Post-Quantum Cryptography

Prof. Dr. J. Buchmann

Written by: P. Schmidt

Last updated: February 25, 2010

Contents

1. Introduction **5**

2. Motivation **5**

3. Quantum Algorithms and Quantum Computers **6**
3.1. The Problem of Factoring 6
3.2. Quantum Computers 7
3.3. Shor's Algorithm 8
3.3.1. Correctness of the algorithm 8
3.4. Quantum Part of Shor's Algorithm 9

I. Hash-based Cryptography **11**

4. Lamport-Diffie One-Time Signature **11**
4.1. Setup of the LD-OTS-Scheme 11
4.2. Security of the LD-OTS-Scheme 12

5. Merkle Signature Scheme (MSS) **13**
5.1. Idea 13
5.2. Setup 13
5.3. Instantiation 15
5.4. Security 15
5.5. Merkle Signature Scheme - practicable? 16
5.6. Winternitz OTS – an improvement of MSS 17
5.7. Secret key problem: Coronado's idea 18
5.8. Key generation problem: Tree Chaining 18
5.9. Authentication path generation problem 20

II. Lattice-based Cryptography **22**

6. Lattices **22**
6.1. Definitions and notations 22
6.2. Lattice Problems 23

7. Lattice (Basis) Reduction **26**

8. Lattices in a post-quantum world **29**

9. How to find hard instances: the idea of Ajtai **30**
9.1. Practical questions about Ajtai's construction 31

10. Efficiency Improvements using Ideal Lattices **32**
10.1. Definition of Ideal Lattices 33

11. Lattice-based signatures **33**
11.1. First instantiation of Goldreich, Golwasser and Halevi (GGH) 34
11.2. Cryptanalysis of GGH by Nguyen and Regev 34
11.3. Second instantiation of Lyubashevsky and Micciancio (LM-OTS) 36
11.4. The security of the LM-OTS 36

III. Code-based Cryptography **38**

12. Motivation **38**

13. Definitions **38**

14. Application in Cryptography **40**
14.1. McEliece 40
14.2. Niederreiter 41
14.2.1. Goppa codes 42

15. Security Hypothesis **42**

16. Code-based Signatures **42**
16.1. Signatures with Codes 43
16.2. The CFS Signature Scheme 43
16.3. Stern Identification Scheme 44

17. Identity-based Identification and Signature Scheme using Error-Correcting Codes **45**

18. Summary **48**

IV. Multivariate Cryptography **50**

19. Introduction **50**

20. Notation and generalized scheme **50**
20.1. General notation 50
20.2. Generalized encryption and signature scheme 51

21. How to create $\mathbb{F}_{2^n}$ – the finite field with 2^n elements **51**

22. Matsumoto-Imai (MI) **51**
22.1. Construction of MI 52

22.2. The quadratic map Q 52
22.3. Inverting Q — difficult? 52
22.4. Perturbed MI 53

23. Oil and Vinegar Schemes and Rainbow **54**
23.1. General Oil and Vinegar Scheme 54
23.2. Cryptanalysis of Balanced Oil and Vinegar Scheme 54
23.3. Unbalanced Oil and Vinegar Scheme 55
23.4. Rainbow 55
23.5. "Parameters" of Rainbow 56

1. Introduction

Literature:

- D. Bernstein, J. Buchmann, E. Dahmen (Eds.): Post-Quantum Cryptography, Springer 2009.
- J. Buchmann: Einführung in die Kryptographie, Vierte, erweiterte Auflage, Springer 2008.

Announcements:
There will be a "problem section" on Friday from 9:50am to 11:30am in A213. We publish the problems on our website. More information can be found there.

Content of the lecture *Post-Quantum Cryptography*:

1. Motivation.
2. Power of quantum computers, four approaches to design post-quantum cryptography.
3. Hash-, lattice- and code-based and multivariate cryptography.

2. Motivation

For a first motivation one can consider the question: Why do we do this lecture?
The first thing we want to do is to give some examples where public-key cryptography is in practice *today*. By discussion we considered three situations: The first is about the certificates which the RBG offers you to sign emails etc. The second one is online-banking in which we want to make ourselves sure of the confidentiality of transactions from Germany with our bank on the Cayman Islands. And the last one is the sort of software updates of antivirus programs or Windows.
Especially the (first and) third one as a natural application of digital signatures with a high amount of users (the number of users may hereby be greater than 10^6) shows the necessity of public-key cryptography: The manufacturer of the antivirus program authenticates its updates which can be downloaded by the users. After the download the users verify the authentication of the updates and install them. But the manufacturer has to make sure that no user itself is able to produce authenticity proofs for some update he created, as can be seen in Figure 1.
One can think here e.g. of the "Bundestrojaner" or "Federal Trojan Horse" where the first plan was to use re-configured antivirus updates to infect a suspect's computer for investigation.
As a conclusion there are two main things to be mentioned:

1. *We really need public-key cryptography* since we use it in the internet (and elsewhere) every day.

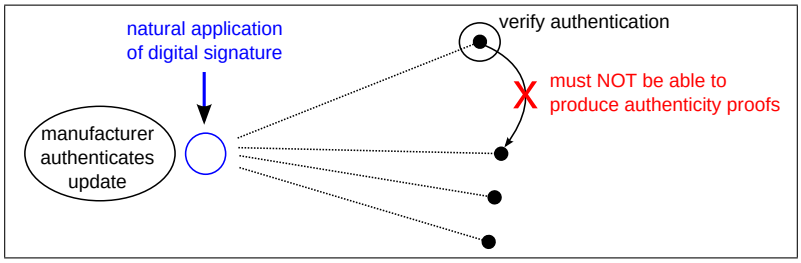

Figure 1: Authentication of updates

2. The public-key cryptosystems that are currently in use can be made insecure by quantum computers (as Shor showed in 1994).

3. Quantum Algorithms and Quantum Computers

The next thing we want to introduce are quantum algorithms and computers, and discuss why they weaken or degrade current cryptosystems.

3.1. The Problem of Factoring

For the factoring problem consider the following: Given a composite natural number $n \in \mathbb{N}$. The goal now is to find a proper divisor (which is a positive integer) in polynomial time.
By solving the factoring problem, RSA can be broken.

Idea: Fix some integer $x \in \mathbb{Z}$ with $\gcd(x, n) = 1$ (which means that x is coprime to n, gcd being the *greatest common divisor*), and consider the map

$$f : \mathbb{Z} \to \mathbb{Z}_n := \{0, \dots, n-1\}, \quad a \mapsto x^a \mod n.$$

We know that this function is periodic.
Example: Let $n = 4$ and $x = 3$, then

a	0	1	2	3	4	5
$f(a)$	1	3	1	3	1	3

And now the question is: What is the period of that function f?
The **period** is the order r of the residue class $x + n\mathbb{Z}$ in $(\mathbb{Z}/n\mathbb{Z})^*$ (residue class of invertible elements) being the smallest exponent $a > 0$ such that $x^a \equiv 1 \mod n$.

Theorem: It is $x^{\varphi(n)} \equiv 1 \mod n$ where

$$\varphi(n) := |(\mathbb{Z}/n\mathbb{Z})^*| = |\{y : 0 < y \leq n-1, \gcd(y, n) = 1\}|.$$

Statement: The order divides $\varphi(n)$.

Example: Let $n = 15$, then $\varphi(n) = 8 = 2 \cdot 4$ because $\varphi(pq) = (p-1)(q-1)$.

Now the main assertion is: Shor's algorithm finds the order r.
In the case of RSA with $n = pq$ we have that $|(\mathbb{Z}/n\mathbb{Z})^*| = (p-1)(q-1)$ is even and at least half of the orders are even.

Claim: If r is even, we can factor.
This is clear because by the "Third Binomial Formula" we obtain

$$(x^{\frac{r}{2}} + 1)(x^{\frac{r}{2}} - 1) = x^r - 1 \equiv 0 \mod n \implies n|(x^{\frac{r}{2}} + 1)(x^{\frac{r}{2}} - 1).$$

Since r is the order of x, we have $x^{\frac{r}{2}} - 1 \not\equiv 0 \mod n$ and $\gcd(x^{\frac{r}{2}} - 1, n)$ is "hopefully" a proper divisor of n.

What we see is that our goal to find the order of elements can be translated into the problem to find periods of a periodic function what quantum computers are able to.
For this we have to explain what quantum computers in this context are and how they work.

3.2. Quantum Computers

In classical physics we have classical bits being either 0 or 1. In contrast to this we have qubits in quantum mechanics. Qubits in this context are linear combinations of the two basis elements $|0\rangle$ and $|1\rangle$ (which can be thought of as $(0,1)^T$ and $(1,0)^T$ in $\mathbb{C}^2$) and in this way called to be in "superpositions" of them. We can write qubits like

$$|x\rangle = w_0|0\rangle + w_1|1\rangle \quad \text{with} \qquad w_0, w_1 \in \mathbb{C} \quad \text{and} \quad |w_0|^2 + |w_1|^2 = 1,$$

where $w_0 = \mathrm{Re}(w_0) + i \cdot \mathrm{Im}(w_0)$ and $|w_0|^2 = \mathrm{Re}(w_0)^2 + \mathrm{Im}(w_0)^2$. Think of $|w_0|^2$ and $|w_1|^2$ being the probabilities of $|x\rangle$ being $|0\rangle$ or $|1\rangle$ after measurement.
Now consider quantum registers made of many qubits, e.g. a register of m qubits. The state of the register x can be written as

$$|x\rangle = \sum_{k=0}^{q-1} w_k |x_k\rangle,$$

where $|x_k\rangle$ is one of the base states (for a register with m qubits) and we have $q = 2^m$ base states.

Example: Let $m = 2$, then one base state would be $\boxed{|0\rangle}\boxed{|1\rangle}$.

Example: We want to evaluate the function $f(x) := 2x \mod 7$. Therefore we prepare two 3-qubit quantum registers and initialize $|x, 0\rangle$ with a superposition of $|0,0\rangle, \ldots, |7,0\rangle$ in uniform distribution

$$|x, 0\rangle = \frac{1}{\sqrt{8}} \sum_{k=0}^{7} |k, 0\rangle.$$

Then we apply the given function to the quantum registers and obtain

x	0	1	2	3	4	5	6	7
$f(x)$	0	2	4	6	1	3	5	0

The result will be $|x, f(x)\rangle$ with

$$|f(x)\rangle = \frac{1}{\sqrt{8}} \sum_{k=1}^{6} |k\rangle + \frac{1}{\sqrt{4}} |0\rangle,$$

which shows that measurement is not very helpful in this case.

3.3. Shor's Algorithm

The first version of the paper in which Shor published the announced algorithm was 1994, the last in 1996.
For an input of a composite $n \in \mathbb{N}$ the algorithm does the following steps:

1. Pick $x \in \{2, \ldots, n-1\}$ uniformly at random (written as $x \xleftarrow{\$} \{2, \ldots, n-1\}$ or $x \in_R \{2, \ldots, n-1\}$).
2. If $\gcd(x, n) \neq 1$: Return $\gcd(x, n)$.
3. Find period r of $f(a) = x^a \mod n$.
   (This is the quantum part of the algorithm.)
4. If r is odd or $x^{r/2} \equiv -1 \mod n$: Goto 1.
5. Return $\gcd(x^{r/2} \pm 1, n)$.

It can be shown that step 5. from step 4. is reached with probability at least $\frac{1}{2}$ in the RSA case.

3.3.1. Correctness of the algorithm

We want to discuss why Shor's algorithm is correct.
Let us first see what we know:

1. r is even, and
2. $x^{r/2} \not\equiv -1 \mod n$.

This leads to

$$0 \equiv x^r - 1 \overset{1.}{\equiv} \left(x^{r/2} + 1\right)\left(x^{r/2} - 1\right) \mod n,$$

where $x^{r/2} + 1 \not\equiv 0$ by 2. and $x^{r/2} - 1 \not\equiv 0$ since r is the order of x. This means that the elements $x^{r/2} + 1$ and $x^{r/2} - 1$ are zero divisors in $\mathbb{Z}_n$.

Now let p be a prime divisor of n and consider

$$0 \equiv \left(x^{r/2}+1\right)\left(x^{r/2}-1\right) \mod p.$$

Since $\mathbb{Z}_p$ is a field with no zero divisors, one of the factors has to be zero.
(*Quiz*: Which values can $\gcd(a,n)$ take for $n=pq$ with primes p,q and $a \in \mathbb{N}$?
The greatest common divisor of a and n is $\gcd(a,n) \in \{1,p,q,n\}$.)
If we now assume that the first factor is zero, we obtain $\gcd(x^{r/2}+1,n)=p$. This shows that Shor's algorithm really yields a proper factor of n.

3.4. Quantum Part of Shor's Algorithm

Remember that we are at the beginning of step 3. in the classical part at which we have x and n as input. What we now do, is:

1. We set $q=2^k$ such that $n^2 \leq q < 2n^2$.
2. We initialize the quantum registers to be in state
$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |x\rangle\,|0\rangle\,.$$
3. Computing $f(x)=x^a \mod n$ and storing in the second register yields
$$\frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |x\rangle\,|x^a \mod n\rangle\,.$$
4. Now we do Fourier transformation
$$|a\rangle \mapsto \frac{1}{\sqrt{q}} \sum_{c=0}^{q-1} \omega^{ac}\,|c\rangle$$
on the first register using the q-th root of unity $\omega := \exp(2\pi i/q)$ and obtain
$$\frac{1}{q} \sum_{a=0}^{q-1}\sum_{c=0}^{q-1} \omega^{ac}\,|c\rangle\,|x^a \mod n\rangle\,.$$
5. Observe both registers.
6. Try to compute r from $\frac{c}{q}$ using continued fractions.
(This is classical again.)

Now the question is how often we need to run these six steps.
The probability of observing $|c\rangle \left|x^k \mod n\right\rangle$ is

$$\Pr\left[\text{observing } |c\rangle \left|x^k \mod n\right\rangle\right] = \left|\frac{1}{q} \sum_{a:x^k \equiv x^a \mod n} \omega^{ac}\right|^2.$$

Some analysis yields that

$$\Pr\left[\begin{array}{ll} & \text{observing } |c\rangle \left|x^k \mod n\right\rangle \text{ and} \\ (1) & \exists d \in \mathbb{Z} : \left|\frac{c}{q} - \frac{d}{r}\right| \leq \frac{1}{2q} \end{array}\right] \geq \frac{1}{3r^2}.$$

Under condition (1) we have:

- Since $q \geq n^2$, one can show that there is at most one fraction $\frac{d}{r}$ with $r < n$. This can be found efficiently from known $\frac{c}{q}$ by using continued fractions.
- If d and r are coprime ($\gcd(d, r) = 1$), r is the denominator of the fraction. So in this case using continued fractions directly yields r, otherwise it does not!

How many favorable states are there? This questions is interesting since we want to count the number of states for which we successfully compute r.

- There are $\varphi(r)$ working d where φ denotes Euler's totient function such that each fraction $\frac{d}{r}$ is close to one $\frac{c}{q}$. *(1^{st} register)*
- There are also r distinct values of x^k. *(2^{nd} register)*

Using both remarks get us to know the

$$\text{chance of success} = \frac{r\varphi(r)}{3r^2} > \frac{\text{const.}}{\log\log r}.$$

So we need $\mathcal{O}(\log\log r)$ trails to find r with high probability. This can be improved to $\mathcal{O}(1)$ (see P. Shor: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer).

Part I.
Hash-based Cryptography

4. Lamport-Diffie One-Time Signature

Recall the problem in which we discussed the need of digital signatures: The example was an antivirus company that published its product together with a public key pk_{company} and a real update with a good signature which all users of this antivirus program can verify using $ver(pk_{\text{company}}$,update,signature). By this they (the users) can assure themselves of authenticity and integrity of the update. We want to assure that no evil hacker has the chance to commit a fake update with a fake signature that the users would believe to be good.
The arising question is: How can we achieve this?
Lamport and Diffie in 1979 published a one-time signature scheme that will ensure this, if only one update / signature is needed.

4.1. Setup of the LD-OTS-Scheme

Let $n > 0$ and $f : \{0,1\}^n \to \{0,1\}^n$ be a one-way function.
KeyGen: Choose $X \xleftarrow{\$} (\{0,1\}^n)^{2\times n}$ as secret key, e.g.

$$X = \begin{pmatrix} 001 & 100 & 000 \\ 110 & 110 & 101 \end{pmatrix},$$

and for the public key $Y \in (\{0,1\}^n)^{2\times n}$ compute $Y_{ij} = f(X_{ij})$, e.g.

$$Y = \begin{pmatrix} f(001) & f(100) & f(000) \\ f(110) & f(110) & f(101) \end{pmatrix}.$$

Sign: For a message $m \in \{0,1\}^n$ we obtain the signature s by

$$s = (X_{m_0,0}, X_{m_1,1}, \ldots, X_{m_{n-1},n-1}),$$

e.g. for $m = (010)$ we get $s = (001, 110, 000)$.

Verification: For a given message $m \in \{0,1\}^n$ and signature $s = (s_0, \ldots, s_{n-1})$ we have to check

$$f(s_i) \stackrel{?}{=} Y_{m_i,i} \qquad \text{for all } 0 \le i < n.$$

The key size in the Lamport-Diffie One-Time Signature Scheme is for public and secret key $2n^2$ bit and for the signature n^2 bit, e.g. for $n = 256$ for pk, sk we have 16KB and for the signature 8KB.

4.2. Security of the LD-OTS-Scheme

We want to discuss the security of the Lamport-Diffie One-Time Signature Scheme. For this we have to define what we mean by "security" by creating the model of *existentially unforgeable under adaptive chosen message attacks*, or EU-CMA in short.

Definition (EU-CMA): A signature scheme $\mathsf{DS} = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify})$ is EU-CMA if for any efficient algorithm $\mathcal{F}$ the probability that the experiment $\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{F},\mathsf{DS}}$ evaluates to 1 is negligible, where $\mathsf{negligible}$ means that the probability $p(n)$ vanishes faster than the inverse of any polynomial, or more formally:

$$\forall c \exists n_0 \forall n \geq n_0 : p(n) < \frac{1}{n^c}$$

Experiment $\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{F},\mathsf{DS}}(n)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{KeyGen}(n)$
$(m^*, \sigma^*) \leftarrow \mathcal{F}^{\textsc{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$
let σ_i be the answer returned by $\textsc{Sign}(\mathsf{sk}, \cdot)$ on input m_i, for $i = 1, \ldots, k$.
Return 1 iff $\textsc{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$ and $m^* \notin \{m_1, \ldots, m_k\}$.

For one-time signatures we assume $k = 1$.

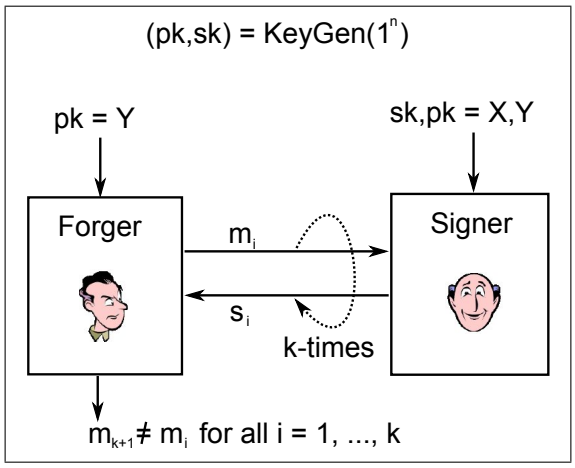

Figure 2: EU-CMA

Given a signing oracle S, a forger may

- see the public key pk,
- choose some message m of his opinion,
- get the signature $s = S(m)$ for the message,

and has to produce a message $m' \neq m$ which verifies correctly. Let the messages differ in the i^{th} bit, then the attacker must have inverted f because the attacker – of course – knows $x_{m_i,i} = s_i$ but does not know $x_{m'_i,i}$ because it is unknown.

Why the scheme is one-time can be seen above.
We have seen that the LD-OTS has minimal security requirements. Since its security is based on the security of a one-way function, there is no number theory which could be broken. But the LD-OTS has a big disadvantage: It can only be used for one signature with a given key pair. But we need the possibility to sign more than one document or message, e.g. an electronic banking server would need many key pairs. We will show now how the Merkle Signature Scheme (MSS) fixes this disadvantage.

5. Merkle Signature Scheme (MSS)

5.1. Idea

The idea of the MSS reduces the validity of many verification keys to the validity of one public key using a binary hash-tree construction. The root of the tree is the public key of the MSS. The leafs of the tree correspond to the (hash of the) OTS verification keys.

5.2. Setup

Let $\mathcal{G}(n) = \{g : \{0,1\}^* \to \{0,1\}^n | k \in K\}$ be a family of cryptographic hash-functions. For MSS a security parameter n and an OTS with OTS-triple

$$(\mathsf{KeyGen}_{\mathrm{OTS}}, \mathsf{Sign}_{\mathrm{OTS}}, \mathsf{Verify}_{\mathrm{OTS}})$$

are selected. Also, a positive integer H is chosen. MSS will then be able to sign 2^H messages $m \in \{0,1\}^*$ with one public key since it constructs a binary hash-tree of height H.

KeyGen: The MSS key generation works as follows: First, 2^H OTS key pairs (X_i, Y_i) with $0 \leq i < 2^H$ are generated by using $\mathsf{KeyGen}_{\mathrm{OTS}}$. (Here X_i is the OTS secret key and Y_i is the OTS public key.) Then one has to choose a hash-function $g \xleftarrow{\$} \mathcal{G}(n)$ uniformly at random. Next the Merkle tree which is a binary hash-tree of height H is generated (similar to Figure 3).

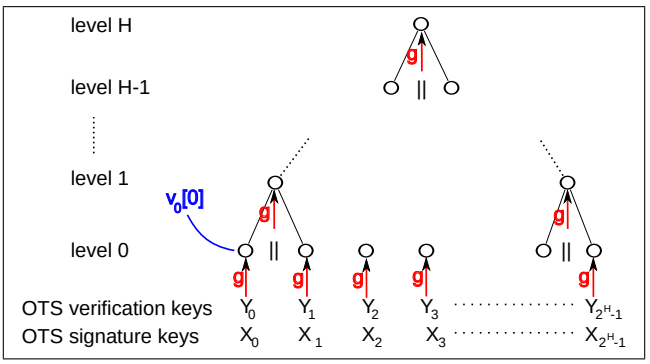

Figure 3: Construction of the Merkle hash-tree

The nodes on height i are denoted by $v_i[j]$ where $0 \leq j < 2^{H-i}$.
We start by constructing the leafs of the Merkle tree. They are the hashes of the OTS verification keys, i.e. $v_0[j] = g(Y_j)$, $0 \leq j < 2^H$. The inner nodes of the Merkle tree are constructed according to the following construction rule: A parent node is the hash of the concatenation of its left and right child, which means $v_i[j] = g(v_{i-1}[2j]||v_{i-1}[2j+1])$ for $0 \leq i \leq H$ and $0 \leq j < 2^i$, where $||$ denotes concatenation.
At the end a counter C, which will be incremented by every signature operation and thus in which the number of the last used OTS signing key will stored, is initialized via $C \leftarrow -1$. Also a status field S is created which contains the counter (and maybe other additional information): $S = (C)$.
The MSS public key pk is the root $v_H[0]$ of the Merkle tree and the hash-function g, i.e. $pk = (v_H[0], g)$.
The MSS secret key sk consists of the sequence $(X_0, \ldots, X_{2^H-1})$ of OTS secret keys and the status field S, i.e. $sk = ((X_0, \ldots, X_{2^H-1}), S)$.

Sign: Let $d \in \{0,1\}^*$ be the document or message to be signed.
At first the signer increases the counter $C \leftarrow C + 1$ and refreshes the status field S.
Then he generates the one-time signature σ_{OTS} of d using the OTS signing key X_C in the OTS signing function $\mathsf{Sign}_{\text{OTS}}$. Since the OTS verification key Y_C for the potential verifier is not known to be valid, the signer constructs the so-called *authentication path* that allows a verifier to reduce the validity of the OTS verification key Y_C to the validity of the (first part of the) public key pk.
To be more precise, the authentication path allows the construction of a path from the leaf $g(Y_C) = v_0[C]$ to the root $v_H[0]$ of the tree. The authentication path is a sequence $(a_0, \ldots, a_{H-1})$ of nodes in the Merkle tree. The node a_h, $0 \leq h < H$, is the sibling of the node at height h on the path from leaf $g(Y_C)$ to the root of the Merkle tree. In detail, we use

$$l_h = \left\lfloor \frac{C}{2^h} \right\rfloor \tag{1}$$

and have

$$a_h = \begin{cases} v_h[l-1] & \text{for } l_h \equiv 1 \mod 2 \\ v_h[l+1] & \text{for } l_h \equiv 0 \mod 2 \end{cases}$$

Example: Consider the situation as in Figure 4.
The nodes with a red cross form the path from leaf $g(Y_3)$ to the root of the Merkle tree. The authentication path of this leaf is

$$(a_0, a_1, a_2) = (v_0[2], v_1[0], v_2[1]).$$

Verification of the correctness: For $h = 2$ we have $l_2 = \lfloor \frac{3}{4} \rfloor \equiv_2 0$ and thus $a_2 = v_2[0+1]$.

In general, the whole signature of MSS then is

$$\sigma = (C, \sigma_{\text{OTS}}, Y_C, (a_0, \ldots, a_{H-1})). \tag{2}$$

Verification: The signature verification works as follows: The verifier obtains the document or message d, the public key pk and the signature σ as in (2). The verifier uses

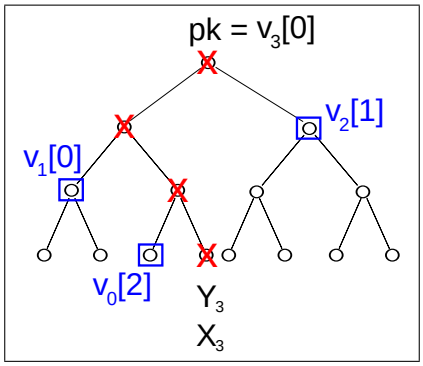

Figure 4: Example of a binary MSS hash-tree

the verification key Y_C in the verification function $\mathsf{Verify}_{\mathrm{OTS}}$ to verify the one-time signature σ_{OTS}. If this verification is successful, the verifier uses the authentication path to construct the path from the leaf $g(Y_C)$ to the root $v_H[0]$. This path is $(p_0, \ldots, p_H)$. Here $p_0 = g(Y_C)$ and

$$p_h = \begin{cases} g(a_{h-1}||p_{h-1}) & \text{for } l_{h-1} \equiv 1 \mod 2 \\ g(p_{h-1}||a_{h-1}) & \text{for } l_{h-1} \equiv 0 \mod 2 \end{cases}$$

with $1 \leq h \leq H$ and l_h as in (1). So we see that the presence of C in the signature allows to select the right order for the two nodes p_{h-1}, a_{h-1} in the construction of p_h. If $p_H = v_H[0]$ is equal to the (first part of the) public key pk, then the signature is accepted. Otherwise it is rejected.

5.3. Instantiation

The MSS with underlying LD-OTS is an instantiation of the general MSS.

5.4. Security

Definition $((t, q, \varepsilon)$-EU-CMA): A signature scheme $\mathsf{DS} = (\textsc{KeyGen}, \textsc{Sign}, \textsc{Verify})$ is (t, q, ε)-EU-CMA if there is no efficient algorithm $\mathcal{F}$, that runs in time less than or equal to t and needs less than or equal to q signature queries, for which $\mathrm{Prob}\left[\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{F},\mathsf{DS}}(n) = 1\right] \geq \varepsilon$.

Experiment $\mathsf{Exp}^{\mathsf{eu\text{-}cma}}_{\mathcal{F},\mathsf{DS}}(n)$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow \textsc{KeyGen}(n)$
$(m^*, \sigma^*) \leftarrow \mathcal{F}^{\textsc{Sign}(\mathsf{sk},\cdot)}(\mathsf{pk})$
let σ_i be the answer returned by $\textsc{Sign}(\mathsf{sk}, \cdot)$ on input m_i, for $i = 1, \ldots, k \leq q$.
Return 1 iff $\textsc{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$ and $m^* \notin \{m_1, \ldots, m_k\}$.

Definition $((t, \varepsilon)$-collision-resistant): A family $\mathcal{G}(n)$ of hash-functions $g : \{0,1\}^* \to \{0,1\}^n$ is (t, ε)-collision-resistant if there is no efficient algorithm $\mathcal{F}$, that runs in time less than or equal to t, for which $\mathrm{Prob}\left[\mathsf{Exp}^{\mathsf{cr}}_{\mathcal{F},\mathcal{G}(n)}(n) = 1\right] \geq \varepsilon$.

Experiment $\mathsf{Exp}^{\mathsf{cr}}_{\mathcal{F},\mathcal{G}(n)}(n)$

$g \xleftarrow{\$} \mathcal{G}(n)$
$(m_1^*, m_2^*) \leftarrow \mathcal{F}^g(n)$
let $h_i = g(m_i)$ on input m_i, for $i = 1, \ldots, k \leq q$.
Return 1 iff $g(m_1^*) = g(m_2^*)$ and $m_1^* \neq m_2^*$ and $m_1^*, m_2^* \notin \{m_1, \ldots, m_k\}$.

Claim: The Merkle Signature Scheme is (t, q, ε)-EU-CMA as long as the hash-function is (t', ε')-collision-resistant and the underlying OTS is $(t'', q'', \varepsilon'')$-EU-CMA.
(The computation of the values of $t', \varepsilon', t'', q'', \varepsilon''$ will be an exercise.)

Sketch of the proof: The idea of the proof is to use a forger against the Merkle Signature Scheme to break the OTS on the one hand, to construct an adversary to find collisions on the other hand and then combine them to find an adversary for at least one of the two things.

5.5. Merkle Signature Scheme - practicable?

We have seen that the Merkle Signature Scheme has minimal security requirements: The underlying OTS has to be secure and it needs a cryptographic hash-function. The security of the OTS relies on the security of the used one-way function.
So we see that, since no additional computational assumptions such as factoring, discrete logarithm etc. occur, MSS is hard. And we see that the Merkle Signature Scheme has many, many instantiations.
But this also leads to a disadvantage of MSS: There are no standards!
Another question is, whether the Merkle Signature Scheme can be used in practice?! Unfortunately, in the version described so far, there are several efficiency problems:

1. Key size: The public key is a single hash-value which is an n bit string ($n \geq 160$). So this key is very small. However, the secret key is very large since it consists of 2^H OTS signature keys. If LD-OTS is used, then each signature key is of size $2n^2$ bit, such that the whole secret key has a size of $2^{H+1}n^2$ bit (for $n = 160$ this would be $2^{H+1}25600$ bit which is too big for large H).

2. Key generation: To generate the secret key in MSS, 2^H one-time signature keys must be determined. For large H this may be very time-consuming. For the public key the full Merkle tree must be computed (for height H this would mean $2^{H+1} - 1$ nodes). Again, for large H this is very time-consuming. For example, if $H = 40$, then the Merkle tree has $2^{41} - 1 = 2{,}2$ trillion nodes.

3. Authentication path computation: To compute the authentication path, the signer may keep the full Merkle tree. However, for large H this is too inefficient.

4. Signature size: Let $\sigma = (s, Y_s, \sigma_{\text{OTS}}, (a_0, \ldots, a_{H-1}))$ be the signature. The signature size is dominated by the size of the one-time signature. In the LD-OTS case it is because Y_s has a size of n^2 bit, σ_{OTS} of n^2 bit and $(a_0, \ldots, a_{H-1})$ of Hn bit, such that σ has a size of $\sim n^2$ bit. In contrast the size of an elliptic curve signature is of $O(n)$ bit where n is the security parameter.

Since the LD-OTS signs each bit with one bit string of length n, we will now show a method to improve this.

5.6. Winternitz OTS – an improvement of MSS

The idea of the Winternitz OTS is to sign several bits simultaneously, thereby reducing the size of the signature.
The number of bits signed simultaneously is $\omega \geq 2$. (So, now we sign ω bits with one bit string of length n and reduce the size from $O(n^2)$ to $O(n^2/\omega)$.)
To define the Winternitz OTS we use the following notations:

$$t_1 = \left\lceil \frac{n}{\omega} \right\rceil, \qquad t_2 = \left\lfloor \frac{\lfloor \log_2 t_1 \rfloor + 1 + \omega}{\omega} \right\rfloor, \qquad t = t_1 + t_2.$$

KeyGen:
The signature key is

$$X = (x_{t-1}, \ldots, x_0) \xleftarrow{\$} \{0,1\}^{(n,t)}$$

with size nt bit, and the verification key is $Y = (y_{t-1}, \ldots, y_0)$, where

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

Sign:
The Winternitz signature generation for a message or document $d = (d_{n-1}, \ldots, d_0) \in \{0,1\}^n$ works as follows: We prepend a minimal number of zeros to d such that the length is divisible by ω. (We call the extended bit string d.) We then can write d as

$$d = b_{t-1} || \ldots || b_{t-t_1} = b_{t-1} || \ldots || b_{t_2},$$

where b_i, $t_2 \leq i \leq t-1$, is a bit string of length ω. Additionally, the checksum

$$c = \sum_{i=t-t_1}^{t-1} (2^\omega - (b_i)_{10})$$

is computed, where $(b_i)_{10}$ is the decimal representation of the bit string b_i. Since $c \leq t_1 2^\omega$, the length of this checksum is smaller than or equal to $\lfloor \log_2(t_1 2^\omega) \rfloor + 1 = \lfloor \log_2 t_1 \rfloor + \omega + 1$. Again a minimal number of zeros is prepended to the binary representation of c and the resulting bit string is (again) denoted by c. When we write c as

$$c = b_{t_2-1} || \ldots || b_0,$$

then the whole signature σ of document d in the Winternitz OTS is computed as

$$\sigma = (\sigma_{t-1}, \ldots, \sigma_0) = (f^{(b_{t-1})_{10}}(x_{t-1}), \ldots, f^{(b_0)_{10}}(x_0)).$$

Verification:
The verifier calculates $b_{t-1}, \ldots, b_0$ as above and checks whether

$$f^{2^\omega - 1 - (b_i)_{10}}(\sigma_i) = y_i, \qquad 0 \leq i < t.$$

We discuss the advantage of this method: The key length is reduced to $O(2n^2/\omega)$, the signature length to $O(n^2/\omega)$. The number of evaluations of f is $O(2^\omega n/\omega)$. This means that the time complexity grows exponentially in ω so that this method is only of limited use. An alternative is using an OTS that is based on lattice problems and has a signature size of $O(n)$.
Now we want to consider the problem of the size of the secret key as well as the key generation and authentication path generation problem.

5.7. Secret key problem: Coronado's idea

Recall a Merkle tree. It consists of $2^{H+1} - 1$ nodes and has 2^H underlying OTS key pairs from which the secret keys X_i, $0 \leq i \leq 2^H - 1$, form the secret key X. As we can see the size of the secret key is too big. Thus we now introduce an improvement of this which was the idea of Coronado.
His idea was to use a pseudo-random number generator (PRNG), where $\mathsf{PRNG} : \{0,1\}^n \to \{0,1\}^{2n}, seed \mapsto (r_1, r_2)$ and r_1, r_2 "look random", from which we can construct each single secret key for the OTS. The PRNG must be public.
The secret key generation works as follows: At first one has to choose an n bit string s_0 (seed) uniformly at random which is the secret key at the beginning of the key generation. From this s_0 we gain $\mathsf{PRNG}(s_0) = (s_1, r_{0,0})$ with two n bit strings $s_1, r_{0,0}$ which seem to be chosen uniformly at random.
Now $r_{0,0}$ is used with PRNG to create the OTS secret key X_0: Using PRNG with input $r_{0,0}$ we get $\mathsf{PRNG}(r_{0,0}) = (X_0[0], r_{0,1})$, where $X_0[0]$ are the first n bits of the OTS secret key X_0 and $r_{0,1}$ will be used as next input for PRNG to construct $(X_0[1], r_{0,2})$. By applying PRNG t times we get the first OTS secret $X_0 = (X_0[0], \ldots, X_0[t-1])$.
With s_1 we can do the same thing for the OTS secret key X_1, and so on.
Since we create an OTS secret key to obtain a new signature, we can remove the preceding seed (which was used for the generation of a now used OTS secret key) and only have to store the current one to create further OTS secret keys. (E.g. we can delete s_0 if we obtained s_1 and X_0, and thus we just have to save the current seed s_1 as secret key.)
The whole procedure therefore will look similar to the one in Figure 5.
By this improvement the size of the secret key is of n bit and no longer of $tn2^H$ bit as in the MSS case (using Winternitz).
And since we delete old seeds (old secret keys) and only store the actual seed (secret key), we gain a forward secure scheme. This means that no "old" (used) OTS secret key can be gained by the knowledge of a "new" one.
For a general adoption, one can replace the usage of all random parts $r_{k,i}$ from above by the usage of the n bit string $r_k := r_{k,0}$ in the OTS key generation procedure to obtain the secret OTS key X_k.

5.8. Key generation problem: Tree Chaining

Because the generation of a full Merkle tree of height H is expensive, one could ask which value for H would be adequate, e.g.: "Is a height of $H = 40$ good?"

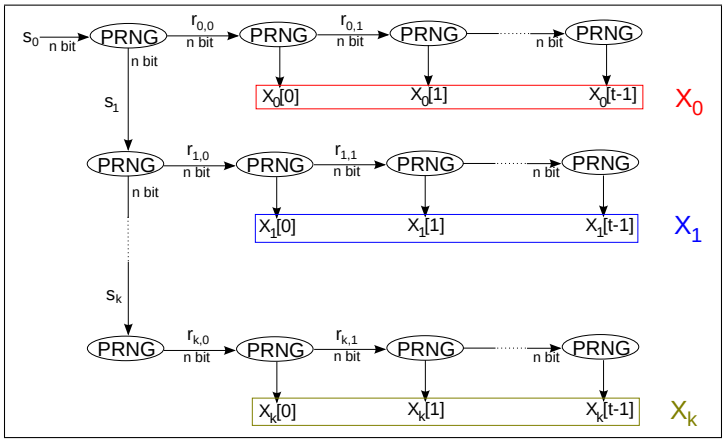

Figure 5: Using a PRNG to create OTS secret keys

Since we have to store $2^{41} - 1$ nodes in the full Merkle tree, one could answer: "No.". To create a tree of height 40 would be of no good idea. But, since signing is fast, there is an idea that improves the key generation process: We can distribute the key generation process over the signature, which means that we start with a "partial" (secret) key and find the complete secret while we sign documents or messages.
In other words, one generates two trees of height $H/2$ (instead of a tree of height H) and combines them for signing as can be seen in Figure 6.

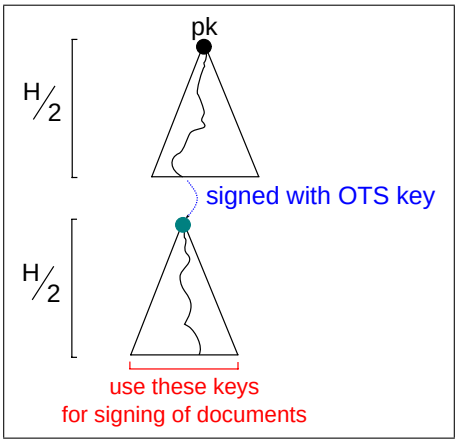

Figure 6: Using Tree Chaining for signing

Since the verifier only knows the document d and the root pk of the tree on top, we have to assure him of the correctness of the OTS secret we used.
Because of this, we construct the signature σ for a document d as follows:
We write down the index i of the OTS secret key X_i in the lower tree, the OTS signature σ_{OTS} of d generated with X_i, the OTS public key Y_i and the authentication path A_i. Then we can construct the path in the lower tree and compare the result with the root

r of the lower tree.
But how can we as the signer now assure the verifier to know the correctness of r? Therefore we use the current OTS secret key $X'_{i'}$ of the upper tree to sign r and compute its OTS signature σ'_{OTS} which can be verified using the OTS public key $Y'_{i'}$ and the corresponding authentication path $A'_{i'}$ in the upper tree.
Thus the signature will look like

$$\sigma = (i, \sigma_{\text{OTS}}, Y_i, A_i, i', \sigma'_{\text{OTS}}, Y'_{i'}, A'_{i'}).$$

(In some sort you can think of using two trees for signing a document by "gluing" them together and then obtaining the right OTS signature, OTS public key and the authentication path.)
If we go on doing this for each new signature, we will end when we did $2^{H/2}$ signatures because the lower tree was fully used. So we see that we have to generate (more) new subtrees as we go along.
Additionally, we can use another allocation for the height of the trees: We can use more levels of subtrees of varying height. This leads to a more flexible way of handling big(ger) trees by making the key generation much faster, but we have to recognize that the signature by this way gets longer.

5.9. Authentication path generation problem

As a last thing, we want to consider the problem of the authentication path generation. Szydlo was one of the first pioneers who thought about the question if and how one can use the authentication path Auth$[s]$ for a leaf with index s to find the authentication path for the leaf with index $s+1$ in a Merkle hash-tree.
Therefore assume we have a hash-tree of height H and denote the node in the Merkle tree on height h with index $j \in \{0, \ldots, 2^h - 1\}$ by $v_h[j]$. (So the initial authentication path can be denoted by $(v_h[1])_{0 \leq h < H}$.)
The idea of the improvement is that some of the nodes change and some do not, which then can be reused.
Consider the following: For a leaf with index s we can write s in its binary representation as

$$s = b_0 + 2b_1 + 2^2 b_2 + \ldots + 2^{H-1} b_{H-1}.$$

Claim: Let $\tau = \max\{h : 2^h | s+1\}$. Then the authentication paths Auth$[s]$ and Auth$[s+1]$ differ in all nodes on heights $0, \ldots, \tau$ and no others.

E.g. we have $s = 1 \cdot 2^0 + 12^1 + \ldots + 1 \cdot 2^{\tau-1} + 0 \cdot 2^\tau + b_{\tau+1} 2^{t+1} + \ldots + b_{H-1} 2^{H-1}$ and thus $s+1 = 0 \cdot 2^0 + 0 \cdot 2^1 + \ldots + 0 \cdot 2^{\tau-1} + 1 \cdot 2^\tau + b_{\tau+1} 2^{t+1} + \ldots + b_{H-1} 2^{H-1}$. On heights $0, \ldots, \tau - 1$ the nodes in the authentication path change from left to right and on height τ from right to left.
Now the question is: How to compute the left node?
From previous computations we know something like in Figure 7.
So we can compute the new left node from previous knowledge.

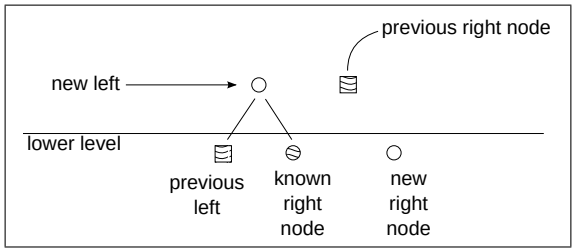

Figure 7: Knowledge of previous computations

The new right nodes have to be computed from scratch using the tree-hash algorithm. We gain a new flexibility for the last step: We can choose a height T from which we store the upper part of the tree as in Figure 8, a part for which we do not want to compute nodes in it because that is expensive, and use the tree-hash algorithm only for lower heights in which it is expected to be fast since we only consider smaller subtrees.

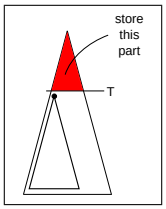

Figure 8: New flexibility by storing the upper part of the tree

Part II.
Lattice-based Cryptography

6. Lattices

6.1. Definitions and notations

Definition: Let $n \in \mathbb{N}_{>0}$. A *lattice* is a discrete abelian subgroup of $\mathbb{R}^n$.
For most applications we need a more tangible definition. We need a "handle" on the lattice. A natural handle is a *basis*.
Definition: Let $L \subseteq \mathbb{R}^n$ be a lattice. There is a basis $B := (b_1, \ldots, b_d)$ which consists of linearly independent column vectors b_i such that $L = \mathcal{L}(B) := \text{span}_{\mathbb{Z}}(B) = \{Bx : x \in \mathbb{Z}^d\}$.
As an example take $L_1 := \begin{pmatrix}0\\1\end{pmatrix}\mathbb{Z} + \begin{pmatrix}1\\0\end{pmatrix}\mathbb{Z} = \mathbb{Z}^2$ or $L_2 := \begin{pmatrix}0\\1\end{pmatrix}\mathbb{Z} \subseteq \mathbb{Z}^2$ being lattices in $\mathbb{Z}^2$.
Definition: The number $d \geq 1$ in the Definition above is a lattice constant. It is the *dimension*

$$d := \dim(L) := \text{rank}_{\mathbb{R}}(B)$$

of the lattice L.
In the example we have $\dim(L_1) = 2$ and $\dim(L_2) = 1$.
There is one important thing at this point that has to be mentioned: The span of a lattice is the $\mathbb{R}$-span of its basis, $\text{span}(L) := \text{span}_{\mathbb{R}}(B)$, which differs from the definition of $L = \text{span}_{\mathbb{Z}}(B)$. You can see the difference in Figure 9.

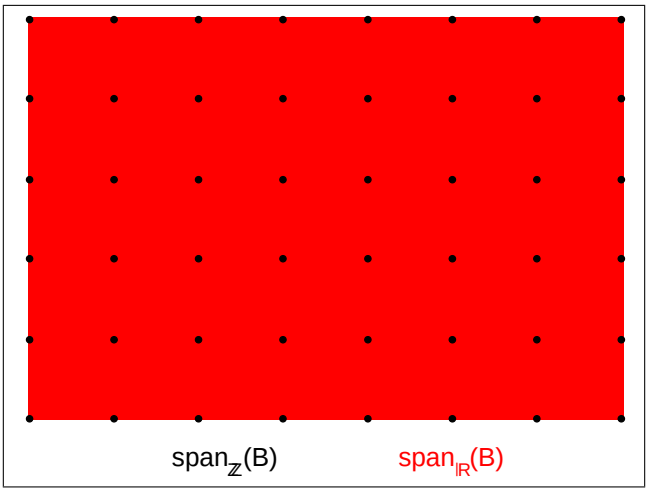

Figure 9: The difference of $\mathbb{R}$- and $\mathbb{Z}$-span

Another important fact about lattices is that a basis for a lattice is not unique: For

$d \geq 2$, any lattice L has infinitely many bases (why?). For $d = 1$, one has only two bases. This is due to the fact that if one basis is (b_1), then the other basis can only be $(-b_1)$, which you can prove at home. Two bases are the same if they are bitwise equal.
Fact: Let $L \subseteq \mathbb{R}^n$ be a lattice with $\dim(L) = d$ and $L = \mathcal{L}(B)$. The *set of all bases* of L is

$$B \cdot GL_d(\mathbb{Z}) := \{BT : T \in GL_d(\mathbb{Z})\},$$

where the general linear group $GL_d(\mathbb{Z})$ is the set of all (invertible) matrices $T \in \mathbb{Z}^{d \times d}$ with $\det(T) = \pm 1$ (T in this context is called *unimodular*).
Example: Let $n = d = 1$ and $B = (b_1) = (\pi)$. Then all bases have to be of the form $\{b_1 t : t, t^{-1} \in \mathbb{Z}\} = \{\pi, -\pi\}$. So, we can see that taking a basis like (2π), (-2π), etc. would lead to a different lattice.
Definition: The *determinant* of a lattice $L = \mathcal{L}(B)$ is defined as

$$\det(L) := \sqrt{\det(B^T B)}.$$

Proposition: Let $L \subseteq \mathbb{R}^n$ be a lattice generated by basis B ($L = \mathcal{L}(B)$). The determinant $\det(L) = \sqrt{\det(B^T B)}$ is a lattice constant.
Proof: The proof follows from Fact 6.1: In case $d = 1$ we see directly that $\mathcal{L}(B) = \mathcal{L}(-B)$ and $\sqrt{\det((-B)^T(-B))} = \sqrt{\det(B^T B)}$. For the case $d \geq 2$ take any basis $C \in B \cdot GL_d(\mathbb{Z})$. Then we know that there exists a matrix $T \in GL_d(\mathbb{Z})$ with $BT = C$. So we can conclude

$$\begin{aligned}\sqrt{\det(C^T C)} &= \sqrt{\det((BT)^T(BT))} = \sqrt{\det(T^T B^T BT)} \\ &= \sqrt{\det(T^T)\det(T)\det(B^T B)} = \sqrt{\det(B^T B)} = \det(L),\end{aligned}$$

since we have $\det\left(T^T\right) = \det(T)$ and $\det(T) \in \{\pm 1\}$. □
Most of the time we will deal with lattices of special form: For *full-rank* lattices ($n = d$), there is a shortcut for the determinant, i.e. $\det(L) = |\det(B)|$.
Example: Figure 10 will illustrate the geometrical interpretation of the determinant of a lattice L.
Definition: Let $\mathcal{L}(B) = L \subseteq \mathbb{R}^n$ and $\dim(L) = d$. The *parallelepiped* $P(B)$ is defined as

$$P(B) = \{Bx : x \in [0,1)^d\}$$

and its volume is $\text{vol}(P(B)) := \det(L)$. Thus, the volume is an invariant of the lattice and its shape only depends on the basis B (cf. Figure 10).
We can write $\text{span}(L) = L + P(B)$.

6.2. Lattice Problems

Now we want to introduce the most important computational/search problems in lattices.
Computation on lattices often involves solving certain standard problems, such as

- shortest vector problem

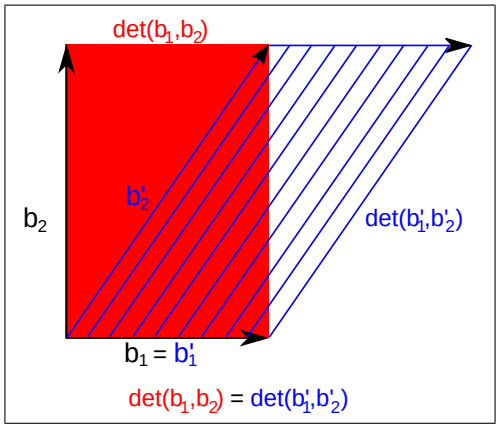

Figure 10: Geometrical interpretation of the determinant of a lattice

- closest vector problem
- shortest independent vector problem, and
- many many others which are mostly derivatives and relate on the problem to "find something short".

Typically, they are defined with respect to the minimum distance of a lattice (which is another invariant).
Definition: The *minimum distance* of a lattice L is the length of the shortest vector $v \in L \setminus \{0\}$.
Definition: Let $L \subseteq \mathbb{R}^n$ be a lattice. The *i-th successive minimum* $\lambda_i(L)$ is the radius of the smallest sphere that contains i linearly independent vectors of L.
The first successive minimum corresponds to the minimal distance.
Examples:

- For $L_1 = \mathbb{Z}^2$ we have $\lambda_1(L_1) = 1 = \lambda_2(L_1)$.
- For $L_2 = \begin{pmatrix}1\\1\end{pmatrix}\mathbb{Z} + \begin{pmatrix}1\\0\end{pmatrix}\mathbb{Z}$ we likewise have $\lambda_1(L_2) = 1 = \lambda_2(L_2)$.
- For $L_3 = \begin{pmatrix}2\\0\\0\end{pmatrix}\mathbb{Z} + \begin{pmatrix}0\\2\\0\end{pmatrix}\mathbb{Z} + \begin{pmatrix}2\\2\\1\end{pmatrix}\mathbb{Z}$ we have $\lambda_1(L_3) = 1$ and $\lambda_2(L_3) = \lambda_3(L_3) = 2$.

Now the question, naturally arising at this point, is: For any lattice L, can we find a basis $B = (b_1, \ldots, b_d)$ of L with $||b_i|| = \lambda_i(L)$ for $i = 1, \ldots, d$?
For $d \geq 5$ this is wrong, as we can see in the next

Example: Consider

$$L = \mathcal{L}(b_1, \ldots, b_d) = \mathcal{L}\left(\begin{pmatrix}2\\0\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\2\\0\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\2\\0\\0\end{pmatrix}, \begin{pmatrix}0\\0\\0\\2\\0\end{pmatrix}, \begin{pmatrix}1\\1\\1\\1\\1\end{pmatrix}\right).$$

We have $||b_i|| = 2$ for $i = 1, \ldots, 4$, but $||b_5|| = \sqrt{5} > 2$. Since $b_5' := 2b_5 - \sum_{i=1}^{4} b_i = (0, 0, 0, 0, 2)^T \in L$, we know that $\lambda_{1/2/3/4/5} = 2$.
Another question is: What happens if we take the basis $C := (b_1, \ldots, b_4, b_5')$? Can we achieve $\mathcal{L}(B) = \mathcal{L}(C)$? The answer is "No.", because $b_5 \notin \mathcal{L}(C)$, what in particular means that there is no unimodular matrix $T \in GL_d(\mathbb{Z})$ with $BT = C$ (since $b_5' = 2b_5 \ldots$).
Now we come to the definitions of the most important lattice problems.
Definition ($(\gamma$-)SVP): The (approximate) shortest vector problem (γ-)SVP for $\gamma \geq 1$ is defined as follows: Let $L \subseteq \mathbb{R}^n$ be a lattice. Find $x \in L \setminus \{0\}$ with $||x|| \leq (\gamma)\lambda_1(L)$.
Example: For an example consider the lattice $L = \begin{pmatrix}1\\1\end{pmatrix}\mathbb{Z} + \begin{pmatrix}0\\1\end{pmatrix}\mathbb{Z}$. The SVP leads to the solution(s) $x \in \left\{\begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}0\\-1\end{pmatrix}, \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}-1\\0\end{pmatrix}\right\}$. So we see that the solution is not unique. There are at least two solutions $v, -v$.
Definition ($(\gamma$-)CVP): The (approximate) closest vector problem (γ-)CVP is defined as follows: Let $L \subseteq \mathbb{R}^n$ be a lattice and $t \in \text{span}_{\mathbb{R}}(L)$. Find $x \in L$ with $||t-x|| \leq (\gamma)||t-w||$ for all $w \in L$.
Example: Consider Figures 11 and 12.

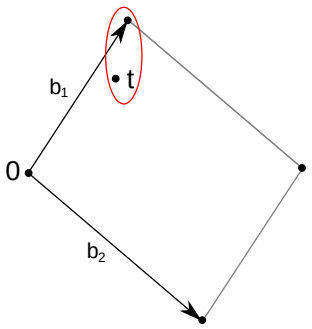

Figure 11: Unique solution for the CVP

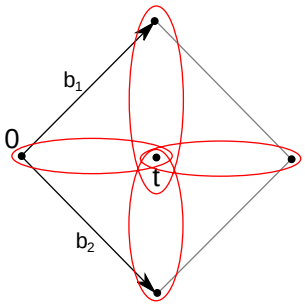

Figure 12: No unique solution for the CVP

We see that in Figure 12 we have $L = \begin{pmatrix}1\\1\end{pmatrix}\mathbb{Z} + \begin{pmatrix}1\\-1\end{pmatrix}\mathbb{Z}$ and for $t = \begin{pmatrix}1\\0\end{pmatrix}$ we do not obtain a unique solution for the CVP.
Definition ($(\gamma$-)SIVP): The (approximate) shortest independent vector problem (γ-)SIVP is defined as follows: Let $L \subseteq \mathbb{R}^n$ be a lattice with $\dim(L) = d$. Find $v_1, \ldots, v_d$ linearly independent such that $\max\{||v_1||, \ldots, ||v_d||\} \leq (\gamma)\lambda_d(L)$.

In general, we do not know the values of $\lambda_i(L)$ in advance and thus we cannot prove the correctness of a "solution". Still, we want to be able to make a statement about the relative length of a vector or about the quality or "reducedness" of a basis.
For this purpose we use the Hermite-SVP.
Definition (Hermite-SVP): Given a lattice $L \subseteq \mathbb{R}^n$ with $\dim(L) = d$ and $\gamma > 0$. Find $x \in L \setminus \{0\}$ with $||x|| \leq \gamma^d \det(L)^{1/d}$.
Example: Interestingly, easy and hard instances of this problem are very close solvable. For $\gamma = 1.02$ the Hermite-SVP is solvable in practice, even in higher dimensions, but for $\gamma = 1.01$ it becomes intractable for large d, say $d \geq 500$. (It also depends on the structure of L.)
Whether these problems are efficiently solvable, i.e. in polynomial time in n, depends on the input description (basis) B of L: Therefore consider the following Figure 13.

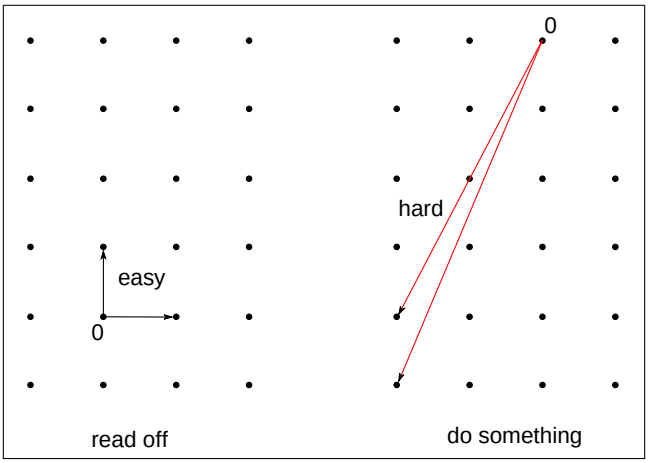

Figure 13: The hardness of the problems relies on the size of the basis

If the vectors in B are short and almost orthogonal, we can basically read off the solutions. But if the basis consists of long vectors, it is harder and we have to "do something" to solve those problems. Now we turn to what this "do something" is.

7. Lattice (Basis) Reduction

We start with lattices $L \subseteq \mathbb{R}^n$ of dimension $d = 2$ as there is a polynomial-time algorithm in the input size that solves SVP (and others).
Since Gauß and Lagrange developed the same idea independently, the algorithm can be called "Gauß algorithm" or "Lagrange algorithm". This shall just be a remark for you not to be confused reading the same algorithm in different papers with different names. We will call it Gauß reduction algorithm here.
The Gauß reduction algorithm consists of two steps, called Normalize and Swap, that are iterated until a certain condition holds.

- Normalize:
  For Normalize, we require the Gram-Schmidt orthogonalization (GSO) that is computed with the GS-process.

  Definition: Given a matrix $B = (b_1, \ldots, b_d) \in \mathbb{R}^{n \times d}$. The GS-process does the following: Set $\tilde{b}_1 := b_1$ and recursively define $\tilde{b}_i := b_i - \sum_{j=1}^{i-1} \mu_{i,j} \tilde{b}_j$ for $i \geq 2$, where the $\mu_{i,j} := \dfrac{\left\langle b_i, \tilde{b}_j \right\rangle}{||\tilde{b}_j||^2}$ denote the Gram-Schmidt coefficients.

  Example: Consider the two vectors b_1, b_2 as in Figure 14.

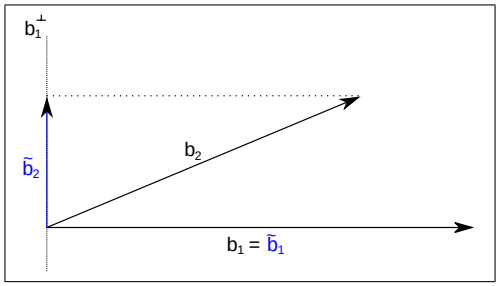

Figure 14: Proceedings in the GS-process

The vector b_1 stays the same ($\tilde{b}_1 := b_1$), but the second vector b_2 is projected onto the orthogonal space $b_1^{\perp}$ of b_1. The output of the GS-process thus is $\tilde{B} := (\tilde{b}_1, \tilde{b}_2)$.

An alternative notation that will be helpful later is that of orthogonal projections of the basis $B = (b_1, \ldots, b_d)$ via $\pi_i : \mathbb{R}^d \to \text{span}\,(b_1, \ldots, b_{i-1})^{\perp}$, in which we have the special case of $\pi_i(b_i) = \tilde{b}_i$.

The GS-process can shorten and orthogonalize vectors. Why can we not just use it to find an orthogonal basis of $L = \mathcal{L}(B)$ or its shortest vector? The answer is that the process does not necessarily yield lattice vectors. It does not respect or preserve the structure of the lattice. In particular, the matrix that corresponds to π_i is not unimodular.

But, it can be used as a guiding principle.

Normalize(b_1, b_2)
Input: $b_1, b_2 \in L$ with $||b_1|| \leq ||b_2||$
Output: $b' \in L$ where $b' := b_2 - \lfloor \mu_{2,1} \rceil\, b_1$

When $|\mu_{2,1}| < \frac{1}{2}$, nothing happens and we cannot improve b_2 with b_1 any further. Observe that, after Normalize(b_1, b_2), we have

$$\langle b', b_1 \rangle = \langle b_2 - \lfloor \mu_{2,1} \rceil\, b_1, b_1 \rangle = \langle x b_1^{\perp} + y b_1 - \lfloor y \rceil\, b_1, b_1 \rangle = (y - \lfloor y \rceil)||b_1||^2,$$

since one can write b_2 as $b_2 = x b_1^{\perp} + y b_1$ because b_1, b_2 are linearly independent.

The observation leads to another

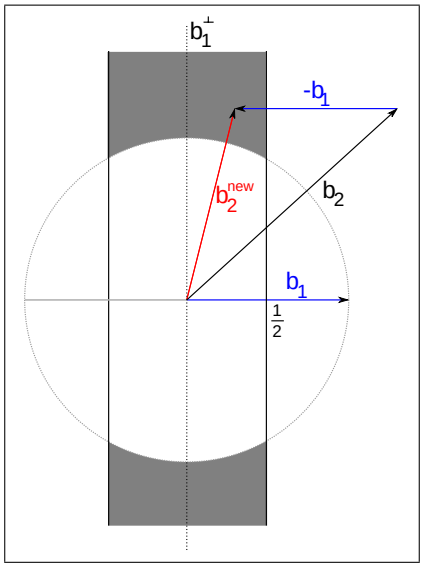

Figure 15: Proceedings during the Normalize-step

Fact: After the Normalize-step we have $|\mu_{2,1}^{\text{new}}| \leq \frac{1}{2}$.

For the angle this means $|\angle(b_1, b_2)| \in [60°, 120°]$, which is an easy task for the reader to do him-/herself at home (using the cosine will help).

Definition: A normalized basis $B \in \mathbb{R}^{n \times d}$ for $d \in \mathbb{N}$ is *size-reduced* iff $|\mu_{i,i-1}| < \frac{1}{2}$ for $i = 2, \ldots, d$.

This definition leads to the question: Why is Normalize not enough? To see this we give another

Example: Let $b_1 := \begin{pmatrix} 3 \\ 4 \end{pmatrix}$ and $b_2 := \begin{pmatrix} 6 \\ 7 \end{pmatrix}$. We obtain $\mu_{2,1} = \frac{46}{25}$ and thus $b_2' := b_2 - \lfloor \mu_{2,1} \rceil b_1 = \begin{pmatrix} 6 \\ 7 \end{pmatrix} - 2 \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$. Now, we have $|\mu_{2,1}^{\text{new}}| \leq \frac{1}{2}$, but we see that we can shorten b_1 with b_2'.

- Swap:
  Therefore, we need the second operation Swap which switches the two vectors via $(b_1, b_2) \mapsto (b_2, b_1)$.

  If we now have $||b_1|| \leq ||b_2||$, we can call Normalize again.

  Example: In the example above, we call Normalize with $(\begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 3 \\ 4 \end{pmatrix})$ and thus obtain $b_2' = \begin{pmatrix} 3 \\ 4 \end{pmatrix} - \lfloor \frac{-4}{1} \rceil \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$.

  After a polynomially bounded number (in the input length) of iterations of

  Normalize; Swap;

we have $||b_1|| \leq ||b_2||$ and $|\mu_{2,1}| \leq \frac{1}{2}$ and we can stop and obtain a so-called *Gauß-reduced basis* $B = (b_1, b_2)$. This is because $|\mu_{2,1}| < \frac{1}{2}$ implies that we cannot improve b_2 with b_1. For $|\mu_{2,1}| = \frac{1}{2}$, we would swap back and forth without actually improving the lengths. In addition, swapping does not help because $||b_2|| \geq ||b_1||$. Thus, adding b_2 to b_1 would only make the basis worse.

The Gauß reduction algorithm becomes

Gauß:
Input: basis (vectors) $b_1, b_2 \in L$ with $L = \mathcal{L}(b_1, b_2)$ and $||b_1|| \leq ||b_2||$
Output: a Gauß-reduced basis $b_1', b_2' \in L$ with $L = \mathcal{L}(b_1', b_2')$
`while` $|\mu_{2,1}| > \frac{1}{2}$ `do`
    Normalize: $b_2 := b_2 - \lfloor \mu_{2,1} \rceil \, b_1$
    `if` $||b_2|| < ||b_1||$ `then` Swap b_1 and b_2
`return` (b_1, b_2)

An important fact about the time needed for the Gauß reduction algorithm is the following:
Fact: The Gauß algorithm runs in time $\mathcal{O}\big((\text{bitsize}(B))^2\big)$.
And another fact is
Fact: The Gauß reduction algorithm finds $b_1, b_2 \in L$ where $L = \mathcal{L}(b_1, b_2)$ with $\lambda_1(L) = ||b_1||$ and $\lambda_2(L) = ||b_2||$ which is the best possible.
Proposition: Given a basis B of $L = \mathcal{L}(B)$. Then the Gauß algorithm on input B outputs a basis C with $L = \mathcal{L}(B) = \mathcal{L}(C)$.
Proof: This is clear since the steps Normalize and Swap only perform unimodular operations. The detailed proof is left to the reader as an exercise.
The Gauß reduction algorithm can be generalized to lattice dimensions $d \leq 4$ (non trivial!). Beyond that, this simple framework breaks down and we need a more sophisticated approach.

8. Lattices in a post-quantum world

We want to consider the question: Why are lattice problems good for post-quantum cryptography? This is due to the fact that SVP is $\mathcal{NP}$-hard.
But, why is this a good reason for believing in the security of cryptographic schemes?
Consider the problem of factoring: Is factoring $\mathcal{NP}$-hard? Of course, factoring is in $\mathcal{NP}$ and the language to describe factoring is $C := \{(n, c) : n \text{ has a factor } \leq c\}$. Now $C \in \mathcal{P}$ iff factoring is in $\mathcal{P}$, since $\mathbb{N} - C = \mathbb{P} \cup \{1\}$, so that there would be a polynomial time algorithm for deciding whether a string $s \in \mathbb{P}$ or not. If we assume that C is $\mathcal{NP}$-complete, we can conclude $\mathcal{P} = \mathcal{NP}$. But, in cryptography we hope that $\mathcal{P} \neq \mathcal{NP}$ (so that C is not $\mathcal{NP}$-complete).
When you construct cryptosystems based on ($\mathcal{NP}$-complete) problems you have to find a way of constructing hard instances of the problem.

Example: Is factoring of a 1024 bit number hard? It depends on the number itself, as we will see now: An easy instance of the factoring problem is given by the number 2^{1023}. But can you give some hard instance? This task is not obvious (which also is the problem of RSA). A hard instance would be $n = pq$ with primes $p \approx q$, $p \neq q$, and $\lfloor \log_2 n \rfloor = 1024$.
We do not only have the problem of finding hard instances in the RSA case, but also with Discrete Logarithms and lattices, as we see now.
Let $n = 1024$. Give an example for a lattice $L \subseteq \mathbb{Z}^n$ with $\dim(L) = n$ in which the SVP is easy and an example for which the SVP is hard to solve. An easy instance of the SVP is given by the basis

$$\begin{pmatrix} 1 & & & 0 \\ & 2 & & \\ & & \ddots & \\ 0 & & & 2 \end{pmatrix}$$

because the shortest vector can be read off. So we see that finding hard instances of the SVP is the hardest part in post-quantum cryptography.
Another $\mathcal{NP}$-complete problem is the knapsack problem in which one is given some integer weights $a_1, \ldots, a_n$ and a knapsack-size parameter b and has to decide whether there is $c := (c_1, \ldots, c_n) \in \{0,1\}^n$ such that $b = \sum_{i=1}^{n} c_i a_i$ and, if so, to find c. There have been some cryptosystems based on the knapsack problem using superincreasing knapsacks (with $a_1 \ll a_2 \ll a_3 \cdots$) and books have been written about those knapsack-based cryptosystems, but all of them became useless since all knapsack-based cryptosystems were broken via lattice basis reduction(s).

9. How to find hard instances: the idea of Ajtai

Fortunately, Ajtai proposed some idea for solving the problem to find "hard instances": Ajtai reduced the problem of finding short vectors in any of the lattices in dimension n to the problem of finding (approx.) short vectors in a random lattice of dimension $m (\gg n)$. This is called "worst case to average case reduction". (As a first thought you can think of $m \approx n \log_2 n$, as an example take $n \geq 300$ and obtain $m \approx 2400$.)
If finding short vectors in a random lattice of dimension m is easy, then finding short vectors in all lattices of dimension n is easy. This means, if there exists at least <u>one</u> hard instance of the SVP in dimension n, then the SVP for a random lattice instance in dimension m is hard. This construction is conceptually different from RSA in which the primes p, q have to be random.
Now we come to the construction of Ajtai who constructed a compression function f with the property that f is collision-resistant as long as the SVP in some lattice of dimension m is hard: Therefore, we have to choose some positive integers $n, m, q, d \in \mathbb{N}$ and select a matrix $A \in_R \mathbb{Z}_q^{n \times m}$ uniformly at random. (A possible choice could be $d = 2$, $q = n^2$ and $m > n \log_2 q / \log_2 d = 2n \log_2 n$.) Then the compression function is (in general) given by

$$f_A : \mathbb{Z}_d^m \to \mathbb{Z}_q^n : \; y \mapsto Ay \mod q.$$

Why is f_A a compression function? For $y = (y_1, \ldots, y_m) \in \mathbb{Z}_2^m$ of binary length $m \log_2 d = m$ and $x = Ay = (x_1, \ldots, x_n) \in \mathbb{Z}_n^q$ of binary length $n \log_2 q$ the inequality $m > n \log_2 q / \log_2 d$ guarantees the compression property. A more typical compression factor in fact would be

$$\frac{m \log_2 d}{n \log_2 q} \approx 2.$$

Collisions of the compression function f_A correspond to short vectors in the set $L = \{y \in \mathbb{Z}^m : Ay \equiv 0 \mod q\}$:

- L is a lattice since it is a subgroup (for two elements $y, y' \in L$ we obtain $y + y' \in L$) and because of $L \subseteq \mathbb{Z}^m$ it is discrete as a point set.
- A collision $(y, y') \in \{0,1\}^{2m}$ means that $Ay \equiv Ay' \mod q \iff A(y - y') = 0 \mod q$, and we know $y - y' \in L$ for $y, y' \in L$. The resulting vector $y - y'$ is short because its entries are in $\{0, \pm 1\}$.

For Ajtai's construction one has to choose parameters n, m, d and q (and as we saw, a good choice could be $d = 2$, $q = n^2$ and $m > n \log_2(q) / \log_2(d)$, in which the last inequality is induced by the compression condition).
That the function $f_A : \mathbb{Z}_d^m \to \mathbb{Z}_q^n, y \mapsto Ay \mod q$ for randomly chosen $A \in_R \mathbb{Z}_q^{n \times m}$ is a hash-function, which maps bit strings of length m to bit strings of total length $n \log_2(q)$, we already discussed above.
We introduced the meaning of "secure" lattices by taking a view at short vectors which can be constructed out of collisions of the hash-function f_A as follows: For a collision (y, y') with $y \neq y'$ we have $A(y - y') \equiv 0 \mod q$ and, thus, the vector $y - y'$ is in the lattice $\Lambda_q^\perp(A) := \{z \in \mathbb{Z}^m : Az \equiv 0 \mod q\}$. It is short because $y, y' \in \mathbb{Z}_2^n$ are bit strings.
The "theorem" of Ajtai now states that finding short vectors in a randomly chosen lattice $\Lambda_q^\perp(A)$ leads to finding short vectors for all lattices of dimension n, whereby "short" in this context means $\leq \sqrt{m} \cdot \mathcal{O}(\sqrt{n}) \cdot \lambda_1(L)$.

9.1. Practical questions about Ajtai's construction

Now we want to consider *practical* questions of Ajtai's constructed hash-function, e.g.: Can we submit this hash-function as a proposal to the SHA-3 competition? Besides the fact that the deadline for submitting is over, we should make more plausible why it would be a good candidate and therefore prove more properties like one-wayness or pseudo-randomness and, of course, we have to show that Ajtai's construction is efficient. But the last point is in fact not true: The drawback is that the function f_A is represented by the matrix $A \in \mathbb{Z}_q^{n \times m}$ itself, and with $m > 2n \log_2(n)$ we see that the size is $> n^2$. Compared to the RSA case, in which we are only linear in the security parameter, we here are quadratic in n which is too big (think of small devices onto which Ajtai's hash-function f_A should be used). The operations $Ay \mod q$ are efficient though: Consider $n = 2^9 = 512$ and $q = n^2 = 2^{18}$, such that the bit string y is mapped to the bit string Ay

of length $18 \cdot 2^9$. Since $A \in \mathbb{Z}_q^{n \times m}$ and $y \in \{0,1\}^m$, we only have addition of bit strings of length 18 and the reduction modulo q is just truncation.
Nevertheless, can we improve the efficiency of Ajtai's construction using a better way of representing the lattices we deal with with fewer bits?

10. Efficiency Improvements using Ideal Lattices

Let the parameters n, m, q and d be as above.
We now construct the matrix A in a special form: Take an irreducible (over $\mathbb{Z}$) and monic polynomial $f \in \mathbb{Z}[x]$ of degree n and consider the ring

$$R := \mathbb{Z}[x]/\langle f \rangle,$$

where *monic* means that the leading coefficient is 1 and $\langle f \rangle$ is the ideal generated by f. The elements of R are residue classes of polynomials of degree less than n with integer coefficients and the operations on R are addition and multiplication modulo f and inversion (if possible) using the extended Euclidean Algorithm in $\mathbb{Z}[x]$.
For a first understanding you can think of $\mathbb{Z}/n\mathbb{Z}$ where the elements are residue classes modulo n and the operations are addition and multiplication with reduction modulo n and inversion (if possible) using the extended Euclidean Algorithm in $\mathbb{Z}[x]$.
As an example we have

	$\mathbb{Z}/n\mathbb{Z}$	$R = \mathbb{Z}[x]/\langle f \rangle$
	$n = 7$	$f(x) = x^2 + x + 1$
elements:	$\overline{0}, \overline{1}, \ldots, \overline{6}$	$\overline{0}, \overline{1}, \overline{x}, \overline{x+1}, \ldots$
	reduce modulo 7	reduce modulo $x^2 + x + 1$
	finitely many elements	set of representatives: set of all polynomials of degree ≤ 1 with integer coefficients $\rightarrow$ infinitely many

Furthermore, we will write R_k for the elements in R with coefficients in $\{0, \ldots, k-1\}$ (reduced modulo k) and we can identify $y_0 + y_1 x + \ldots + y_{n-1} x^{n-1} \in R_d$ with $(y_0, \ldots, y_{n-1}) \in \mathbb{Z}_d^n$ since we identify R_d with $\mathbb{Z}_d^n$. Then Ajtai's function $\mathbb{Z}_d^m \rightarrow \mathbb{Z}_q^n$ with $m \gg n$ is used as a guiding principle to create a function

$$f_{(a_0,\ldots,a_{m/n-1})} : R_d^{m/n} \rightarrow R_q, (y_0, \ldots, y_{m/n-1}) \mapsto \sum_{i=0}^{m/n-1} a_i y_i \mod q \in R_q$$

for randomly chosen elements $a_0, \ldots, a_{m/n-1} \in R_q$.
Is this new function a special case of Ajtai and what is the size for its representation $a_0, \ldots, a_{m/n-1}$?

Let us discuss the representation size first: The original function has a representation size of n^2 bit. An element $a \in R_q$ has a length of $n \log_2(q)$ bit, such that the total length of $a_0, \ldots, a_{m/n-1}$ is $m \log_2(q)$ and for $m \approx 2n \log_2(n)$ we thus have $4n(\log_2(n))^2$.
Now, let us consider whether the new function a special case of Ajtai's function: Since $R_q \to R_q, y \mapsto ay \mod q$ is a linear map (you can prove this easily for yourself) and linear maps of a vector space into itself are matrix multiplications, we conclude that the function $f_{(a_0,\ldots,a_{m/n-1})} : R_d^{m/n} \to R_q$ with $y \mapsto \sum_{i=0}^{m/n-1} a_i y_i$ for elements $a_i \in R_q$, $0 \leq i \leq m/n - 1$, also is a linear map which can be represented by a matrix.
Thus, we can say that the function $f_{(a_0,\ldots,a_{m/n-1})}$ is a special case of Ajtai's function with better representation.

10.1. Definition of Ideal Lattices

Finding collisions of $f_{(a_0,\ldots,a_{m/n-1})}$ leads to short vectors in $\Lambda_q^{\perp}(A)$ (with A being determined in the exercise). This is the average case problem (with the random lattice). What is the corresponding worst case problem to this average case (lattice) problem? The answer is: Finding short vectors in ideal lattices.
Let $I \subseteq R$ be an ideal. Then the set

$$L := \{(y_0, \ldots, y_{n-1}) \in \mathbb{Z}^n : y_0 + y_1 x + \ldots + y_{n-1} x^{n-1} \in I\}$$

is a lattice, called *ideal lattice*, since it obviously is a discrete additive subgroup of $\mathbb{Z}^n$.
The function $f_{(a_0,\ldots,a_{m/n-1})}$ is collision-resistant as long as finding short vectors in L is difficult.
(Interesting research question: How does the security of L relate to the algorithmic problems of number theory?)

11. Lattice-based signatures

Now we give a generic construction of lattice-based signature schemes and then we will see two instantiations, namely the GGH scheme and the LM-OTS.
For the generic case, we use a hash-function $h : \{0,1\}^* \to \mathbb{Z}^n$ to create a document's hash $h(d)$ in $\mathbb{Z}^n$ such that we can use the CVP for the signing process. Therefore we have to consider a lattice L. For this lattice we have a "bad" lattice basis on one hand which forms the public key, and on the other hand we have a good (reduced) basis which forms the secret key, and by reducedness enables us to solve the CVP. Thus, the signing process is finding the lattice point s which is closest to the hash $h(d)$ of the document d. (Since we have a reduced lattice basis as our secret key, we can easily find the point s.) The verification first checks whether s is a lattice point by using the bad lattice basis, and then it verifies if s is close(st) to $h(d)$.
This is the generic construction of lattice-based signature schemes. There were some instantiations and research topics on lattice-based signature schemes:

- instantiation of Goldreich, Goldwasser and Halevi (1997)

- NTRUSign (2003)
- OTS of Lyubashevsky and Micciancio (2008)
- instantiation of Gentry, Peikert and Vaikuntanathan (2008)

To make it more precise, we now come to the first instantiation, the one of Goldreich, Goldwasser and Halevi (GGH).

11.1. First instantiation of Goldreich, Golwasser and Halevi (GGH)

The fundamental of this instantiation is the *Hermite Normal Form (HNF)*. Let us recall what HNF means: A matrix B is in Hermite Normal Form (HNF) if

1. B is an upper triangular matrix.
2. $0 < b_{ii}$ for all $1 \leq i \leq n$, and
3. $0 \leq b_{ij} < b_{ii}$ for all $1 \leq i < j \leq n$.

The instantiation of GGH uses the HNF of the reduced basis of full rank as a "bad" lattice basis.
But: Why is this indeed a "bad" lattice basis?
Suppose the HNF is not the worst representation of the lattice. Then there exists an algorithm which attacks the signature scheme by taking advantage of the HNF, because every basis can be transformed into HNF in polynomial time of the input length. We see that an advantage exists for every representation. Thus, the HNF is the worst representation of a lattice.
The setup of GGH is:

KeyGen: As secret key choose a randomly selected basis $B \subseteq \mathbb{Z}^n$ of short vectors and for the public key compute $A = \mathrm{HNF}(B)$ which then consists of much longer vectors.

Sign: To sign a document d, compute its hash $h(d)$ and solve the CVP for $h(d)$ using the reduced lattice basis B. Return the signature $s = \mathrm{CVP}_B(h(d))$.

Verify: Check whether $s \in L(A)$ and if $||s - h(d)||$ is small.

11.2. Cryptanalysis of GGH by Nguyen and Regev

The problem with this approach is the following: If we find the translated parallelepiped, we can shift it to the parallelepiped of the good basis and know the secret.
This is due to the fact that each signature yields a point in the translated parallelepiped $\mathcal{P}_{1/2}$ and lots of points inside let us get an impression of the shape of the parallelepiped. Consider Figure 16.
From hashes $h(d_i)$ (of documents d_i) and their signatures s_i we can build the vectors $h(d_i) - s_i$, such that we obtain something like in Figure 16 c) if we use about 200.000 documents at all. This gives us the chance to reconstruct the shape of the parallelepiped

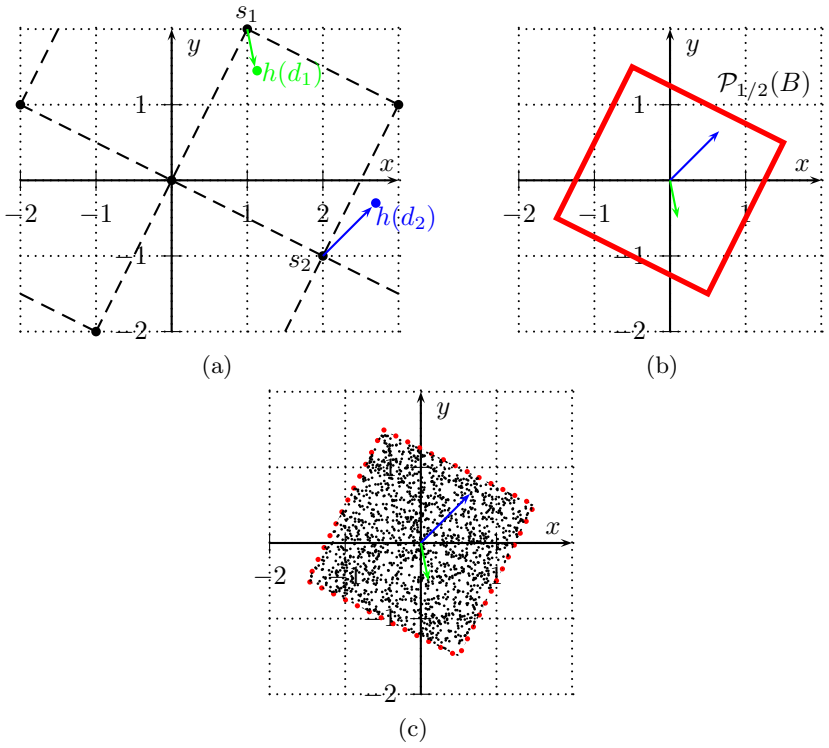

Figure 16: Picture a) is the lattice with two signatures and picture b) is the translated parallelepiped $\mathcal{P}_{1/2}(B)$ of the good basis. We (as an adversary) only know the points in picture c).

and thus the shape of a good basis. (In comparison to GGH where 200.000 documents are sufficient, for NTRU we only need about 300 messages.)
(For completeness, it is to remark that the GGH scheme is not really used in practice, but it is more a theoretical construction for an instantiation.)

11.3. Second instantiation of Lyubashevsky and Micciancio (LM-OTS)

As mentioned in the history above, we want to consider another instantiation of lattice-based signature schemes, which is the one-time signature scheme of Lyubashevsky and Micciancio of 2008.
Therefore, recall the definition of $R := \mathbb{Z}[x]/\langle f \rangle$ for some monic and irreducible polynomial f of degree n, as well as the definition of R_d and R_q (where $d = 2$ typically).
As we want to use the improved version of Ajtai's hash-function $\mathbb{Z}_d^m \to \mathbb{Z}_q^n$, we identify $\mathbb{Z}_d^m$ with $R_d^{m/n}$ and $\mathbb{Z}_q^n$ with R_q and for fixed $a_0, \ldots, a_{m/n-1}$ use the function

$$\begin{array}{rccl} h = h_{(a_0,\ldots,a_{m/n-1})}: & R_d^{m/n} & \to & R_q \\ & (y_0, \ldots, y_{m/n-1}) & \mapsto & \sum_{i=0}^{m/n-1} a_i y_i \mod q = h(y), \end{array}$$

which is linear, i.e. $h(y + y') = h(y) + h(y')$, and for all "small" $D \in R$ we have $h(Dy) = Dh(y) \pmod q$.
Now the setup for the LM-OTS is the following:

KeyGen Randomly select two very small elements $\widetilde{x}, \widetilde{y} \in \mathbb{Z}_d^m$ and translate them into elements $x, y \in R_d^{m/n}$. The secret key thus is (x, y) and the public key is $(X, Y) = (h(x), h(y))$.

Sign Identify a message $\widetilde{D} \in \mathbb{Z}^n$ with $D \in R$ (D has to be small such that we can use the linearity of the multiplication). Now, the signature of D is $s = xD + y$.

Verify To verify a given signature s, check whether $DX + Y = h(s) \pmod q$.

11.4. The security of the LM-OTS

We want to discuss the security of the LM-OTS. Therefore we want to use a forger who is given h, X, Y and ask ourselves whether the forger can obtain a signature s for a message z of its choice?
(The existence of such a forger implies the ability to calculate short vectors in lattices of dimension n in the worst case.)
Assume, the forger forges the signature s' of some message z' with $(s, z) \neq (s', z')$. We know:

1. Since h is a hash-function, there are many $x', y' \in \mathbb{Z}_d^m$ such that $h(x') = h(x)$ and $h(y') = h(y)$. (This is an analogous argument as in the proof of the security of the one-time pad.)

2. The forger has the following information: $(h, h(x), h(y), s, z)$. This information yields negligible information about x and y (information theoretical argument).

3. The forger produces the signature s' which is not the same as $s = xz + y$ with very high probability because the forger has no knowledge of x and y.

4. But the verification works: We have $h(s') = z'X + Y = z'h(x) + h(y) = h(z'x + y)$, but since $s' \neq z'x + y$ we found a collision $(s', z'x + y)$ for the hash-function h (by which we can find short vectors in ideal lattices).

If we combine the LM-OTS with Merkle, we obtain a signature scheme LM-Merkle based on lattices. In comparison to the LD-OTS that has a signature length which is quadratic in the input length n, the LM-Merkle scheme has a signature length of about $n\text{poly}(\log_2 n)$ which is much better.

Part III.
Code-based Cryptography

12. Motivation

Coding theory was created by Shannon in 1948 and its goal is to find and correct errors that occur during transmission of messages through some sort of "noisy" channel:

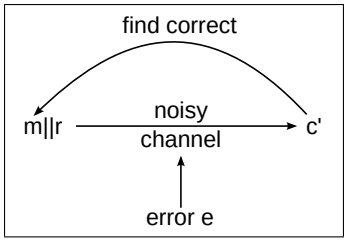

Figure 17: Find correct message m for noised code c'

An application of coding theory you can find in CD- and DVD-ROMs in which one wants to assure oneself the integrity of the data copied/read from the disc(s).
Now, the question is if and how we can use coding theory in cryptography and which benefits and drawbacks it has.
Most advantages are obvious: Coding theory is very fast and easy to implement. And since it is based on a well-known hard problem it remains secure in a post-quantum world of cryptography.
The disadvantages are not that obvious at present, but we will discover them throughout this paper.

13. Definitions

At the beginning, we have to define several things such that we have a common language. (Note: Throughout this section all vectors are denoted as row vectors.)
The *Hamming distance* $d(x, y)$ for two words $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in \mathbb{F}_q^n$ is defined as

$$d(x, y) := |\{i : x_i \neq y_i\}|$$

the number of entries in which x differs from y.
The *Hamming weight* $wt(x)$ of a word $x \in \mathbb{F}_q^n$ is the number of non-zero values/entries of x, $wt(x) := |\{i : x_i \neq 0\}|$, which can also be written as $wt(x) = d(x, 0)$ with 0 being the null-vector $(0, \ldots, 0)$.
A *generator matrix* is a $(k \times n)$-matrix G which defines a *vectorial subspace* C of dimension k in $\mathbb{F}_q^n$, i.e. $C = \{xG : x \in \mathbb{F}_q^k\}$, or—equivalently— G is a matrix whose rows form

a basis for the vector space C. (The subspace C is called a *linear* (n,k)*-code* or short (n,k)*-code*.)
An example of a generator matrix G would be $G = (I_k, A_{k\times(n-k)})$ where I_k denotes the $k \times k$ identity matrix and A is a $k \times (n-k)$ matrix. A code c for a message m is given by $c = mG$. Now, if the generator matrix was a random matrix, we would not be able to know where entries of the message are.
We call $R = k/n$ the *rate* of the code and want to achieve $R \approx 1$.
The *encoding algorithm* simply works as follows: A message m is mapped via f_G to its code c:

$$f_G : \mathbb{F}_q^k \to \mathbb{F}_q^n : m \mapsto mG = c.$$

The *decoding algorithm* for a (valid) code c returns the corresponding message m:

$$\gamma_G : \mathbb{F}_q^n \to \mathbb{F}_q^k : c = mG \mapsto m.$$

The decoding algorithm *corrects* t errors iff for all errors $e \in \mathbb{F}_q^n$ and for all messages $m \in \mathbb{F}_q^k$ the implication

$$wt(e) \leq t \implies \gamma_G(mG + e) = m$$

holds. (Let $c' := mG + e$ denote the noised code.)
Some methods for decoding would be

- exhaustive search of errors: By this first naive way one chooses e of weight smaller than or equal to t, computes $c' - e$ and checks if the result is in the code C.
- exhaustive search of code words: This second naive way computes $e = c' - mG$ for all possible m and checks e's weight.

Note that both methods are exponential.
We will introduce some other important variables.
The *minimal distance* d of an (n,k)-code C is the minimal Hamming distance between two different code words, i.e. $d = \min\{d(x,y) : x, y \in C, x \neq y\}$, or—equivalently— the minimal weight of non-zero code words: $d = \min\{wt(x) : x \in C, x \neq 0\}$.
We call C an (n,k,d)*-code* iff n is the length of the code words, k is the dimension of the subspace $C \subset \mathbb{F}_q^n$ and d is the minimal distance.
The *correction capacity* t of an (n,k,d)-code C is $t = \lfloor (d-1)/2 \rfloor$. This becomes clear by taking a look at the following Figure (18), since each ball circled around a code word x covers those noised codes of x that can be corrected, such that we obtain the unique solution x for a corrupted x'.
The *Singleton Bound* for an (n,k,d)-code C is $n \geq k+d-1$ or $n-k+1 \geq d$ which can be visually clarified by the use of a $(k \times n)$ generator matrix G of the shape $(I_k, A_{k\times(n-k)})$.
Let $A_q(n,d)$ be the maximum value of M such that there exists a (not necessarily linear) code over $\mathbb{F}_q$ of length n with M codewords and minimum distance d. With this, the *Gilbert Varshamov Bound* is $A_q(n,d) \sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i \geq q^n$.
The usual *scalar product* is $(x,y) \mapsto x.y = \sum_{i=1}^{n} x_i y_i$.

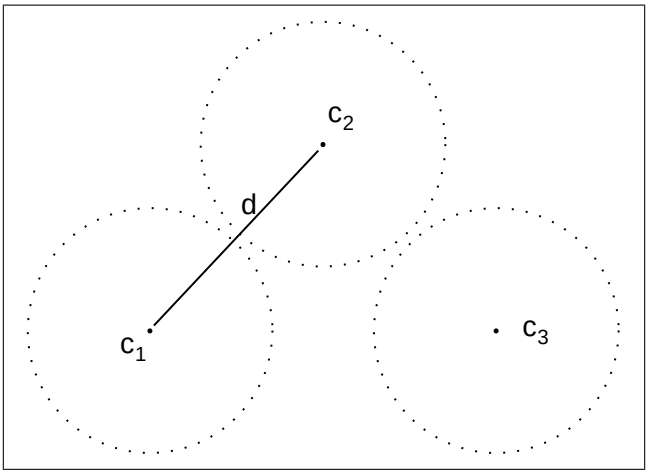

Figure 18: Unique solution

With this, the *dual code* $C^\perp$ of the code C is defined via $C^\perp := \{z \in \mathbb{F}_q^n : \forall c \in C : x.z = 0\}$ and we have $\dim\left(C^\perp\right) + \dim(C) = n$.
The *parity check matrix* (or *generator matrix of* $C^\perp$) of an (n,k)-code C is the $(n-k)\times n$ matrix which defines the dual code $C^\perp$ or—equivalently— whose rows form a basis of the orthogonal complement of the subspace C. It holds that $Hc^T = 0$ for all $c \in C$.
The *syndrome* of a word $x \in \mathbb{F}_q^n$ is the vector $s \in \mathbb{F}_q^{n-k}$ defined by $s = Hx^T$ where H is the parity check matrix of the (n,k)-code C. It is hard to find x for given H, s and $wt(x)$ (this is known as $\mathcal{NP}$-complete).
The *syndrome decoding algorithm* is an application $\gamma_H : \mathbb{F}_q^{n-k} \to \mathbb{F}_q^n$ which is able to correct t errors iff for all $e \in \mathbb{F}_q^n$ it holds

$$wt(e) \leq t \implies \gamma_H(He^T) = e.$$

14. Application in Cryptography

Now, we want to consider cryptographic schemes that are based on error-correcting codes.

14.1. McEliece

The McEliece scheme was the first code-based cryptosystem invented in 1978.
We now describe the KeyGen- as well as the Encrypt- and Decrypt-steps of the McEliece scheme. (The small s denotes the security parameter.)

KeyGen(1^s) 1. Choose n, k and t according to s.

2. Pick a random generator matrix G_0 of an $(n,k,2t+1)$-binary code (these are also called *binary Goppa-codes*).

3. Choose a random permutation P ($n \times n$ matrix over $\mathbb{F}_2$).
4. Pick a random invertible $k \times k$ matrix S (over $\mathbb{F}_2$).
5. Compute $G = SG_0P$.

Then the public key is $pk = (G, t)$ and the secret key $sk = (S, G_0, P, \gamma_{G_0})$ where γ_{G_0} is the decoding algorithm.

Encrypt(pk, m) The encryption algorithm for a message $m \in \mathbb{F}_2^k$ using the public key pk.

1. Randomly pick e of weight t.
2. Compute $c = mG + e$.

Output c.

Decrypt(sk, c) The decryption algorithm for a ciphertext $c \in \mathbb{F}_2^n$ that is decrypted using the secret key sk.

1. Compute $cP^{-1} =: z = (mG + e)P^{-1} = mSG_0 + eP^{-1}$, where $wt(eP^{-1}) = wt(e) = t$ since P is a permutation.
2. Compute $\gamma_{G_0}(z) = mS =: y$.
3. Compute $yS^{-1} = m$.

14.2. Niederreiter

The Niederreiter encryption scheme is another code-based scheme invented in 1986. The difference to the McEliece scheme is the description of the codes: The Niederreiter scheme uses the parity check matrix, whereas the McEliece scheme uses a generator matrix. Here are the three steps of the Niederreiter scheme, with s being the security parameter as above:

KeyGen(1^s) 1. Choose n, k and t according to the security parameter s.
2. Pick H_0 at random as the parity check matrix for an $(n, k, 2t + 1)$-binary Goppa code.
3. Choose P as a random permutation ($n \times n$ matrix).
4. Randomly pick S as an invertible $(n - k \times n - k)$ matrix.
5. Compute $H = SH_0P$.

Output as public key $pk = (H, t)$ and as secret key $sk = (S, H_0, P, \gamma_{H_0})$, where γ_{H_0} is the decoding algorithm.

Encrypt(pk, m) The encryption algorithm to encrypt a message $m \in W_{2,n,t}$ using the public key pk where $W_{2,n,t} := \{x \in \mathbb{F}_2^n : wt(x) = t\}$:

Compute $c = Hm^T$ and output c.

Decrypt(sk, c) The decryption algorithm to decrypt a ciphertext $c \in \mathbb{F}_2^{n-k}$ using the secret key sk.

1. Compute $S^{-1}c = S^{-1}SH_0Pm^T = H_0Pm^T =: z$ where $wt(Pm^T) = wt(m) = t$ (same argument as in McEliece).
2. Compute $\gamma_{H_0}(z) = Pm^T =: y$.
3. Compute $P^{-1}y = m^T$.

14.2.1. Goppa codes

Goppa codes are linear codes of the form $(2^m, 2^m - mt, 2t + 1)$ for some m and t.
The size of the public key is $2^m(2^m - mt)$.
Here is a small list of Goppa codes related to their security and the size of the used public key:

Code	security	size of pk in bits
$(1024, 524, 2 \cdot 50 + 1)$	2^{64}	535,576
$(2048, 1025, 2 \cdot 93 + 1)$	2^{106}	2,099,200
$(4096, 2056, 2 \cdot 170 + 1)$	2^{136}	8,421,376

15. Security Hypothesis

1. Hard to find G_0 from G. *(structural attacks)*
2. Hard to invert the decoding algorithm. *(decoding attacks)*

1. Complexity of best known attack (for McEliece with binary Goppa code) is $\mathcal{O}(2^{\dim(C \cap C^\perp)})$.
2. The *ISD algorithm* does the following: It takes $n - k$ random columns of H and puts them on the left. The same is done for the corresponding variables of x (since we have $s = Hx^T$). Then H is transformed into echelon form using Gaussian elimination by which we can then read off the first $n-k$ entries of x. The complexity is $\mathcal{O}\left(\frac{\binom{n-t}{n-k}}{\binom{n}{n-k}}\right)$.

16. Code-based Signatures

We will give an overview over the results on code-based signature schemes in which we will cover how to

- create signatures using codes
- establish an identity-based identification and signature scheme using error-correcting codes

16.1. Signatures with Codes

Above, we discussed public-key cryptosystems based on error-correcting codes. Unfortunately, we cannot directly derive a signature scheme by using a public-key cryptosystem that is based on codes like the McEliece or the Niederreiter scheme. (In contrast, with RSA it is possible to establish a signature scheme from a cryptosystem.)
The problem is that the McEliece and the Niederreiter scheme are not invertible: If we take $y \in \mathbb{F}_2^n$ at random and an (n, k, d)-code C for which we can correct at most t errors, it is improbable to decode y. The solution is that the hash-value has to be decodable which directly leads to

16.2. The CFS Signature Scheme

(The CFS signature scheme is named by N. Courtois, M. Finiasz and N. Sendrier.)
Let m be the message we want to sign, h a hash-function with values in $\{0,1\}^{n-k}$ and γ_H a syndrome decoding algorithm.
We search $s \in \mathbb{F}_2^n$ of given weight t with $h(h(m)||i_0) = Hs^T$ for parity check matrix H and some index i_0.
Therefore, we do the following:

1. Set $i := 0$.
2. As long as $h(h(m)||i)$ is not decodable, increase i.
3. Compute $s = \gamma_H(h(h(m)||i))$.

(In our context here, $||$ denotes concatenation.)
Then, the signer sends the signature (s, j) with $h(h(m)||j) = Hs^T$. An example for a signature creation you can see in Figure 19.

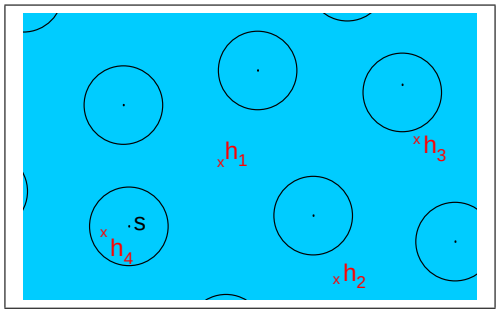

Figure 19: Increase counter and re-hash until it is decodable. In this example we have $(s, 4)$ as signature for the message m.

Since we see that it is almost impossible to decode, we need a dense family of codes: Therefore we use Goppa codes $((2^m, 2^m - tm, 2t+1)$-codes) with small t such that the probability for a random element to be decodable is about $1/t!$. As an instantiation, we

take $n = 2^m$, $m = 16$ and $t = 9$, such that we have a chance of 1 to 362880 to have a decodable word.
The parameters of the CFS signature scheme are as follows:

Signature costs:	$t!t^2m^3$	$12 \cdot 10^{10}$ operations
Signature length:	$(t-1)m + \lfloor \log_2 t \rfloor$	131 bits
Verification costs:	t^2m	1296 operations
Size of pk:	$tm2^m$	9 Mbits
Cost of ISD attack:	$2^{tm}(1/2 + o(1))$	$\approx 2^{80}$

As we see, there are several cons for the CFS scheme: We have to decode a lot of words ($t!$) before we find a good/decodable one and a small parameter t leads to very big parameters such as the huge size of the public key pk of 9 Mbits.
Recently, there was proposed a General Birthday-like attack against this scheme which led to the choice of new the parameters $m = 19$ and $t = 11$.
There is also work in progress for this scheme, which means that people try to find a dense family of compact codes (*QD Goppa codes*), improve the decoding algorithm (*Patterson algorithm for QD Goppa codes*) and use quantum algorithm to decrease the decoding time (like *Grover's algorithm*).

16.3. Stern Identification Scheme

The Stern Identification Scheme was presented by J. Stern in 1993. It is a zero-knowledge scheme and its security is based on the syndrome-decoding problem.
The protocol in general is like this:

- Generate a random matrix H of size $(n-k) \times n$.
- Choose an integer t that represents the weight.
- The public key is $pk = (H, t)$.
- Each user receives its secret key s of weight t and n bits.
- Each user computes $i = Hs^T$ which is done once and the syndrome i is public.

Now A wants to prove to B that it knows the secret, but it does not want to divulgate it. The protocol for this is on r rounds and each of them is defined as follows:

1. A chooses y of n bits randomly and also a permutation σ of $\{1, \ldots, n\}$, and computes
$$c_1 = h(\sigma || Hy^T), \qquad c_2 = h(\sigma(y)), \qquad c_3 = h(\sigma(y \oplus s))$$
which it sends to B.
2. B sends back a random $b \in \{0, 1, 2\}$.
3. Now, there are 3 possibilities:

a) If $b = 0$, A reveals y and σ.

b) If $b = 1$, then A reveals $y \oplus s$ and σ.

c) Otherwise, A reveals $\sigma(y)$ and $\sigma(s)$.

4. For the verification, there are naturally 3 possibilities:

a) If $b = 0$, B checks that c_1 and c_2 are correct.

b) If $b = 1$, then B checks if c_1 and c_3 are correct. (Notice, that for c_1 we have $Hy^T = H(y \oplus s)^T \oplus i$ and i is public.)

c) Otherwise, B checks if c_2 and c_3 are correct and that $\sigma(s)$ is of weight t.

Analyzing the Stern scheme, one obtains that in each round the probability to cheat is 2/3, such that we need 150 rounds to gain a security of 2^{80}. (The norm ISO/IEC-9798-5 proposes the two probabilities 2^{-16} and 2^{-32} for which one needs 28 resp. 56 rounds.)
A dual construction to the Stern identification scheme is the Véron Identification Scheme, which we will not present here since it it very similar to the construction of Stern.
We yet have another problem: The publicly known random matrix H is too big. Therefore, we can replace it by the parity check matrix of a certain family of codes, the so-called *double-circulant codes*: Let l be an integer. A random *double-circulant matrix* H of size $l \times 2l$ is defined as $H = (I_l, A)$ where A is a cyclic matrix of the form

$$A = \begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_l \\ a_l & a_1 & a_2 & \cdots & a_{l-1} \\ \vdots & \vdots & \vdots & & \vdots \\ a_2 & a_3 & a_4 & \cdots & a_1 \end{pmatrix}$$

and $a = (a_1, \ldots, a_l)$ is a random vector of $\mathbb{F}_2^l$.
For the storage of H we need l bits since we only store the vector a.
Double-circulant codes have the following properties: The minimum distance is the same as for random matrices, the syndrome-decoding is still hard and they are very interesting for the implementation in low-resource devices.
The parameter sizes are the following: For $n = 2l$ the private data which consists of the secret s has bit-length n and the public data too, since i is of size $n/2$ as well as a.
This means that we need to choose $l = 347$ and $t = 74$ for a security of 2^{85}, in which the public and secret key have a size of 694 bits.
On this field there also is work in progress, such as improving the ISD algorithm for quasi-cyclic and/or quasi-dyadic codes, improving the success probability, using this constructions with lattice problems and studying the fault injection sensitivity of this scheme.

17. Identity-based Identification and Signature Scheme using Error-Correcting Codes

Public key cryptography lacks the management of the authenticity of the public keys. In 1984 Shamir introduced the identity-based public-key cryptography (ID-PKC) to

simplify the management and the authenticity of the public key: The public key is linked to the user using its email-address, name, etc. So verification becomes easy since one only has to use the name of the person who sends a signature to verify it. The problem with this approach is that one has to secretly determine the secret key sk out of the public key pk (which is the name). Because of the secrecy of the derivation process an authority has to be involved which does this using some extra secret.
In the next Figure 20 you can see Alice, Bob and an authority, in which Alice achieves her private key of the authority and Bob uses Alice's public key to cipher and send a message to her.

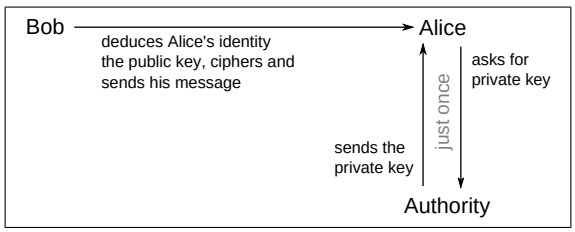

Figure 20: Alice and Bob using ID-PKC

The advantages of the identity-based signature scheme are obvious: No certificate for Alice's public key is needed, the key itself can be easily memorized and –because of this– no directory has to be used to store it. The main drawbacks of this approach are that the authority is very powerful (key escrow) and the distribution of the keys to the endusers is not trivial (a secure channel is needed).
In spite of everything, there are several identity-based signature schemes (IBS) like the ones of A. Shamir (1984), F. Hess (2002), F. Zhang and K. Kim (2002) or Cha and Cheon (2003) as well as identity-based encryption schemes like the ones of D. Boneh and M. Franklin (2001), C. Cocks (2001), D. Boneh and X. Boyen (2004) or Baek and Zheng (2004).
The main idea of identity-based signature schemes using codes is the following: We use the CFS signature scheme to have to notion of identity-based because we use the identity to compute the syndrome and so we obtain the secret key (s, j). Furthermore, we use the Stern scheme as a classical identification scheme for which we use the matrix H' of the first part and a classical Stern scheme (except the knowledge of j).
A detailed description is given now: Let C be a linear $(n, k, 2t + 1)$-code which corrects up to t errors, H be a parity check matrix of C (which is private and owned by the authority), $H' = SHP$ be public with invertible matrix S and permutation matrix P, h a hash-function with values in $\{0, 1\}^{n-k}$ and id_A Alice's identity which also is public.
Alice (the prover) wants to identify herself to Bob (the verifier). As already shown above, the protocol is splitted into two parts:

1. The authority gives to Alice her secret key which it derived from Alice's identity (public).

2. Alice identifies herself to Bob.

For the key deliverance, we search a mean to find s such that $h(\mathrm{id}_A) = H's^T$. The problem arising hereby is that $h(\mathrm{id}_A)$ in principle is not in the space of decodable elements of $\mathbb{F}_2^n$. We can solve this problem by using the following variant of the algorithm of CFS (in which γ is the decoding algorithm for the hidden code):

1. Set $i := 0$.
2. While $h(h(\mathrm{id}_A)||i)$ is not decodable, increase i.
3. Compute $s = \gamma(h(h(\mathrm{id}_A)||i))$.

We obtain a couple (s, j) such that $h(h(\mathrm{id}_A)||j) = H's^T$. This couple is Alice's secret key which is transmitted by a secure channel from the authority to Alice.
Now Alice wants to identify herself to Bob. The matrix H', that will be used, is the same as the one used in the first part. (Remark: Although we give an identification scheme here, we can derivate a signature scheme by classical constructions.) Now the protocol is given by:

1. Alice chooses randomly a word y of n bits and a permutation σ of $\{1, \ldots, n\}$, and computes

$$c_1 = h(\sigma||H'y^T), \qquad c_2 = h(\sigma(y)), \qquad c_3 = h(\sigma(y \oplus s))$$

   which it sends to Bob.
2. Bob sends back a random $b \in \{0, 1, 2\}$.
3. Now, there are 3 possibilities:
   a) If $b = 0$, Alice reveals y and σ.
   b) If $b = 1$, then she reveals $y \oplus s$ and σ.
   c) Otherwise, Alice reveals $\sigma(y)$ and $\sigma(s)$.
4. For the verification, there are naturally 3 possibilities:
   a) If $b = 0$, Bob checks that c_1 and c_2 are correct.
   b) If $b = 1$, then he checks if c_1 and c_3 are correct. (Notice again, that for c_1 we have $H'y^T = H'(y \oplus s)^T \oplus i$ and i is public.)
   c) Otherwise, Bob checks if c_2 and c_3 are correct and that $\sigma(s)$ is of weight t.

This, in fact, is the classical Stern-protocol again.)
The security of the identification scheme relies to the choice of the parameters of the CFS scheme since it is used to create the private key and the matrix H' is reused in Stern's protocol. Also, there are two imperative conditions the scheme has to respect: The computation of (s, j) has to be difficult without the knowledge of the description of H' and the number of tries to determine j shall not be too important in order to reduce the costs of the computation of s.

Conversely to the original Stern scheme, the code used here has length 2^{16} instead of 2^9, which increases the communication costs. Here are some practical values for an identity-based identification scheme with $m = 16$ and $t = 9$:

pk	sk	matrix	communication	signature	KeyGen
tm	tm	$2^m tm$	$\approx 2^m \cdot$ #rounds	$\approx 2^m \cdot$ #rounds	
144 bits	144 bits	9 Mbits	$\approx$ 500 Kbits (58 rounds)	$\approx$ 1.5 Mbits (150 rounds)	1s

Current work in progress are identity-based encryption schemes, dual constructions (like the Véron identification scheme), QD constructions and the proof of the security in the random oracle model (ROM).

18. Summary

In this section we will summarize and end up the topic "code-based cryptography".
At first, we will give some "reminders": Encoding is a function $f_G : \mathbb{F}_q^k \to \mathbb{F}_q^n, m \mapsto mG$ for a generator matrix G of a code C. Decoding is a function $\gamma_G : \mathbb{F}_q^n \to \mathbb{F}_q^k, mG \mapsto m$.
The decoding algorithm can correct t errors iff for all $e \in \mathbb{F}_q^n$ and for all $m \in \mathbb{F}_q^k$ we have $wt(e) \leq t \implies \gamma_G(mG \oplus e) = m$.
Let H be a parity check matrix of a code C. The syndrome of a word $x \in \mathbb{F}_q^n$ is the vector $s \in \mathbb{F}_q^{n-k}$ such that $s = Hx^T$. The syndrome decoding algorithm thus is the function $\gamma_H : \mathbb{F}_q^{n-k} \to \mathbb{F}_q^n$.
The syndrome decoding algorithm can correct (up to) t errors iff $wt(e) \leq t \implies \gamma_H(He^T) = e$.
With this, we saw two cryptosystems based on codes: McEliece and Niederreiter. We give a short description of both schemes:
McEliece

Key Generation: We use Goppa codes and let G_0 be a generator matrix, P a permutation matrix and S an invertible matrix. Then we have for the public key $G = SG_0P$ and t, and the secret key is S, G_0, P.

Encrypt: Pick a random element e of weight t. Compute $c = mG \oplus e$ and output the ciphertext c.

Decrypt: Compute $z = cP^{-1}$, $y = \gamma_{G_0}(z)$ and $m = yS^{-1}$.

Niederreiter

Key Generation: We use a Goppa code with parity check matrix H_0. Let S be an invertible and P a permutation matrix. The secret key is S, H_0, P and the public key consists of t and $H = SH_0P$.

Encrypt: W.l.o.g. the message m is from $W_{2,n,t}$ which was defined one lecture ago. Compute $Hm^T = c$ as ciphertext. Output c.

Decrypt: Compute $z = S^{-1}c$, $y = \gamma_{H_0}(z)$ and $m = yP^{-1}$.

There are two kinds of attacks against both schemes:

1. Recover G_0 from G.

2. Inverting the decoding algorithm.

The best known attack to 1) is due to N. Sendrier and called "*Support Splitting Algorithm*" which has a complexity of $\mathcal{O}(2^{\dim(C\cap C^{\perp})})$. Against 2), the best known attack is "*Information Set Decoding*" (ISD) which randomly chooses $n-k$ coordinates of x, puts all corresponding columns of H to the left, does Gaussian elimination on the resulting matrix to obtain something like (I_{n-k}, A) and on this way tries to find the values for x. Next, we had the CFS signature scheme and the Stern identification scheme which is zero-knowledge. For the last, let H be a $(n-k)\times n$, randomly chosen binary matrix which satisfies the Gilbert-Varshamov-Bound $2^{n-k} \geq \sum_{i=0}^{d-2}\binom{n-1}{i} = t$. Also choose s with weight t and $Hs^T = y$. As private key we have $s \in \mathbb{F}_2^n$ of $wt(s) = t$, and as public key we have H, y, t and a hash-function h. Now we describe the 3 passes of the Stern protocol: At first, the prover chooses $u \in_R \mathbb{F}_2^n$ and a permutation σ in the set of all permutations of $\{1,\ldots,n\}$ uniformly at random. Then it computes $c_1 = h(\sigma||Hu^T)$, $c_2 = h(\sigma(u))$ and $c_3 = h(\sigma(u\oplus s))$ which it sends to the verifier. The verifier chooses a challenge $b \in_R \{0,1,2\}$ uniformly at random and sends it to the prover. Now the prover has to respond correctly to the challenge: If $b=0$, the prover reveals u and σ for which the verifier proofs if c_1 and c_2 are correct. If the challenge is $b=1$, the prover reveals $u\oplus s$ and σ for which the verifier proofs the correctness of c_1 and c_3. If otherwise $b=2$, then the prover sends $\sigma(u)$ and $\sigma(s)$ to the verifier which then proofs c_2 and c_3 to be correct.
There are several ways to cheat in the Stern scheme:

1. The first would be that the cheater chooses u and σ uniformly at random and switches the original s to its choice s' of weight t. The cheater succeeds if $b=0$ or $b=2$, but it fails for $b=1$.

2. A second way would be to again choose u and σ at random and replacing s by s' of weight t and to change the computation of c_1 to $c_1 = h(\sigma||H(u\oplus s')^T \oplus y)$. Then the cheater succeeds for $b=1$ and $b=2$ and fails for $b=0$.

3. The last way to cheat is to choose s' such that $Hs'^T = y$, but $wt(s') \neq t$. Then the cheater will fail for $b=2$ because $\sigma(s')$ is not of weight t which it has to be, and it will succeed for the two other values.

Part IV.
Multivariate Cryptography

19. Introduction

Multivariate cryptography schemes are supposed to be very efficient in "light-weight" cryptography, e.g. in embedded devices such as microcontrollers. Multivariate cryptography is based on the difficulty of solving non-linear multivariate system of equations over finite fields.
As an example of such a system think of the following: Given $\mathbb{K} = \mathbb{F}_2$ and variables $x_1, x_2, x_3 \in \mathbb{K}$. Suppose we have the following equations and polynomials:

$$\begin{aligned} p_1(x_1, x_2, x_3) &= x_1x_2 + x_1x_3 \\ p_2(x_1, x_2, x_3) &= x_1 + x_2x_3 \\ p_3(x_1, x_2, x_3) &= x_1x_3 + 1. \end{aligned}$$

We write $P = (p_1, p_2, p_3)$ and $\underline{x} = (x_1, x_2, x_3)$. It is obvious that the equation system $P(\underline{x}) = 0$, i.e. $p_i(x_1, x_2, x_3) = 0$ for $1 \leq i \leq 3$, has the solution $\underline{x} = (1, 1, 1)$.
In general, the problem is $\mathcal{NP}$-complete/-hard, but we have to identify hard instances of this problem since the $\mathcal{NP}$-hardness is not enough for the difficulty of this scheme.

20. Notation and generalized scheme

20.1. General notation

Here are listed the basic notations used in this chapter:

- $\mathbb{K} := \mathbb{F}_q$ is the (finite) field of q elements where $q := p^e$ for p prime and $e \in \mathbb{N}$.
- By $m \in \mathbb{N}$ we denote the length of the ciphertext or length of the hash and $n \in \mathbb{N}$ is the length of the plaintext or the signature length.
- The vector $\underline{w} := (w_1, \ldots, w_n) \in \mathbb{K}^n$ stands for the variables of the plaintext and $\underline{z} := (z_1, \ldots, z_m)$ is the one belonging to the ciphertext.
- We have an affine map $S : \mathbb{K}^n \to \mathbb{K}^n : \underline{w} \mapsto \underline{x} := M_S\underline{w} + c_S$, where $M_S \in \mathbb{K}^{n\times n}$ is (a matrix) of full rank (i.e. it is regular) and $c_S \in \mathbb{K}^n$, and also another affine map $T : \mathbb{K}^m \to \mathbb{K}^m : \underline{y} \mapsto \underline{z} := M_T\underline{y} + c_T$ with a matrix $M_T \in \mathbb{K}^{m\times m}$ of full rank and $c_T \in \mathbb{K}^m$. The maps S and T hide the non-linear central map Q which is defined below.
- The map $Q : \mathbb{K}^n \to \mathbb{K}^m : \underline{x} \mapsto \underline{y} := Q(\underline{x}) := (q_1(\underline{x}), \ldots, q_m(\underline{x}))$ is the central map with (non-linear) quadratic polynomial functions $q_i(x)$ that are easy to invert. "Invert" means that for all images an inverse image can be computed.

The elements $q_i \in \mathbb{K}[x]$ are polynomials with coefficients in $\mathbb{K}$ and variables $x_1, \ldots, x_n$. Since every non-linear equation can be reduced to a quadratic equation, it is sufficient that the q_i are quadratic polynomials.

$$\begin{array}{ccccccc} & S & & Q & & T & \\ \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q^m & \longrightarrow & \mathbb{F}_q^m \\ \underline{w} & \mapsto & \underline{x} & \mapsto & \underline{y} & \mapsto & \underline{z} \\ & \text{(affine) linear,} & & \text{hidden,} & & \text{(affine) linear,} & \\ & \text{bijective} & & \text{quadratic} & & \text{bijective} & \end{array}$$

20.2. Generalized encryption and signature scheme

KeyGen As the public key we have q and $P := T \circ Q \circ S$. The secret key T, Q, S allows to "compute" P^{-1} what is pretty obvious since $P^{-1} = S^{-1} \circ$"Q^{-1}"$\circ T^{-1}$. The map P^{-1} allows us to calculate an inverse image for a given element in $\mathrm{Im}(P)$.

Encryption and Decryption As the ciphertext we take $P(\underline{w}) =: \underline{z}$, so the plaintext is $\underline{w} =$"$P^{-1}(\underline{z})$" (cf. Rabin function).

Signatures Given a hash-function $h : M \to \mathbb{K}^m$ that maps from the message space M to $\mathbb{K}^m$. The signature for a hash $h(\underline{m})$ of a message $\underline{m}$ is $\underline{w} \in P^{-1}(h(\underline{m}))$. For verification we check if $P(\underline{w}) = h(\underline{m})$ holds. So the signature is a solution of a multivariate system which is impossible to solve without the knowledge of the secret key.

21. How to create $\mathbb{F}_{2^n}$ – the finite field with 2^n elements

As a first example remind yourself of the finite field $\mathbb{F}_{2^2}$ of 4 elements and $\mathbb{F}_2^2$, the vector space of dimension 2 over $\mathbb{F}_2$. We have $\mathbb{F}_2^2 = \{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}$ with the addition-operation. $\mathbb{F}_{2^2}$ is constructed in the following way:

$$\mathbb{F}_{2^2} = \mathbb{F}_2[x]/\langle x^2 + x + 1\rangle$$

where x^2+x+1 is an irreducible element of $\mathbb{F}_2[x]$ over $\mathbb{F}_2$. This means that $x^2+x+1 = 0$, $x^2 = x + 1$, and so on. So, we get $\mathbb{F}_{2^2} = \{0, 1, x, x + 1\}$ and have addition and multiplication as field-operations. We now want to identify elements of $\mathbb{F}_{2^2}$ with those of $\mathbb{F}_2^2$. This we do by the following way: $0 = 0\cdot 1+0\cdot x$, $1 = 1\cdot 1+0\cdot x$, $x = 0\cdot 1+1\cdot x$ and $x+1 = 1\cdot 1+1\cdot x$ which means we identify those elements with the vectors $\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}$. In general, we can identify $\mathbb{F}_{2^n}$ with basis $1, \ldots, x^{n-1}$ as $\mathbb{F}_2^n$. We also know that $\mathbb{F}_2^n \setminus \{0\}$ is a multiplicative group with $2^n - 1$ elements which is cyclic.
So much to the construction of finite fields with 2^n elements.

22. Matsumoto-Imai (MI)

Matsumoto and Imai proposed another multivariate scheme which we will discuss now.

22.1. Construction of MI

We suppose $n = m$ for the whole section.
The basic idea is that

$$\mathbb{F}_q^m \simeq \mathbb{F}_{q^m} \cong \mathbb{F}_q[x]/\langle f\rangle = \{\underbrace{a_0 + a_1 x + \ldots + a_{m-1}x^{m-1}}_{\text{unique representatives}} + \langle f\rangle\}. \tag{MI1}$$

(Note that both $\mathbb{F}_q^m$ and $\mathbb{F}_{q^m}$ are vector spaces over $\mathbb{F}_q$.)
So because of (MI1) we can identify the residue class $a_0 + a_1 x + \ldots + a_{m-1}x^{m-1}$ with the vector $(a_0, \ldots, a_{m-1})$ (vector space isomorphism).

22.2. The quadratic map Q

As the quadratic map Q Matsumoto and Imai proposed

$$\begin{array}{rcl} C^* : \mathbb{F}_{q^m} & \to & \mathbb{F}_{q^m} \\ x & \mapsto & x^{1+q^\alpha} \end{array}$$

which is invertible if $\gcd(1 + q^\alpha, q^n - 1) = 1$ (cf. RSA). (A good choice would be $\alpha = 8$ and n appropriately.)
Now, the question is why this map is easy to invert. This is due to the fact that $C^{*-1}(y) = y^{(1+q^\alpha)^{-1} \bmod (q^n-1)}$.
The inverse can be computed by solving $h(1+q^\alpha) \equiv 1 \mod (q^n - 1)$ using the Extended Euclidean Algorithm. Since we have to assure $C^*(x)^h \stackrel{!}{=} x$, we have $h(1 + q^\alpha) = 1 + \lambda(q^n - 1)$, which indeed yields the equation $C^*(x)^h = x$:

$$C^*(x)^h = (x^{1+q^\alpha})^h = x^{h(1+q^\alpha)} = x^{1+\lambda(q^n-1)} = x(x^{q^n-1})^\lambda = x.$$

22.3. Inverting Q — difficult?

Patarin showed that the quadratic map Q is not difficult to invert. (Recall that inverting means to find a preimage x for a given image $Q(x)$.)
The proof is as follows: We will construct a bilinear map $F : \mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K}^n, (x, y) \mapsto F(x, y)$ with the following property: For all $x \in \mathbb{K}^n$ there is $F(x, Q(x)) = 0$. Note: $F = (F_1, \ldots, F_n)$ where F_i is a bilinear form.
For bilinear forms we have that for fixed x the map $F_i(x, \cdot) : \mathbb{K}^n \to \mathbb{K}^n, y \mapsto (x, y)$ is linear, i.e. $F(x, y_1 + y_2) = F(x, y_1) + F(x, y_2)$, and for fixed y the map $F(\cdot, y) : \mathbb{K}^n \to \mathbb{K}^n, x \mapsto (x, y)$ is also linear, i.e. $F(x_1 + x_2, y) = F(x_1, y) + F(x_2, y)$. In general, bilinear forms look like $\mathbb{K}^n \times \mathbb{K}^n \to \mathbb{K} : (x, y) \mapsto x^T A y$, e.g. $\mathbb{K}^2 \times \mathbb{K}^2 \to \mathbb{K}, (x, y) \mapsto F(x_1, x_2, y_1, y_2) = x_2 y_1 + x_1 y_2$.
Summa summarum, each component F_k can be written as

$$F_k(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^{(k)} x_i y_j.$$

The goal $(*)$ now is to find the plaintext w for a given ciphertext $z = P(w)$ where $P = T \circ Q \circ S$.

1. Since Q satisfies a bilinear relation $F(x, Q(x)) = 0$, it follows that P also follows a bilinear relation $F^*(w, P(w)) = 0$. For further readings suppose $F_k^* = \sum_{i=1}^n \sum_{j=1}^n a_{ij}^{(k)} x_i y_j$.

2. If F^* (the coefficients $a_{ij}^{(k)}$) is known, then the goal $(*)$ can be solved as follows: We know that $F^*(w, z) = 0$.Thus, since we know z, the system $F^*(w, z) = 0$ is a linear system in w:
$$\sum_{i=1}^{n} \sum_{j=1}^{n} w_i \underbrace{a_{ij}^{(k)} z_j}_{\text{known}} = 0.$$

3. The last step is to find F^* using a known-plaintext-attack: Each known pair $(w, z = P(w))$ satisfies $F^*(w, z) = 0$. So we get linear equations for the coefficients $a_{ij}^{(k)}$.

In the MI-scheme we had $C^* : \mathbb{F}_{q^n} \to \mathbb{F}_{q^n}, x \mapsto x^{q^\alpha+1}$ where $\mathbb{F}_{q^n}$ is a finite field of characteristic $q \in \mathbb{P}$.
Fact: The *Frobenius automorphism* $\mathbb{F}_{q^n} \to \mathbb{F}_{q^n}, x \mapsto x^q$ is a linear map. (The proof is done by yourself in Exercise 13, Task 2.)
From this we can conclude that the maps $x \mapsto x^{q^\alpha}$ and $x \mapsto x^{q^{2\alpha}}$ are linear, too (cf. Ex. 13, Task 2).
Thus, the bilinear relation F is given by

$$F(x, y) = x^{q^{2\alpha}} y - x y^{q^\alpha}.$$

(You can prove on your own, that $F(x, x^{q^\alpha+1}) = 0$.)

22.4. Perturbed MI

Since we have seen that the original MI-scheme is insecure, there was proposed a perturbed version of it which we describe now.
We write $C^* = (C_1^*, \ldots, C_n^*)$ such that we have

$$\begin{array}{ccc} \mathbb{F}_q^n & \xrightarrow{(C_1^*,\ldots,C_n^*)} & \mathbb{F}_q^n \\ \uparrow & & \uparrow \\ \mathbb{F}_{q^n} & \xrightarrow{C^*} & \mathbb{F}_{q^n} \end{array}$$

Thus, we write

$$q_i(x) = \begin{cases} C_i^*(x) + f_i(v(x)) & 1 \le i \le n \\ g_i(x) & n < i \le n + a \end{cases}$$

where $f = (f_1, \ldots, f_n)$ are random quadratic polynomials in r variables (and r is small), $v = (v_1, \ldots, v_r)$ are linear polynomials in n variables and $g_{n+1}, \ldots, g_{n+a}$ are random quadratic polynomials in n variables.
In this perturbed version we also have to invert the map $Q = (q_1, \ldots, q_{n+a})$, i.e. for given $y = Q(x)$ find $x \in \mathbb{K}^n$.

We only use $q_1, \ldots, q_n$ such that $(q_1(x), \ldots, q_n(x)) = (y_1, \ldots, y_n)$ and ignore the indices $n+1, \ldots, n+a$. This means we have to solve $q_i(x) = y_i$ for $1 \leq i \leq n$. Since $q_i(x) = C_i^*(x) + f_i(v(x))$ and there exists $h = (q^\alpha + 1)^{-1} \mod (q^n - 1)$ with $(C^*(x))^h = x$, we can write $y_i - f_i(v(x)) = C_i^*(x)$ or equivalently $(y - f(v(x))) = C^*(x)$ such that we obtain $(y - f(v(x)))^h = (C^*(x))^h = x$. The remaining problem is, that we do not know $f(v(x))$. But for $q = 2$ and small r, there are only 2^r possibilities such that we can get it by sampling.

23. Oil and Vinegar Schemes and Rainbow

As there are various MQ-problems and we discussed the ones in "Big Fields" such as Matsumoto-Imai (MI) or perturbed Matsumoto-Imai (we did not discuss Hidden Field Equations (HFE)), we now take a look at a "Single Field" and especially at the (Unbalanced) Oil and Vinegar Scheme and at the Rainbow Scheme.

23.1. General Oil and Vinegar Scheme

The Oil and Vinegar Scheme has a similar construction as MI. The secret key consists of the two private maps $S : \mathbb{F}_q^n \to \mathbb{F}_q^n$ and $Q : \mathbb{F}_q^n \to \mathbb{F}_q^o$ where S is an (invertible) linear map and Q is the oil and vinegar map consisting of *o oil and vinegar polynomials*. An oil and vinegar polynomial $\overline{q} \in \mathbb{F}_q[y_1, \ldots, y_o, x_1, \ldots, x_v]$ looks like

$$\overline{q} = \sum_{i=1}^{o} \sum_{j=1}^{v} a_{ij} y_i x_j + \sum_{i=1}^{v} \sum_{j=1}^{v} b_{ij} x_i x_j + \sum_{i=1}^{o} c_i y_i + \sum_{j=1}^{v} d_j x_j + e,$$

where the coefficients $a_{ij}, b_{ij}, c_i, d_j, e$ are chosen randomly. We write

$$Q(y_1, \ldots, y_o, x_1, \ldots, x_v) = (q_1, \ldots, q_o).$$

(Remark: It is $n = o + v$.)
The public key is the map $P = Q \circ S : \mathbb{F}_q^n \to \mathbb{F}_q^o$, the secret key Q, S.
The verification equation is $P(w) \stackrel{?}{=} z$. Signing is done in the following way: For $z = (z_1, \ldots, z_0) \in \mathbb{F}_q^o$ it is $q_i(y_1, \ldots, y_o, x_1, \ldots, x_v) = z_i$ for $i = 1, \ldots, o$. Now, fix some randomly chosen $(x_1^*, \ldots, x_v^*) \in \mathbb{F}_q^v$ and solve the linear system $q_i(y_1, \ldots, y_o, x_1^*, \ldots, x_v^*) = z_i, i = 1, \ldots, o$. With high probability we obtain a solution $(y_1^*, \ldots, y_o^*) \in \mathbb{F}_q^o$. Then $S^{-1}(y_1^*, \ldots, y_o^*, x_1^*, \ldots, x_v^*) = w$ is the signature for the message z.
The original Oil and Vinegar Scheme was proposed by Patarin in 1997 as balanced version with $o = v$.

23.2. Cryptanalysis of Balanced Oil and Vinegar Scheme

The Balanced Oil and Vinegar Scheme was cryptanalyzed by Kipnis and Shamir in 1998. The set $O = \{(a_1, \ldots, a_o, 0, \ldots, 0) : a_i \in \mathbb{F}_q\}$ will be important therein.

Let $Q = (q_1, \ldots, q_o)$ be the Oil and Vinegar map. If the characteristic of $\mathbb{F}_q$ is not 2, we can write the quadratic part of the polynomial q_i as matrix $Q_i = \begin{pmatrix} 0 & B_{i_1} \\ B_{i_1}^T & B_{i_2} \end{pmatrix}$, such that $q_i(x) = x^T Q_i x$. One can observe that $u_1^T Q_i u_2 = 0$ for all $u_1, u_2 \in O$.
Writing the quadratic part of the publicly known polynomial p_i as a matrix P_i we also have $P_i = S^T Q_i S$ because of $P = Q \circ S$. Again, we obtain $w_1^T P_i w_2 = 0$ for all $w_1, w_2 \in S^{-1}(O)$.
Suppose it is possible to find a basis $(b_1, \ldots, b_o)$ of $S^{-1}(O)$, we prolong this to a basis $(b_1, \ldots, b_o, b_{o+1}, \ldots, b_n)$ of $\mathbb{F}_q^n = \mathbb{F}_q^{o+v}$. (There exists a probabilistic algorithm to do this.)
Writing this basis as a matrix $B = (b_1 | \ldots | b_n)$, we have a symmetric $P_i' = B^T P_i B = \begin{pmatrix} 0 & * \\ * & * \end{pmatrix}$ that corresponds to quadratic polynomials p_i' which are also Oil and Vinegar polynomials. Concluding, with $P' = (p_1', \ldots, p_n')$ we get an equivalent key (P', B^{-1}), since $P' = P \circ B \implies Q \circ S = P = P' \circ B^{-1}$. Thus we replace $Q \circ S$ by $P' \circ B^{-1}$.

23.3. Unbalanced Oil and Vinegar Scheme

As an answer to their cryptanalysis of the Balanced Oil and Vinegar Scheme, Kipnis, Patarin and Goubin in 1999 proposed the Unbalanced Oil and Vinegar Scheme in which we have $v > o$, $v \approx o$. A generalized attack is given in time $O(o^u q^{v-o-1})$. Therefore, $v - o$ should not be too small because of security, but in the same it should not be too big because of efficiency. (Side-Remark: $v = 20$ seems secure.)

23.4. Rainbow

Using the construction of the Unbalanced Oil and Vinegar Scheme, Ding and Schmidt in 2005 proposed Rainbow, a multilayer unbalanced oil and vinegar scheme. We will give the most important definitions first before presenting the Rainbow scheme.
Let $S = \{1, \ldots, n\}$ and $v_1, \ldots, v_u$ with $0 < v_1 < v_2 < \ldots < v_u = n$. Further, let $V_l = \{1, \ldots, v_l\}$ such that $v_i = |V_i|$ and $V_1 \subset V_2 \subset \ldots \subset V_u = S$. Let $O_i = V_{i+1} \setminus V_i$ for $i = 1, \ldots, u-1$ and $o_i = v_{i+1} - v_i$ such that $o_i = |O_i|$. It is $V_i \cap O_i = \emptyset$ and $V_{i+1} = V_i \cup O_i$. (Now you can think of O_i as the oil and V_i as the vinegar variables in layer i.)
Let P_l be a set of polynomials of the form

$$\sum_{i \in O_l, j \in V_l} \alpha_{ij} x_i x_j + \sum_{i,j \in V_l} \beta_{ij} x_i x_j + \sum_{i \in V_{l+1}} \gamma_i x_i + \eta \in \mathbb{F}_q[x_1, \ldots, x_n]$$

for $l = 1, \ldots, u-1$.
Now, for all $i = 1, \ldots, u-1$ randomly choose o_i polynomials from P_i and construct the central map $Q : \mathbb{F}_q^n \to \mathbb{F}_q^{n-v_1}$.
Here is an overview of the oil and vinegar variables on each layer:

Layer	Vinegar	Oil
1	$\{x_1, \ldots, x_{v_1}\}$	$\{x_{v_1+1}, \ldots, x_{v_2}\}$
2	$\{x_1, \ldots, x_{v_1}, x_{v_1+1}, \ldots, x_{v_2}\}$	$\{x_{v_2+1}, \ldots, x_{v_3}\}$
$\vdots$	$\vdots$	$\vdots$
$u-1$	$\{x_1, \ldots, x_{v_{u-1}}\}$	$\{x_{v_{u-1}+1}, \ldots, x_n\}$

The secret key consists of the three private maps $Q, T : \mathbb{F}_q^{n-v_1} \to \mathbb{F}_q^{n-v_1}$ and $S : \mathbb{F}_q^n \to \mathbb{F}_q^n$ whereas $P = T \circ Q \circ S$ is the public key.
Signing is done similar to the Oil and Vinegar Scheme: For the first layer assign random values to the vinegar variables $(x_1^*, \ldots, x_{v_1}^*)$ and solve the $o_1 \times o_1$-linear system to obtain $(x_{v_1+1}^*, \ldots, x_{v_2})$. Proceed iteratively for the other layers... At the end you will have $(x_1^*, \ldots, x_n^*)$. Then $w = S^{-1}(x_1^*, \ldots, x_n^*)$ is the signature.
A signature can be verified using the equation $P(w) = z$.
Note: If there is no solution for the oil variables on one of the layers, return to the layer above and choose other random values for the vinegar variables.

23.5. "Parameters" of Rainbow

For most purposes, some people chose $\mathbb{F}_q = \mathbb{F}_{256} = \mathbb{F}_{2^8}$ as well as $n = 37$ and $u = 5$ with

i	v_i	o_i
1	10	10
2	20	4
3	24	3
4	27	10
5	37	

With this we have $Q : \mathbb{F}_{256}^{37} \to \mathbb{F}_{256}^{27}$, a public key size of 16KB, a secret key size of 16KB, a document size of 216 bits and a signature size of 296 bits.

Index

(n, k)-code, 39
(n, k, d)-code, 39
(t, ε)-collision-resistant, 15
(t, q, ε)-EU-CMA, 15
$C^{\perp}$, 40
(γ-)CVP, 25
(γ-)SIVP, 25
(γ-)SVP, 25

Affine map, 50
Ajtai, 30
    Construction, 30
Ajtai's function, 30, 33, 36
Attack
    Know-Plaintext-, 53
Attacks
    Decoding, 42
    Structural, 42
Authentication path, 14, 20

Base states, 7
basis, 22
Bilinear form, 52
Bilinear map, 52
Bilinear relation, 53

Central map Q, 50
CFS, 43, 46, 49
CFS signature, 43
Characteristic, 53
Closest vector problem (CVP), 25
Coding theory, 38
Collision-resistant, 15, 33
Continued fractions, 9, 10
Coprime, 10
Coronado: Secret key generation, 18
correct, 39, 40
correction capacity, 39
Cosine, 28
Courtois, 43
CVP, 25, 33

decoding algorithm, 39
Decoding attacks, 42
determinant, 23
Determinant of full-rank lattice, 23
Difference of spans, 22
Digital signatures, 11
dimension, 22
Ding, 55
double-circulant codes, 45
double-circulant matrix, 45
dual code $C^{\perp}$, 40

encoding algorithm, 39
EU-CMA, 12
Euler's totient function, 10
Extended Euclidean Algorithm, 32, 52

Factoring problem, 6
Field, 9
Finiasz, 43
Finite field, 50, 51
    Construction of a, 51
Fourier transformation, 9
Frobenius automorphism, 53
full-rank, 23
Full-rank lattices, 23

Gauß reduction algorithm, 26, 29
Gauß-reduced basis, 29
General linear group, 23
generator matrix, 38, 41, 48
generator matrix of $C^{\perp}$, 40
Geometrical interpretation of the determinant, 24
GGH, 34
GGH public key, 34
GGH secret keys, 34
GGH signature, 34
Gilbert Varshamov Bound, 39
Goldreich, 34
Goldwasser, 34
Goppa codes, 42, 43

Goubin, 55
Gram-Schmidt coefficients, 27
Gram-Schmidt orthogonalization (GSO), 27
Greatest common divisor, 6
Grover's algorithm, 44

Halevi, 34
Hamming distance, 38
Hamming weight, 38
Hard instances, 30, 50
Hash-function, 31, 33, 36, 43, 46, 49, 51
Hash-functions, 13
Hermite Normal Form (HNF), 34
Hermite-SVP, 26
HFE, 54
Hidden Field Equations (HFE), 54
HNF, 34

i-th successive minimum, 24
IBS, 46
IBS practical values, 48
IBS protocol, 47
IBS public key, 46
IBS secret key, 46, 47
ID-PKC, 45
Ideal lattice, 33
Identity-based public-key cryptography (ID-PKC), 45
Identity-based signature schemes (IBS), 46
Imai, 51
Information Set Decoding, 49
invertible matrix, 48
Inverting Q, 52
    Goal, 52
Irreducible, 32, 36, 51
ISD, 49
ISD algorithm, 42, 45

Key escrow, 46
Kipnis, 54, 55
Known-plaintext-attack, 53

Lattice, 31
lattice, 22
Lattice basis, 22
Lattice dimension, 22
LD-OTS public key, 11
LD-OTS secret key, 11
Leafs of Merkle tree, 14
Light-weight cryptography, 50
linear (n, k)-code, 39
LM-Merkle, 37
LM-OTS, 36
LM-OTS public key, 36
LM-OTS secret keys, 36
LM-OTS signature, 36
Low-resource devices, 45
LT-OTD signature, 11
Lyubashevsky, 36

Matsumoto, 51
Matsumoto-Imai, 51
McEliece, 40, 43, 48
McEliece ciphertext, 41, 48
McEliece public key, 41, 48
McEliece secret key, 41, 48
MI, 51
Micciancio, 36
minimal distance, 39
Minimum distance, 45
minimum distance, 24
Monic, 32, 36
MSS public key, 14
MSS secret key, 14
MSS signature, 14
Multilayered unbalanced oil and vinegar scheme, 55
Multivariate
    General notation, 50
    Invert, 50
    Overview of maps, 51
Multivariate ciphertext, 51
Multivariate public key, 51
Multivariate secret key, 51
Multivariate signature, 51
Multivariate system, 50

New left nodes, 20
New right nodes, 21
Nguyen, 34
Niderreiter, 48
Niederreiter, 41, 43
Niederreiter ciphertext, 41, 48
Niederreiter public key, 41, 48
Niederreiter secret key, 41, 48
Normalize, 27
    Angle, 28
NP-complete, 29, 40, 50
NP-hard, 29, 50
NTRU, 36

Oil and Vinegar, 54
    Balanced, 54
    General, 54
    Multilayered Unbalanced, 55
    Unbalanced, 55
Oil and Vinegar map, 55
oil and vinegar polynomials, 54, 55
Oil and Vinegar public key, 54
Oil and Vinegar secret key, 54
One-way function, 11
Orthogonal projections, 27
Overview of maps, 51

Parallelepiped, 34
parallelepiped, 23
parity check matrix, 40, 41
Patarin, 52, 54, 55
Patterson algorithm for QD Goppa codes, 44
Period, 6, 8
permutation matrix, 48
Perturbed Matsumoto-Imai, 53
Perturbed MI, 53
Polynomial, 32, 36, 51
Pseudo-random number generator (PRNG), 18
Public-key cryptography, 5

QD Goppa codes, 44
Quantum computers, 7
Quantum registers, 9
Qubits, 7

Rainbow, 54, 55
    Overview about layers, 55
Rainbow public key, 56
Rainbow secret key, 56
Rainbow signature, 56
rate, 39
Regev, 34
Residue class, 6
Residue classes, 32
Ring, 32, 36
Root of unity, 9

scalar product, 39
Schmidt, 55
Sendrier, 43
set of all bases, 23
Shamir, 45, 54
Shannon, 38
Shor's algorithm, 8
Shortest independent vector problem (SIVP), 25
Shortest vector problem (SVP), 25
Singleton Bound, 39
SIVP, 25
size-reduced, 28
Span, 22
Stern, 44, 49
Stern Identification Scheme, 44, 49
Stern protocol, 44, 49
Stern public key, 44, 49
Stern secret key, 44, 49
Structural attacks, 42
Support Splitting Algorithm, 49
SVP, 25
    Hermite-SVP, 26
    NP-hard, 29
Swap, 28
syndrome, 40
syndrome decoding algorithm, 40, 43
Syndrome decoding problem, 44

Translated parallelepiped, 34
Tree chaining, 18

Tree chaining signature, 19

Unbalanced Oil and Vinegar Scheme, 55
unimodular, 23
Uniqueness of lattice basis, 22

Véron, 45
Véron Identification Scheme, 45
vectorial subspace, 38
Volume of the parallelepiped, 23

Winternitz check sum, 17
Winternitz public key, 17
Winternitz secret key, 17
Winternitz signature, 17
Worst case to average case reduction, 30

Zero divisor, 9
Zero-knowledge, 44