organisation (O)        <Not Part Of Certificate>
Organisational Unit (OU)<Not Part Of Certificate>

Issued By

Common Name (CN)       Let's Encrypt Authority X3
Organisation (O)       Let's Encrypt
Organisational Unit (OU)<Not Part Of Certificate>

Validity Period

Issued On             Monday, 4 February 2019 at 14:51:34
Expires On            Sunday, 5 May 2019 at 15:51:34

Fingerprints

SHA-256 Fingerprint    F2 FC 2F A7 FD 3B 32 66 20 0F 84 D9 F5 E5 97 E0 5C 57 C4 5D 7A 3B 63 F4 E9 55 89 8E 87 CB 0A BA
SHA-1 Fingerprint      46 F6 8F D5 96 9A 8A 00 11 59 16 77 D4 A1 56 9B DC 46 4E 72
### Certificate Viewer: kannwischer.eu

#### Certificate Hierarchy
- **Built-in Object Token:** DST Root CA X3
  - Let's Encrypt Authority X3

#### Certificate Fields
- **Version**
- **Serial Number**
- **Certificate Signature Algorithm**
  - Issuer
  - Validity
    - Not Before
    - Not After
  - Subject
  - Subject Public Key Info

#### Field Value
- PKCS #1 SHA-256 With RSA Encryption
Crypto used when connecting to kannwischer.eu

- **Protocol:** TLS 1.2
- **Key Exchange:** ECDHE
  - Ephemeral Elliptic Curve Diffie-Hellman
  - Curve: P-256 (standardized by NIST)
- **Diffie-Hellmann share authentication:** RSA
  - To protect against man-in-the-middle attacks
  - Hash function: SHA-256 (SHA2)
- **Data encryption:** AES-256
  - Key length: 256 bit
- **Data integrity:** AES-GCM
  - Mode of operation: Galois/Counter Mode
  - Ensures that encrypted data is not modified
AES-256 is fine (Grover: $O(2^{128})$)

AES-GCM is fine

SHA-256 (SHA-2) is fine, but please use SHA-3 if you can

ECDHE is broken by Shor

RSA is broken by Shor

... the key exchange and signature part is broken.

... this part is essential

Essentially the same for all TLS connections (and a lot more)
When do we care about quantum attacks

- Attack scenario
  - Record encrypted traffic now; decrypt in 20 years
- We want post-quantum encryption as soon as possible
- Signatures are less critical (at least for TLS)
  - Only needs to be unbroken at the time of verification
  - We don’t use many long-term signatures
- We want post-quantum signatures *deployed* before *anyone* has a quantum-computer
TIME TO PANIC
Post-Quantum Crypto in 2019
Post-quantum cryptography is public key cryptography that is resistant against attacks by classical computers and quantum computers.

- Integer factorization and computing discrete logarithms is easy
- We need replacement for signatures and key exchange/key encapsulation
- We need different problems to base our cryptography on
  - Lattices (Gitter) \( as + e \leftrightarrow s \)
  - Hashes \( h(x) \leftrightarrow x \)
  - Error-correcting codes \( m\hat{g} + z \leftrightarrow m \)
  - Multivariate quadratics \( y = MQ(x) \)
  - Supersingular isogenies \( \phi : e_1 \rightarrow e_2 \)
In 2016, NIST called for submissions for post-quantum signatures and post-quantum encryption schemes.

Side note: How do we standardize cryptography?

- Driven by Government only: Often NIST
  - Example: SHA-2 (currently in use), but also DualEC DRBG (backdoored by NSA)
- Driven by the Cryptographic Community
  - e.g., CAESAR competition
- Driven by independent Standardization Organizations
  - CFRG as a part of IETF
  - ISO working groups

Current PQC approach: NIST runs a public competition - a lot of academic research, but in the end NIST decides

- This is also how AES and SHA-3 were chosen
Timeline for NISTPQC

- Deadline: November 2017
- Will run for 3-5 years; in 2-3 rounds
- “Not-a-competition”
  - There will be more than one winner.
- December 2017: NIST announces that out of 81 submissions, 69 were “complete and proper”
  - 8 submissions from Radboud University
NIST PQC: Round 1


Slide credit: Daniel J. Bernstein (@hashbreaker)
NIST PQC Round 1: 22 Schemes Broken or Weakened


Slide credit: Daniel J. Bernstein (@hashbreaker)
On 30 January 2019 NIST announces the round 2 candidates
26 schemes are second round candidates
- 17 Key exchange/Encryption schemes
- 9 Signature schemes
Still 8 schemes from Radboud University
NIST PQC: Round 2 Candidates


Slide credit: Daniel J. Bernstein (@hashbreaker)
NIST PQC: Round 2 Mergers

- NTRUEncrypt + NTRU-HRSS-kEM → NTRU
- LEDAkem + LEDApkc → LEDAcrypt
- LAKE + LOCKER + Ouroboros-R → ROLLO
- Hila5 + Round2 → Round5
Key Encapsulation/Key Exchange


Signature Schemes

CRYSSTALS-DILITHIUM. FALCON. GeMSS. LUOV. MQDSS. Picnic. qTESLA. Rainbow. SPHINCS+.
15 Mar 2019: Submission deadline for Round 2 tweaks
Early 2020: Announcement of Round 3 candidates
We will go through 26 cryptographic schemes in great detail
  * ... of course not

I’ll talk about
  * **Hash-based Signatures** (and the corresponding submission SPHINCS+)
  * **Lattice-based Encryption** (with a focus on Kyber)

Focus:
  * Core concepts; omit as much details as possible
  * Applies to many more submissions than those two
  * But also the practical instantiation (sizes, run-time, comparison to what we are used to)
Hash-Based Signatures
What security does a “normal” signature provide ideally?

- **Authenticity:**
  Only you can produce your signature; no one else can forge it.

- **Integrity:**
  After signing, a change of the document is detected.

And in the real-world?

- nothing (?)

How can we have a signature on digital data?

- Scans of signatures are not a solution
- There are bad guys on the Internet; we want to have some security against them.
Digital Signatures

- Two parties: **Signer** and **Verifier**
- Two keys
  - **Secret Key/Private Key** sk: Used by the signer to sign documents
  - **Public Key** pk: Used by the verifier to verify signatures
- Some magic math that connects the secret key and public key
- Three algorithms
  - **Key Generation()** → sk, pk
  - **Sign(sk, m)** → σ
  - **Verify(pk, m, σ)** → {valid, invalid}
Post-quantum Signatures

- All current signatures (RSA, (EC)DSA) get broken by Shor
- We want a drop-in replacement that does exactly the same
- What if we could build a signature scheme from something that is not broken by quantum-computers?
  - Something that is symmetric
  - Something that we use anyway for signatures
  - A cryptographic hash-function
What is a Cryptographic Hash-function?

- There are many existing hash-functions
- Examples
  - MD5 (broken), SHA-1 (broken), SHA-2, SHA-3
- In general: \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \)
- E.g. for
  - SHA-1, \( n = 160 \)
  - SHA2-256, \( n = 256 \)
A cryptographic hash-function should satisfy three properties:

- **Collision resistance** (Kollisionsresistenz)
- **Preimage resistance** aka one-wayness (Urbildresistenz)
- **Second-preimage resistance** (2. Urbildresistenz)

**Collision attack:**
Find \( x_1 \neq x_2 \), s.t. \( h(x_1) = h(x_2) \)

**Preimage attack:**
Given \( y = h(x) \), find \( x' \), s.t. \( h(x') = y \)
(both \( x = x' \) and \( x \neq x' \) are ok)

**Second-preimage attack:**
Given \( x_1 \), find \( x_2 \neq x_1 \), s.t. \( h(x_2) = h(x_1) \)
(Quantum-)Attacks on Cryptographic Hash-functions

- Classical attack complexities (for a perfect hash-function):
  - Collision: $\mathcal{O}(\sqrt{2^n}) = \mathcal{O}(2^{n/2})$
  - Preimage: $\mathcal{O}(2^n)$

- Quantum attack complexities using Grover:
  - Collision: $\mathcal{O}(2^{n/2})$
  - Preimage: $\mathcal{O}(2^{n/2})$
1979: Lamport-Diffie One-Time Signature Scheme (LD-OTS)

- Start with a single-bit OTS signature scheme
- **KeyGeneration**
  - Secret key: $x_0, x_1 \in \{0, 1\}^n$
  - Public key: $y_0 = h(x_0); y_1 = h(x_1)$
- **Signing** message $m \in \{0, 1\}$
  - Signature $\sigma = x_m$
- **Verification** message $m \in \{0, 1\}$, signature $\sigma \in \{0, 1\}^n$
  - Check $y_m \overset{?}{=} h(\sigma)$
LD-OTS: Multi-bit Signature

- **Key Generation**
  - Secret key: \( x_0 \in \{0, 1\}^{n \times n}, x_1 \in \{0, 1\}^{n \times n} \)
  - Public key: \( y_{i,0} = h(x_{i,0}); y_{i,1} = h(x_{i,1}) \) for \( 0 \leq i < n \)

- **Signing message** \( m \in \{0, 1\}^n = m_0, m_1, \ldots, m_{n-1} \)
  - Signature \( \sigma = x_{m_0}, x_{m_1}, \ldots, x_{m_{n-1}} \)

- **Verification message** \( m \in \{0, 1\}^n \), signature \( \sigma \in \{0, 1\}^{n \times n} \)
  - Check \( y_{i,m_i} = h(\sigma_i) \) for \( 0 \leq i < n \)
LD-OTS

- **Private key**
  
  $x_{0,0} \quad x_{1,0} \quad x_{2,0} \quad \cdots \quad x_{N-3,0} \quad x_{N-2,0} \quad x_{N-1,0}$
  
  $x_{0,1} \quad x_{1,1} \quad x_{2,1} \quad \cdots \quad x_{N-3,1} \quad x_{N-2,1} \quad x_{N-1,1}$

- **Public key**
  
  $h(x_{0,0}) \quad h(x_{1,0}) \quad h(x_{2,0}) \quad \cdots \quad h(x_{N-3,0}) \quad h(x_{N-2,0}) \quad h(x_{N-1,0})$
  
  $h(x_{0,1}) \quad h(x_{1,1}) \quad h(x_{2,1}) \quad \cdots \quad h(x_{N-3,1}) \quad h(x_{N-2,1}) \quad h(x_{N-1,1})$

- **Signature on 100...110**
  
  $x_{0,1} \quad x_{1,0} \quad x_{2,0} \quad \cdots \quad x_{N-3,1} \quad x_{N-2,1} \quad x_{N-1,0}$

- **Verification**: hash, compare to public key

- Can still only do this **once**!
Example: $n = 256$ (e.g., SHA3-256)

- Public Key/Private Key: 2 KiB
- Signature: 1 KiB
LD-OTS is one-time, we want a many-time signature scheme

Idea: Just use many OTS, say $2^{20}$ and use them one-by-one

- Public Key: 2 GiB
- Secret Key: 2 GiB (or PRNG: 32 bytes)
- Signature: 1 KiB + 3 bytes index
Can we make the public key smaller?

Merkle: Organize OTS public keys as a binary hash tree

- aka Merkle tree

MSS public key: root of hash tree

MSS signature

- OTS signature
- OTS public key
- Nodes in tree to recover root (aka authentication path)
1979: Merkle Signature Scheme

\[ \text{pk} \]

\[ p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7 \quad p_8 \]
1979: Merkle Signature Scheme

Diagram: A Merkle tree with the root node labeled \( pk \) and the leaves labeled \( p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8 \).
1979: Merkle Signature Scheme

- Secret key
  - store a n bit seed
  - $x_m^{(i,j)} = h(\text{seed}|i|j|m)$
- Signing $2^{20}$ messages (SHA3-256)
- Public key: 32 bytes
- Secret key: 32 bytes + index of next key to use
- Signature
  - 1 KiB OTS
  - 1 KiB OTS public key (Why not 2 KiB?)
  - Authentication path: $19 \cdot 32 = 608$ bytes
  - Index (Why?)
- ca. 2.6 KiB
Last year XMSS was standardized

Internet Research Task Force (IRTF)
Request for Comments: 8391
Category: Informational
ISSN: 2070-1721

A. Huelsing
TU Eindhoven
D. Butin
TU Darmstadt
S. Gagd
Genua GmbH
J. Rijneveld
Radboud University
A. Mohaisen
University of Central Florida
May 2018

XMSS: eXtended Merkle Signature Scheme

Abstract

This note describes the eXtended Merkle Signature Scheme (XMSS), a hash-based digital signature system that is based on existing descriptions in scientific literature. This note specifies Winternitz One-Time Signature Plus (WOTS+), a one-time signature scheme; XMSS, a single-tree scheme; and XMSS^MT, a multi-tree variant of XMSS. Both XMSS and XMSS^MT use WOTS+ as a main building block. XMSS provides cryptographic digital signatures without relying on the conjectured hardness of mathematical problems. Instead, it is proven that it only relies on the properties of cryptographic hash functions. XMSS provides strong security guarantees and is even secure when the collision resistance of the underlying hash function is broken.
2018: XMSS

- SHA-2 or SHA-3
- 256-bit or 512-bit
- Uses Winternitz One-time signatures (W-OTS)
- Tree heights: 10, 16, 20
- Protects against multi-target attacks
- Also defines a multi-tree variant called XMSS\textsuperscript{MT}
  - With total tree heights: 20, 40, 60
W-OTS

- Idea: sign groups of $\log(w)$ bits (let $w = 2^n$)
- Trade time for signature and key size
- Example: $w = 4$, let’s sign 10 00 11 01 01

public key:

private key: $s_0$, $s_1$, $s_2$, $s_3$, $s_4$

- Can still only do this once!

Note: ‘checksum chains’ to prevent forgery omitted for simplicity
For large $h$, key generation becomes too expensive

Instead: split into several layers of tree (aka hyper-tree)
  
  - Top layers sign roots of lower layers
  - KeyGen: only top layer
  - Slower signing and verification; larger signatures
XMSS performance \((n = 256, h = 10)\)
- Signature: 2500 bytes
- Public key: 64 bytes
- Key generation: 1 238 016 hashes (ca. 1.5 sec*)
- Signing: 5 725 hashes (ca. 2ms*)
- Verification: 1 149 hashes (<1ms*)

XMSS\(^{MT}\) performance \((n = 256, h = 60, d = 6)\)
- Signature: 14 824 bytes
- Key generation: 7 428 096 hashes
- Signing: 7 227 hashes
- Verification: 6 894 hashes

* SHA2-256 on modern desktop processor
Statefulness sucks (for many use-cases)

- XMSS/XMSS\textsuperscript{MT} is **stateful**
- You need to keep track of the leaves you already used
- If you re-use a key your security vanishes
- This is ok for some use-cases (e.g., code signing)
- But not possible / extremely painful for others
  - Key backup
  - Multiple servers using same secret key
  - ...
- Wouldn’t it be great to have HBS without a state?
Special note to law-enforcement agents:

“The word ‘state’ is a technical term in cryptography. [...] We are not talking about eliminating other types of states. We love most states, especially yours! Also, ‘hash’ is another technical term and has nothing to do with cannabis.”

— https://sphincs.cr.yp.to
2015: SPHINCS

- Seriously big hyper-tree \((\approx 2^{64} \text{ leafs})\)
  - Allows random leaf selection
  - Stateless!
- Use few-time signature on lowest layer
- Signatures larger and slower
  - 8 KiB – 40 KiB, \(\approx 100\) ms
- SPHINCS+ was submitted to NIST
Lattice-Based Encryption
Public-Key Encryption

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$sk, pk \leftarrow \text{KeyGen}()$</td>
<td></td>
</tr>
<tr>
<td>$c \leftarrow \text{Encrypt}_{pk}(m)$</td>
<td>$m \leftarrow \text{Decrypt}_{sk}(c)$</td>
</tr>
</tbody>
</table>

- Three algorithms
  - KeyGen generating a public/secret key pair
  - Encrypt
  - Decrypt
- Messages aka plaintext $m$
- Ciphertext $c$
In practice: encrypt a small key; use in a block cipher (AES)
This is often called **session key** or **shared secret**
Aka **hybrid encryption**
Asymmetric part: **key encapsulation mechanism (KEM)**
Symmetric part: **data encapsulation mechanism (DEM)**
If both parties contribute to the session key: **key exchange (KEX or KE)**
We work in the commutative finite group $\mathbb{Z}_q = \{1, 2, ..., q - 1\}$

The group has a generator $g$, i.e., $\mathbb{Z}_q = \{g^0, g^1, ... g^{q-2}\}$

**Discrete Logarithm Problem (DLP)**

- Given generator $g$, prime modulus $q$, and $y = g^x \mod q$
  it is hard to find $x$

In real life: Don’t use numbers in $\mathbb{Z}_q$, but points on an elliptic curve → **ECDLP, ECDH**

- Small keys of 32 bytes
- Very fast

Both broken by Shor on a quantum computer
### Pre-Quantum: DH KEX

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \leftarrow g^a \mod q$</td>
<td>$B \leftarrow g^b \mod q$</td>
</tr>
<tr>
<td>$ss \leftarrow B^a = g^{a\cdot b} \mod q$</td>
<td>$ss \leftarrow A^b = g^{a\cdot b} \mod q$</td>
</tr>
</tbody>
</table>

- We want this, but post-quantum
Many PQ KEMs are based upon lattice problems
Lattice KEMs in the second round of NIST: 7/17
What is a lattice (Gitter)?

\[ \Lambda = \left\{ \sum_{i=1}^{n} a_i v_i \mid a_i \in \mathbb{Z} \right\} \subset \mathbb{Z}^n \text{ with basis } \{v_1, \ldots, v_n\} \in \mathbb{Z}^n \]
Lattices
There are lattice problems that are believed to be hard
E.g., finding the shortest vector in an high-dimensional lattice is NP-hard – **Shortest Vector Problem (SVP)**
Modern lattice-based cryptography is either based on **LWE** or **NTRU**
- It can be proved that LWE is at least as hard as some lattice-problems that are believed to be hard
- i.e., LWE is believed to be hard
Learning-with-errors (LWE)

- We work in the finite field $\mathbb{Z}_q$ (Endlicher Körper), prime $q$
- Let $\chi$ be a probability distribution over $\mathbb{Z}_q$ with favours “small” values
  - $x \leftarrow \chi$: draw $x$ randomly according to distribution $\chi$
  - In practice: Discrete Gaussian distribution or centered binomial distribution

Learning-with-errors (LWE)

- $A \in \mathbb{Z}_q^{n \times n}$ uniformly random (publicly known)
- secret $s \in \mathbb{Z}_q^n \leftarrow \chi$, error $e \in \mathbb{Z}_q^n \leftarrow \chi$ (secret)
- Given $t \leftarrow A \cdot s + e \mod q$, it is hard to find $s$
- Note: Given $A \cdot s$, it is easy to find $s$!
**LWE Key Exchange**

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s, e \leftarrow \chi$</td>
<td>$s', e' \leftarrow \chi$</td>
</tr>
<tr>
<td>$b \leftarrow as + e$</td>
<td>$b$</td>
</tr>
<tr>
<td>$u \leftarrow b$</td>
<td>$u$</td>
</tr>
</tbody>
</table>

Alice has $v = us = ass' + e's$

Bob has $v' = bs' = ass' + es'$

- Secret and noise $s, s', e, e'$ are small
- $v$ and $v'$ are approximately the same
Matrix-multiplication is expensive; A is huge
Ring-LWE: Use a polynomial ring instead of $\mathbb{Z}_q$
Let the polynomial ring be $\mathcal{R}_q = \mathbb{Z}_q[X]/(f(X))$
- Coefficients are in the field $\mathbb{Z}_q$ (all arithmetic mod $q$)
- E.g., $f(X) = X^n + 1$, i.e., polynomials have $n$ coefficients
- After every operation, we reduce modulo $f(X) \rightarrow X^n \equiv -1 \pmod{X^n + 1}$
A, s, e all become polynomials in $\mathcal{R}_q$

- Structured lattice
- Everything is smaller
- Polynomial arithmetic is faster than matrix arithmetic
- Attacks on RLWE are currently not better than on LWE (with same $n$)

All lattice-based KEMs in the NIST competition use a ring; except for Frodo
Kyber uses a hybrid between LWE and R-LWE, which is called **Module-LWE (MLWE)**

- **A** is a $k \times k$ matrix of polynomials in $\mathcal{R}_q$
- **s** and **e** are $k$-dimensional vectors of polynomials in $\mathcal{R}_q$

**Benefits**

- We can use a rather small, fixed $n$ (Everything is faster)
- Some future attacks on RLWE may not be as efficient on MLWE (?)
Kyber

- [https://pq-crystals.org/kyber](https://pq-crystals.org/kyber)
- $q = 2^{13} - 2^9 + 1 = 7681$ (soon $q = 3329$)
- $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^{256} + 1)$
  - Allows fast polynomial arithmetic using the Number Theroretic Transform (NTT)
  - NTT: Fast Fourier Transforms (FFT) in a finite field
  - $a \cdot b = \text{NTT}^{-1}(\text{NTT}(a) \circ \text{NTT}(b))$
  - $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n \cdot \log n)$
- $k = 3$ (recommended), $k = 2$ (light), or $k = 4$ (paranoid)
- $\chi$ is the centered binomial distribution with coefficients in $\{-4, -3, \ldots, +3, +4\}$ (recommended)
Side-note: Kyber builds a INDCCA-secure KEM from an INDCPA encryption scheme.

- This talk: CPA variant
- I’m also skipping a lot of other technical details
Kyber - Key Generation

\textbf{KeyGen()}

\begin{itemize}
  \item $A \leftarrow \mathcal{R}_{q}^{k \times k}$
  \item $s, e \leftarrow \chi$ \hspace{1cm} ($\in \mathcal{R}_{q}^{k}$)
  \item $t \leftarrow As + e$
  \item Public key $pk$: $A, t$
  \item Secret key $sk$: $s$
\end{itemize}
Encrypt($A, t, m$)

- Message $m$ is encoded as polynomial $\in \mathcal{R}_q$
- $r, e_1 \leftarrow \chi \quad (\in \mathcal{R}_q^k)$
- $e_2 \leftarrow \chi \quad (\in \mathcal{R}_q)$
- $u \leftarrow A^T \cdot r + e_1$
- $v \leftarrow (t^T \circ r) + e_2 + m$
- Ciphertext: $(u, v)$
Decryption(s, u, v)

- $m' \leftarrow v - (s^T \circ u)$
- $m \leftarrow \lfloor m' \rfloor$

- $m' \approx m$
- The encoding of $m$ is important
- We want to encode 256-bits into 256-coefficients
  - One bit per coefficient
  - Maximize distance
  - i.e., set them to 0 and $q/2$
Kyber - Performance (CCA-secure variant)

Sizes
- Public key: 1,088 bytes (vs. 32 bytes in ECC)
- Ciphertext: 1,152 bytes (vs. 32 bytes in ECC)
- **One of the smallest PQC KEM**
- **Huge compared to ECC**

Speed (Haswell, AVX2)
- KeyGen: 85,472 cycles
- Encapsulation: 112,660 cycles
- Decapsulation: 108,904 cycles
- **Extremely fast**
- **Often faster than ECC**
Some other considerations for lattice-based KEMs

- LWE (Frodo) vs. RLWE (NewHope) vs. MLWE (Kyber)
- (M)LWE (Kyber) vs. NTRU (NTRU)
- Prime q (Kyber) vs. power-of-two q (Saber)
  - Power-of-two: modular reductions for free; but no NTT
- Also prime q (NTRUPrime) vs. power-of-two q (NTRU)
- ...
Huge design space; more complicated than RSA, DH
Sizes are huge compared to ECC; prepare to handle this
Speed is not a problem; compensate for some gain in size
Structured lattices are the very promising PQC candidate
  You can also construct signatures from this: Dilithium, qTesla, Falcon
If you break (R)LWE you break half of the NIST schemes and we are screwed
  But same is true for RSA, DLP (even with classical computers)
Slides & publications is available at kannwischer.eu

Contact: matthias@kannwischer.eu

Thank you!